# Position Paper: Reasoning About Domains with PDDL

**Alexander Shleyfman, Erez Karpas**
Faculty of Industrial Engineering and Management
Technion — Israel Institute of Technology

## Abstract

One of the major drivers for the progress in scalability of automated planners has been the introduction of the Planning Domain Definition Language (PDDL) and the International Planning Competition (IPC). While PDDL provides a convenient formalism to describe planning problems, there is a significant gap with regards to describing domains. Although PDDL is split into a domain description and a problem description, the domain description is not enough to specify a domain completely, as it does not constrain the possible problems in the domain. For example, there is nothing in the BLOCKSWORLD PDDL domain description which says that a block can not be on top of itself in the initial state. In this position paper, we argue that PDDL domains should be extended to incorporate a new section which constrains possible problems in the domain. We argue that such an extension can be based on first-order logic, and describe several use cases where this extension might be of use. We also provide some preliminary empirical results of one way for automatically extracting such constraints based on mutual exclusion.

## Introduction

The domain-independent planning community has made significant progress scaling up planners, allowing them to address bigger and more complicated problem instances. One of the major drivers for this progress has been the introduction of the International Planning Competition (IPC), with its standard language for describing planning problems — PDDL, the Planning Domain Definition Language (McDermott 2000). The PDDL language was further extended to support additional features, which were introduced in later iterations of the IPC (Fox and Long 2003; Edelkamp and Hoffmann 2004; Gerevini and Long 2005).

PDDL splits the definition of a planning problem into two parts: domain and problem. The domain describes the types of objects this domain deals with, along with schemata for the predicates used to describe the state of the world and the operators used to change it. The problem describes the specific objects in the world in this problem instance, as well as the initial state and the goal. Typically, a domain in the IPC is defined by a single PDDL domain description (usually in a separate file), and a random problem instance generator[1]. Planners are then evaluated based on their performance on a set of problem instances generated by the problem generator.

While this is a reasonable way to evaluate how well planners solve planning *problems*, we claim that it is extremely difficult to reason over a *domain*. For example, consider the well known BLOCKSWORLD domain, which features the predicate ON$(x, y)$, indicating that block $x$ is directly on top of block $y$. We would like to be able to prove that a block can never be on top of itself. This is fairly easy to do using techniques such as relaxed reachability. However, relaxed reachability takes an initial state as input, and the initial state is only described in the PDDL problem. In fact, there is nothing preventing us from generating an instance of BLOCKSWORLD in which ON$(A, A)$ does appear in the initial state. Thus, in order to be able to prove that a block is never on top of itself, we would need some *explicit* description of the fact that a block is never on top of itself in the initial state of any valid problem instance. Currently, this knowledge is only *implicit* from our understanding of the meaning of the domain.

Previous work (Helmert 2003) has defined a domain as an infinite set of grounded planning problems. While this definition is good enough to theoretically analyze the complexity of planning in a domain, it does not consider the issue of representation. In this position paper, we argue that the PDDL language needs to be further extended, in order to allow for automated reasoning about *domains*, rather than only single problem instances. We argue that such an extension can be based on first-order logic, and describe several use cases where this extension might be of use. We also provide some preliminary empirical results where we can identify what are *probably* domain-level mutual exclusion (mutex) groups, providing some automated support for encoding our suggested constraints.

## Background

We begin with a brief review of PDDL. For the full details, we refer the reader to the various papers describing the different versions of PDDL (McDermott 2000; Fox and Long 2003; Edelkamp and Hoffmann 2004; Gerevini and Long

[1]Some domains have a separate domain description for each problem instance. We will address this issue in the final discussion.

2005). As previously mentioned, PDDL divides the definition of a planning problem into two parts: the domain, and the problem, which typically are contained in two different files. The division allows for the same domain file to be used with multiple problem files.

A PDDL domain consists of a description of the possible types of objects in the world. A type $t$ can inherit from another type $s$, so that all objects with type $s$ are also of type $t$. While there is a small controversy regarding whether the type hierarchy must form a proper tree, or can be a graph, this issue is irrelevant for the purposes of this paper. The domain also consists of a set of constants, which are objects which appear in all problem instances of this domain.

The second part of the domain description is a set of predicates. Each predicate is described by a name and a signature, consisting of an ordered list of types. Given a set of objects, we can *ground* the given predicates, yielding a set of propositions which describe the state of the world. Note, however, that these objects are only given as part of the problem description, and not in the domain description. The domain also describes a set of derived predicates, which are predicates associated with a logical expression. The idea is that the value of each derived predicate is computed automatically by evaluating the logical expression associated with it.

Finally, the domain description consists of a set of operators. Each operator is also described by a name, a signature, a precondition, and an effect. The signature is now an ordered list of named parameters, each with a type. The precondition is a logical formula, whose basic building blocks are the above mentioned predicates, combined using the standard first order logic logical connectives. We remark that the predicates can only be parametrized by the operator parameters, the domain constraints, or, if they appear within the scope of a forall or exists statement, by the variable introduced by the quantifier. The effect of the operator is similar, except that it described a partial assignment, rather then a formula, and thus can not contain any disjunctions. An operator can also be grounded given a set of objects, yielding grounded actions.

A PDDL problem is much simpler than the domain. It consists of a set of objects, each associated with a type (if a type is not specified, the object is assumed to be of a default type), and a description of the initial state and the goal. The initial state is described by the list of propositions (grounded predicates) that are true in it, where any proposition that is not listed it assumed to be false. The goal is also a logical expression, similarly to the precondition of an operator, except that it can refer to all objects in the problem instance. Although the goal can be an arbitrarily complex logical expression, in most existing planning benchmarks domains, it is a simply conjunction of positive propositions. In the rest of this paper, we will assume the goal takes this simple form, and discuss more complex goals in the conclusion.

As mentioned above, a domain can be grounded given a set of objects, which are described in the problem. Most modern planners start by grounding the given planning problem, and operate on the grounded problem description. However, if our intention is to reason over a domain, this approach is not practical, as there is no single problem to ground over. In the next section, we present our proposal to extend PDDL to allow some reasoning over a domain, even when a problem instance is not given.

```
(forall (?x) (not (init (on ?x ?x))))

(forall (?x ?y ?z) (implies
(and (init (on ?y ?x)) (init (on ?z ?x)))
(= ?z ?y)))

(forall (?x) (or
(init (on-table ?x))
(exists (?y) (init (on ?x ?y)))))

(forall (?x) (not (goal (on ?x ?x))))

(forall (?x ?y ?z) (implies
(and (goal (on ?y ?x)) (goal (on ?z ?x)))
(= ?z ?y)))
```

Figure 1: BLOCKSWORLD Domain Constraints

## Extending PDDL

The heart of our proposed extension to PDDL is to add *constraints* about the problem to the domain description. Following the BLOCKSWORLD example from the introduction, we could specify that in the initial state of any legal instance of BLOCKSWORLD, no block is on top of itself.[2] We could then use a lifted version of relaxed reachability analysis to infer that no block can ever be on top of itself.

Specifically, we propose to add another section to the PDDL domain description, which will consist of a set of constraints. Each constraint will be a first order logic statement, which can refer to domain constants, and, of course, to variables introduced by each quantifier within its scope. However, the basic building blocks will not be predicates, but rather predicates perpended with a modal operator, specifying if this refers to the initial state or the goal. One caveat is that we can not check whether some proposition if false in the goal, as the goal is only a partial state. We also explicitly allow the usage of the (object) equality predicate. As the following examples will show, it is quite useful.

The interpretation of these constraints is, naturally, as constraints over a problem description. We can treat each problem as specifying a full initial state, and a partial goal state (as we assume the goal only describes the propositions we want to be true). Thus, we can evaluate each constraint, and check whether a given problem satisfies it.

Figure 1 shows how our extension can be applied to BLOCKSWORLD. The first constraint states that a block is never on top of itself in the initial state. The second constraint states that there can be at most one block on top of another block (i.e., if $y$ and $z$ are both on top of $x$, then they must be the same block). The third constraint states that every block must be on top of another block or on the table in

[2]These are different than the constraints introduced in PDDL 3.0 (Gerevini and Long 2005), which constrain possible *plans* for a given problem.

the initial state. Finally, the last two constraints are similar to the first two, except they are applied to the goal.

Another example highlights the differences between two different versions of the LOGISTICS domain: the one used in the first IPC (1998) and the one used in the second IPC (2000). Even though the PDDL domain description was the same in both competitions, LOGISTICS-98 is still much harder to solve than LOGISTICS-2000. This is because in the instances generated for second IPC, there was an implicit constraint, that there is exactly one truck in each city. This constraint is shown in Figure 2.

## Use Cases

So far, we have only proposed an extension to PDDL, without explaining why we believe such an extension is useful. In this section we provide several use cases where our proposed extension can be useful. We remark that we have not implemented any of these ideas, we simply claim that these can be the subject of future work.

### Learning and Using Domain Control Knowledge

There has been a significant body of work on learning, and using, domain control knowledge. While a full review of all the relevant literature is beyond the scope of this paper, we review some influential works in this area. First, the original STRIPS system had a macro learning component, which attempted to generalize successful plans from one problem to others (Fikes, Hart, and Nilsson 1972). This is, in fact, an example of explanation based learning (EBL) (e.g., (Mooney and Bennett 1986; Minton 1990)), where a system typically look at a single example and attempts to generalize it.

Another example is the TLPlan planner (Bacchus and Kabanza 2000), which was able to exploit manually coded domain-specific control knowledge expressed in a temporal logic. Later work tried to learn such rules automatically (Yoon, Fern, and Givan 2008). In fact, the learning track in the international planning competition (IPC), introduced in 2011 (Fern, Khardon, and Tadepalli 2011), focuses on learning domain control knowledge. In the learning track, each competitor is given access to a PDDL domain file and a random problem generator. The competitor is then given a very long time to produce a domain control knowledge (DCK) file, which the planner can then use to solve new problems in the domain, with the intent that the DCK will help the planner improve its performance.

With the way this track is set up, the best type of guarantee that can be provided is a probably approximately correct (PAC) (Valiant 1984) style guarantee, i.e., that there is a high probability that the learner has learned something that is fairly good. However, there is no way to guarantee that the learned domain control knowledge will work, because there is no characterization of all possible instances in the domain, but only a sample of problem instances. Adopting the proposed extension to PDDL will allow learners to *prove* something about what they are learning.

For example, suppose we wanted to make the Fast Downward translator (Helmert 2009) more efficient by learning what propositions are grouped together into a finite-domain variable. We might be able to learn, for example, that the location of a truck in LOGISTICS is always a mutex group, and can thus be used to create a finite domain variable. In fact, since the translator looks for invariants in a lifted way in the domain and then generates possible mutex groups from invariants which have a single matching fact that is true in the initial state, it is relatively straightforward to so, as our preliminary empirical results demonstrate. Of course, this is only possible if we know that AT$(T, L)$ has exactly one true proposition for each given truck $T$ in the initial state — something which is easily described using our proposed PDDL extension.

Similar invariants can be seen in the BLOCKSWORLD domain. The same as trucks, blocks each are represented as single finite domain variables, which are generated using the invariants founded in the domain description, and the predicates in the initial state. These mutexes however, are not enough to randomly generate a "realistic" BLOCKSWORLD problem. As we mentioned before, a single block can not be placed on itself, and thus there are no predicates of the form ON$(x, x)$ in the initial state. However, consider a problem with two blocks $A$ and $B$, where block $A$ is placed on top of block $B$ and block $B$ is placed on top of block $A$. It is easy to see that this position satisfies the condition described in the previous section, but in the same time, it's both "unrealistic" and unsolvable, given the blocks $A$ and $B$ have some other positions in the goal description. Even more so, this "ouroboros"[3] of a sort can be extended to a cycle of an arbitrary length, making this condition hard to detect without some recursive logical formula. Thus, our proposed extension must be able to support recursive formulas, to be able to express these restrictions.

### Generalized Planning

A somewhat similar use case occurs in generalized planning. In generalized planning, the objective is to generate a controller which can solve all possible problems from a given planning domain. Examples of work on generalized planning include generating plans with loops and branching (Srivastava, Immerman, and Zilberstein 2011) and finite state controllers (Bonet, Palacios, and Geffner 2009; Aguas, Celorrio, and Jonsson 2016). Again, the issue is that with no formal specification of a domain, it is impossible to prove that a controller will solve all problems in a domain.

On the other hand, using our proposed PDDL extension, it is very easy (in theory) to use the following scheme. First, call a generalized planner on a given set of problems in the domain of interest. Second, verify if the resulting controller solves all possible problems in the domain. If the answer is yes, we have a controller that can solve all problems in the domain. Otherwise, generate a counter example, add it to the given set of problems, and repeat. Of course, the problem of verifying if the given controller works for all possible problems in the domain, and generating a counter example if it does not is undecidable (as we can generate a domain that corresponds to a Turing machine, and each problem corre-

[3]A serpent eating its own tail.

```
(forall (?c - city ?s ?t - truck) (implies
   (and (exists (?l - location) (and (init (in-city ?l ?c)) (init (in ?t ?l))))
        (exists (?l - location) (and (init (in-city ?l ?c)) (init (in ?s ?l)))))
   (= ?s ?t)))
```

Figure 2: LOGISTICS-2000 Additional Constraint

sponds to a given instance terminating). Nevertheless, efficient (incomplete) termination analyzers do exist, thus allowing us to hope this idea might work in practice on some domains of interest.

### Almost Automatic Random Problem Generators

When creating a new domain in PDDL, the burden of specifying which problems are legal and which are not falls to the problem generator. For example, the problem generator for BLOCKSWORLD will never generate a problem in which on$(A, A)$ appears in the initial state. However, this knowledge is part of the problem generator's code. On top of this, the problem generator provides some distribution on the problems.

With our proposed extension, the first part of the random problem generator's job could be automated. The only implementation necessary in a random problem generator would be just the random part — the distribution.

While we believe this would be beneficial by itself, this also has the potential of enabling bootstrapping approaches (Arfaee, Zilles, and Holte 2010), where larger and larger problem instances must be generated. Of course, the issue of where the distribution comes from is still a critical component of such an approach, which is beyond the scope of our proposed PDDL extension.

### State Estimation

Finally, another use case comes from the combination of planning with real world sensing. Consider, for example, a camera looking at a BLOCKSWORLD scene. The camera, along with the image processing and object recognition software that looks at its output, will typically produce a set of real-world coordinates for the position of each block. These coordinates will typically have some error associated with them, due to sensor noise, lighting conditions, probabilistic image processing algorithms, and more.

A state estimator will look at the history of these measurements to produce the symbolic description of the current state. Without telling the state estimator that a block can only be on top of one other block, we might end up with states containing both on$(A, B)$ and on$(A, C)$. However, if our state estimator was able to infer mutual exclusion invariants for the domain, it could reject samples which violate these constraints, yielding more accurate state estimates.

## Case Study: Discovering Domain Mutexes

As a first step to demonstrate reasoning over a domain, rather than over individual problems, we used the Fast Downward translator (Helmert 2009), in order to examine invariant candidates in PDDL domains from IPC benchmarks. As previously mentioned, the Fast Downward translator identifies lifted invariant candidates looking only at the PDDL domain. For example, the translator identifies that for a given truck $T$, the number of locations $L$ for which AT$(T, L)$ holds does not increase for any applicable action. The translator then checks whether this invariant candidate generates a set of mutexes, by checking if the number of locations each truck $T$ is at in the initial state is 1 or less.

In this case study, we used the invariants discovered by the Fast Downward translator for each domain. For each invariant, we checked whether it always led to mutexes in all instantiations of the invariant in all problems. If so, then it is likely safe to add a problem constraint derived from this invariant to the domain. However, without an explicit extension to PDDL, we can never know that this is a true lifted mutex, or whether the random problem generator just happened to only generate problems where this invariant happened to lead to mutexes.

### Experimental Results

For our experiment we used the International Planning Competition benchmarks (IPC'98 – IPC'11), from which we excluded all the benchmarks that have more than one domain description file. In the relevant benchmarks we count the invariant candidates extracted by the Fast Downward translator, and check which of those invariant lead to mutex groups, and which did not (due to the fact that the number of initial state propositions participating in these invariants exceeded 1). The results are presented in Table 1. Note that there are no domain invariants that have not been grounded to a mutex group due to the absence of the appropriate initial states facts.

Most of the invariants in these benchmarks are either always mutex groups, or always *overcrowded* – there are at least 2 propositions in the initial state that participate in that invariant. However, there are some invariants that are mixed, that is, lead to mutex groups in some cases, and are overcrowded in others. Detailed analysis shows that this happens mostly due to the fact that there is a smaller invariant that is contained in a larger one. For example, in the LOGISTICS domain all the locations of a given truck $T$ constitute an invariant, but all the locations of all the trucks are also an invariant of the domain. The latter invariant leads to a mutex group only in the case where there is exactly one truck in the problem. Thus, mixed invariants can be seen grounded in the small problems of the domains, but getting overcrowded in the large ones.

## Conclusion

In this paper, we have proposed an extension to PDDL which will allow for automated formal reasoning about domains. This extension will make no difference to the task of solving a single planning problem, with the possible exception of first validating the given problem instance. However, as we

| Domain | Inv | Pure | Over | Mixed |
|---|---|---|---|---|
| AIRPORT-ADL | 8 | 6 | 0 | 2 |
| ASSEMBLY | 0 | 0 | 0 | 0 |
| BARMAN-OPT11-STRIPS | 3 | 3 | 0 | 0 |
| BARMAN-SAT11-STRIPS | 3 | 3 | 0 | 0 |
| BLOCKS | 3 | 3 | 0 | 0 |
| DEPOT | 5 | 4 | 1 | 0 |
| DRIVERLOG | 2 | 2 | 0 | 0 |
| ELEVATORS-OPT11-STRIPS | 3 | 3 | 0 | 0 |
| ELEVATORS-SAT11-STRIPS | 3 | 3 | 0 | 0 |
| FLOORTILE-OPT11-STRIPS | 5 | 4 | 1 | 0 |
| FLOORTILE-SAT11-STRIPS | 5 | 4 | 1 | 0 |
| FREECELL | 7 | 6 | 1 | 0 |
| GRID | 7 | 5 | 2 | 0 |
| GRIPPER | 3 | 3 | 0 | 0 |
| LOGISTICS00 | 1 | 1 | 0 | 0 |
| LOGISTICS98 | 1 | 1 | 0 | 0 |
| MICONIC-SIMPLEADL | 1 | 1 | 0 | 0 |
| MICONIC | 1 | 1 | 0 | 0 |
| MOVIE | 0 | 0 | 0 | 0 |
| MPRIME | 3 | 3 | 0 | 0 |
| MYSTERY | 3 | 3 | 0 | 0 |
| NO-MPRIME | 2 | 2 | 0 | 0 |
| NO-MYSTERY | 3 | 3 | 0 | 0 |
| NOMYSTERY-OPT11-STRIPS | 2 | 2 | 0 | 0 |
| NOMYSTERY-SAT11-STRIPS | 2 | 2 | 0 | 0 |
| OPENSTACKS | 8 | 5 | 3 | 0 |
| OPTICAL-TELEGRAPHS | 7 | 6 | 1 | 0 |
| PARKING-OPT11-STRIPS | 4 | 3 | 1 | 0 |
| PARKING-SAT11-STRIPS | 4 | 3 | 1 | 0 |
| PEGSOL-OPT11-STRIPS | 2 | 1 | 1 | 0 |
| PEGSOL-SAT11-STRIPS | 2 | 1 | 1 | 0 |
| PHILOSOPHERS | 7 | 6 | 1 | 0 |
| PIPESWORLD-NOTANKAGE | 2 | 1 | 1 | 0 |
| PSR-LARGE | 0 | 0 | 0 | 0 |
| PSR-MIDDLE | 0 | 0 | 0 | 0 |
| ROVERS | 12 | 6 | 3 | 3 |
| SATELLITE | 2 | 1 | 0 | 1 |
| SCANALYZER-OPT11-STRIPS | 0 | 0 | 0 | 0 |
| SCANALYZER-SAT11-STRIPS | 0 | 0 | 0 | 0 |
| SOKOBAN-OPT11-STRIPS | 3 | 2 | 1 | 0 |
| SOKOBAN-SAT11-STRIPS | 3 | 2 | 1 | 0 |
| STORAGE | 3 | 3 | 0 | 0 |
| TIDYBOT-OPT11-STRIPS | 3 | 3 | 0 | 0 |
| TIDYBOT-SAT11-STRIPS | 3 | 3 | 0 | 0 |
| TPP | 5 | 5 | 0 | 0 |
| TRANSPORT-OPT11-STRIPS | 2 | 2 | 0 | 0 |
| TRANSPORT-SAT11-STRIPS | 2 | 2 | 0 | 0 |
| TRUCKS | 3 | 3 | 0 | 0 |
| VISITALL-OPT11-STRIPS | 1 | 1 | 0 | 0 |
| VISITALL-SAT11-STRIPS | 1 | 1 | 0 | 0 |
| WOODWORKING-OPT11-STRIPS | 7 | 6 | 1 | 0 |
| WOODWORKING-SAT11-STRIPS | 7 | 6 | 1 | 0 |
| ZENOTRAVEL | 2 | 2 | 0 | 0 |

Table 1: Inv – number of invariants in the domain; Pure – number of invariants that are always grounded; Over – number of invariants that always have at least two predicates in the initial state; Mixed – number of invariants that sometimes are grounded, and sometimes have to many predicted in the initial state.

have illustrated in the previous section, such an extension will allow us to perform formal reasoning over a domain description, as well as provide a cleaner definition of what constitutes a planning domain.

While the focus of this paper has been on classical planning, our proposal becomes perhaps even more relevant in the context of non-deterministic planning. Specifically, finite state controllers are very useful with non-deterministic and partially observable planning problems, and state estimation is a must for realistic applications that involve sensing in a partially observable world.

The paper does not presume to provide the definitive, best possible, extension to PDDL. Two issue that were already mentioned are that some domains have a separate domain description for each problem instance, and that the goal can be a logical formula, not just a single conjunction. With regards to the first issue, this is usually the result of simplifying ADL (Pednault 1989) to STRIPS for the sake of planners that can not handle ADL. We argue this is not a real issue here, as reasoning over our proposed extension will require ADL-like reasoning (specifically, quantifiers). Furthermore, it is possible to perform reasoning over a domain using the complex ADL domain specification, and then planning using the simplified STRIPS version of the *given problem*.

The second issue, of complex goals, deserves further discussion. It could be possible to modify our proposed extension to PDDL to contain more general statements about the goal, such as "the goal entails $X$" or "the goal contains $X$ as a subexpression in a location specified by $y$". We are skeptical that such statements would be of use in modeling domains of interest to the planning community, and so we do not propose them here.

Finally, despite the issues mentioned above, we believe this paper serves as a starting point for a discussion about what exactly constitutes a domain, and on what the automated planning community can contribute on top of state-of-the-art automated planners.

## References

Aguas, J. S.; Celorrio, S. J.; and Jonsson, A. 2016. Generalized planning with procedural domain control knowledge. In *Proc. IJCAI 2016*, 285–293.

Arfaee, S. J.; Zilles, S.; and Holte, R. C. 2010. Bootstrap learning of heuristic functions. In *Proc. SoCS 2010*, 52–60.

Bacchus, F., and Kabanza, F. 2000. Using temporal logics to express search control knowledge for planning. *AIJ* 116(1-2):123–191.

Bonet, B.; Palacios, H.; and Geffner, H. 2009. Automatic derivation of memoryless policies and finite-state controllers using classical planners. In *Proc. ICAPS 2009*.

Edelkamp, S., and Hoffmann, J. 2004. PDDL2.2: The language for the classical part of the 4th International Planning Competition. Technical Report 195, Albert-Ludwigs-Universität Freiburg, Institut für Informatik.

Fern, A.; Khardon, R.; and Tadepalli, P. 2011. The first learning track of the international planning competition. *Machine Learning* 84(1-2):81–107.

Fikes, R. E.; Hart, P. E.; and Nilsson, N. J. 1972. Learning and executing generalized robot plans. *AIJ* 3:251–288.

Fox, M., and Long, D. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *JAIR* 20:61–124.

Gerevini, A., and Long, D. 2005. Plan constraints and preferences in PDDL3. Technical Report R. T. 2005-08-47, Dipartimento di Elettronica per l'Automazione, Università degli Studi di Brescia.

Helmert, M. 2003. Complexity results for standard benchmark domains in planning. *AIJ* 143(2):219–262.

Helmert, M. 2009. Concise finite-domain representations for PDDL planning tasks. *AIJ* 173:503–535.

McDermott, D. 2000. The 1998 AI Planning Systems competition. *AI Magazine* 21(2):35–55.

Minton, S. 1990. Quantitative results concerning the utility of explanation-based learning. *AIJ* 42(23):363–391.

Mooney, R. J., and Bennett, S. 1986. A domain independent explanation-based generalizer. In *Proc. AAAI 1986*, 551–555.

Pednault, E. P. D. 1989. ADL: Exploring the middle ground between STRIPS and the situation calculus. In *Proc. KR 1989*, 324–332.

Srivastava, S.; Immerman, N.; and Zilberstein, S. 2011. A new representation and associated algorithms for generalized planning. *AIJ* 175(2):615–647.

Valiant, L. G. 1984. A theory of the learnable. *CACM* 27(11):1134–1142.

Yoon, S.; Fern, A.; and Givan, R. 2008. Learning control knowledge for forward search planning. *JMLR* 9:683–718.

# Situated Planning for Execution Under Temporal Constraints

**Michael Cashmore, Andrew Coles**
King's College London

**Bence Cserna**
University of New Hampshire

**Erez Karpas**
Technion — Israel Institute of Technology

**Daniele Magazzeni**
King's College London

**Wheeler Ruml**
University of New Hampshire

**Abstract**

One of the original motivations for domain-independent planning was to generate plans that would then be executed in the environment. However, most existing planners ignore the passage of time during planning. While this can work well when absolute time does not play a role, this approach can lead to plans failing when there are external timing constraints, such as deadlines. In this paper, we describe a new approach for time-sensitive temporal planning. Our planner is aware of the fact that plan execution will start only once planning finishes, and incorporates this information into its decision making, in order to focus the search on branches that are more likely to lead to plans that will be feasible when the planner finishes.

## Introduction

One of the original motivations for domain-independent planning was for controlling robots performing complex tasks (Fikes and Nilsson 1971). The typical approach to controlling robots using a planner is to call the planner to generate a plan which solves the problem, and then execute that plan in the environment. This approach works well if the plan remains applicable regardless of when it is executed. However, if there are external timing constraints, such as deadlines which must be met, things become more complex. This is because we must take into account the *planning time*.

For example, in the Robocup Logistics League (RCLL) challenge (Niemueller, Lakemeyer, and Ferrein 2015), a team of robots must move workpieces between different machines that perform some operations on them, and fulfill some order with a deadline. This calls for using temporal planning, because we would like all robots to work in parallel, and actions have different durations. The typical approach would have the planner come up with a plan which would work had it been executed at time 0, and then execute this plan when the planner completes. Obviously, this might lead to missing the deadline, and thus, plan failure.

One simple approach to handling this problem is to use some estimate on how long planning will take, and adapt all the deadlines assuming plan execution would start when the planner finishes. While using an upper bound on planning time will eliminate the problem of plans failing, it might lead to the planner not finding a feasible plan to begin with. On the other hand, using too low an estimate could still lead to plans failing, as discussed above.

In this paper, we describe a new approach for situated temporal planning. Our planner is aware of the fact that plan execution will start once planning finishes, and incorporates this information into the internal data structure for temporal reasoning used by the planner, together with estimates of remaining planning time. This helps our planner prune partial plans which are likely to lead to the planner finishing planning too late for the plans to be of use, and focus on more promising branches of the search.

Our empirical evaluation demonstrates that this planner can handle temporal planning problems with absolute deadlines much better than a naive baseline approach, in realistic settings where planning time counts, and the plan can only start executing once it is completed. To the best of our knowledge, this is the first temporal planner to explicitly consider planning time, within the context of planning and execution. Thus, our planner is especially applicable to online planning for robotics, where a robot must find a plan to execute, but the world does not stop while the robot is planning.

## Preliminaries

We consider propositional temporal planning problems with Timed Initial Literals (TIL) (Cresswell and Coddington 2003; Edelkamp and Hoffmann 2004). Such a planning problem $\Pi$ is specified by a tuple $\Pi = \langle F, A, I, T, G \rangle$, where:

- $F$ is a set of Boolean propositions, which describe the state of the world.
- $A$ is a set of durative actions. Each action $a \in A$ is described by:
  - Minimum duration $dur_{\min}(a)$ and maximum duration $dur_{\max}(a)$, both in $\mathbb{R}^{0+}$ with $dur_{\min}(a) \leq dur_{\max}(a)$,
  - Start condition $cond_{\vdash}(a)$, invariant condition $cond_{\leftrightarrow}(a)$, and end condition $cond_{\dashv}(a)$, all of which are subsets of $F$, and
  - Start effect $\mathit{eff}_{\vdash}(a)$ and end effect $\mathit{eff}_{\dashv}(a)$, both of which specify which propositions in $F$ become true (add effects), and which become false (delete effects).
- $I \subseteq F$ is the initial state, specifying exactly which propositions are true at time 0.

- $T$ is a set of timed initial literals (TIL). Each TIL $l \in T$ consists of a time $time(l)$ and a literal $lit(l)$, which specifies which proposition in $F$ becomes true (or false) at time $time(l)$.
- $G \subseteq F$ specifies the goal, that is, which propositions we want to be true at the end of plan execution.

A solution to a temporal planning problem is a schedule $\sigma$, which is a sequence of triples $\langle a, t, d \rangle$, where $a \in A$ is an action, $t \in \mathbb{R}^{0+}$ is the time when action $a$ is started, and $d \in [dur_{\min}(a), dur_{\max}(a)]$ is the duration chosen for $a$. A schedule can be seen as a set of instantaneous *happenings* (Fox and Long 2003), which occur when an action starts, when an action ends, and when a timed initial literal is triggered. Specifically, for each triple $\langle a, t, d \rangle$ in the schedule, we have action $a$ starting at time $t$ (requiring $cond_{\vdash}(a)$ to hold a small amount of time $\epsilon$ before time $t$, and applying the effects $\mathit{eff}_{\vdash}(a)$ right at $t$), and ending at time $t + d$ (requiring $cond_{\dashv}(a)$ to hold $\epsilon$ before $t + d$, and applying the effects $\mathit{eff}_{\dashv}(a)$ at time $t + d$). For a TIL $l$ we have the effect specified by $lit(l)$ triggered at time $time(l)$. Finally, in order for a schedule to be valid, we also require the invariant condition $cond_{\leftrightarrow}(a)$ to hold over the open interval between $t$ and $t + d$, and that the goal $G$ holds at the state which holds after all happenings have occurred.

## Related Work

Temporal planners have of course been used in on-line applications before. For example, researchers at PARC built a special-purpose temporal planner for on-line manufacturing (Ruml et al. 2011). As in many temporal planners, each search node contains a Simple Temporal Network (STN) (Dechter, Meiri, and Pearl 1991) to represent the time points of events in the plan and constraints on when they can occur. To reflect the fact that actions cannot occur until planning has completed, the PARC planner includes a hard-coded estimate of the required planning time, and every time point in the STN is constrained to occur at least that far after the time that planning started (Ruml et al. 2011, Figure 11). While this is a reasonable solution in a domain where the expected planning problems are all of similar difficulty, this approach can perform poorly in domains that include a wide variety of problems, as we will see below.

There has also been work on time-aware planning in the search community. Dionne, Thayer and Ruml (2011) present a so-called 'contract algorithm' called Deadline-Aware Search (DAS) that, given a deadline, attempts to return the cheapest complete plan that it can find within that deadline. The main part of the algorithm works by estimating the time that will be required to find a solution beneath each node in the open list, and pruning those for which this estimate exceeds the remaining search time. The estimate is the product of three quantities that are determined on-line: the time required to expand a node, expressed in seconds, an estimate on the number of search nodes remaining on the path to a goal beneath the given node, notated $d(n)$, and the average number of expansions required before a generated node is selected for expansion, called the *expansion delay*. Although DAS was shown to surpass anytime algorithms on combinatorial benchmarks, its ideas have never been implemented in a domain-independent planner.

Bugsy (Burns, Ruml, and Do 2013) is a search algorithm that attempts to minimize the user's utility, which is represented as a linear combination of planning time and plan cost. If plan cost is makespan, then the utility measures the 'goal achievement time', or the time from when the goal is presented to the planner, and planning starts, to when the plan finishes executing, and the goal is achieved by the agent. Bugsy is a best-first search algorithm, and relies on an estimate of remaining planning time similar to that of DAS in order to estimate the utility of each node it expands. While Bugsy is sensitive to its own planning time, it is not cognizant of external timed events such as deadlines, and does not prune nodes based on temporal information.

Related concepts in the search community include real-time search and anytime search. In the real-time search setting, the planner must return within a prespecified time bound the next action for the agent to take. This differs from our setting, in which the planner must return a complete plan and the temporal constraints are fine-grained and can relate individual domain propositions to absolute times. In anytime search, a planner quickly finds a complete plan, and then uses additional computation time to improve it until either it is terminated by an external signal or an optimal solution is found. In our setting, the planner may not run indefinitely, but rather is expected to minimize the agent's goal achievement time. And while doing so, we demand that the planner recognize that time is passing and that it be responsive to timed events in the external world.

## Encoding Planning and Execution Time

Many temporal planners (e.g., (Coles et al. 2009; 2012; 2010; Benton, Coles, and Coles 2012; Fernández-González, Karpas, and Williams 2015; 2017)) rely on an internal Simple Temporal Network (STN) (Dechter, Meiri, and Pearl 1991) (or possibly a linear program or a convex optimization problem — but we will abuse terminology and call all of these the STN) to represent the temporal constraints between the set of the *time points* where actions start or end. Specifically, planners that support required concurrency (Cushing et al. 2007) tend to use this representation to support concurrent execution of actions.

When planning is done offline, the STN contains some time point $t_{ES}$, which is the start of plan execution, and is assigned the value of 0. For convenience, we split each occurrence of action $a$ in the plan into two snap-actions: $a_{\vdash}$ and $a_{\dashv}$, corresponding to the start and end of the action, respectively. For each of these we have a corresponding time point in the STN: $t(a_{\vdash})$ when $a$ starts, and $t(a_{\dashv})$ when $a$ ends. Actions which have started but not yet finished will only have the start time point, since this is a partial plan (as noted earlier, all starts eventually need to be paired with an end, but this is not a requirement of plans that are still under construction). Temporal constraints between the time points are either action *duration* constraints (between the time points of the same action occurrence), or *sequencing* constraints due to causal relations between actions. For example, if the end of action $a$ achieves the start conditions of action $b$, then we would have $t(a_{\dashv}) - t(b_{\vdash}) \geq \epsilon$, where $\epsilon$ is the minimum separation between two events that depend on each other (Fox and Long 2003). Or, if the start of $c$ threatens the preconditions of $d$, then $t(c_{\vdash}) - t(d_{\dashv}) \geq \epsilon$. Additionally, timed initial literals (TIL) (Edelkamp and Hoffmann 2004) are encoded into the STN by adding a time point $t(f)$ for the occurrence of TIL $f$, with the temporal constraint $t(f) - t_{ES} = time(f)$, where $time(f)$ is the time at which $f$ occurs, as specified in the problem definition. These are then ordered with respect to the other steps in the plan by, again, adding sequencing constraints due to the causal relations between $lit(f)$ and the other steps in the plan.

In this paper, we focus on *online* planning. We want to account for the fact that time passes *during* the planning process, and that, in fact, planning time and execution time are both the same. In order to do so, we modify the STN described above by adding two additional time points: $t_{PS}$ which is the time when planning started, and $t_{now}$ which is the current time. We add the temporal constraint that $t_{now} - t_{PS}$ equals the currently elapsed time in planning. The expression $t_{ES} - t_{now}$ corresponds to the remaining planning time, which is, of course, unknown. We will discuss this expression, and how to treat it, in the next section. Now, $t_{PS} = 0$, while $t_{ES}$ is unknown. Finally, because TILs describe absolute time, we must modify the temporal constraints corresponding to TILs to use $t_{PS}$ instead of $t_{ES}$, i.e., the temporal constraint for TIL $l$ would be $t(l) - t_{PS} = time(l)$, where $time(l)$ is the time at which $l$ must occur.

## Time-Aware Planning

We have described a technique for encoding an STN which captures the fact that execution only starts after planning ends, and planning takes time. We now describe the impact this has on search within a temporal planner.

### Forward Planning Search Space

We take as our basis the forward-search approach of the planner OPTIC (Benton, Coles, and Coles 2012). Here, each search state comprises the plan $\pi$ (of snap actions) that reaches that state; the propositions $p \subseteq F$ that hold after $\pi$ was executed from the initial state; and the Simple Temporal Network $STN(\pi)$ encoding the temporal constraints over $\pi$.

When expanding a state in OPTIC, successors were generated in one of three ways:

- By applying a *start* snap-action that is logically applicable: any $a_{\vdash}$ where $p \vDash cond_{\vdash}(a)$; $eff_{\vdash}(a)$ would not break the invariant condition of an action that has started in $\pi$ but not yet ended; and $cond_{\leftrightarrow}$ would be satisfied once $a_{\vdash}$ has been applied. In this case, in the successor state, $\pi' = \pi + [a_{\vdash}]$, $p$ is updated according to $eff_{\vdash}(a)$ to yield $p'$, and a variable $t(a_{\vdash})$ added to $STN(\pi')$. Sequence constraints are imposed on this such that it follows any step in $\pi$ that met one of $cond_{\vdash}(a)$; or whose effects refer to the same propositions as $eff_{\vdash}(a)$; or whose preconditions (including invariant conditions) would be threatened by $eff_{\vdash}(a)$[1].
- By applying an *end* snap-action that is logically applicable – any $a_{\dashv}$ where $a$ has started in $\pi$ but not yet ended; $p \vDash cond_{\dashv}(a)$; and whose effects $eff_{\vdash}(a)$ would not break the invariant of *any other* action that has started in $\pi$ but not yet ended. In this case, the successor state is updated in a way analogous to starting an action, with the additional STN constraint $dur_{\min}(a) \leq t(a_{\dashv}) - t_{a_{\vdash}} \leq dur_{\max}(a)$.
- By applying a *Timed Initial Literal* $l$ that has not already occurred in $\pi$. In this case, $\pi' = \pi + [l]$, $p$ is updated according to $lit(l)$ to yield $p'$, and a variable $t(l)$ is added to $STN(\pi')$. For the purposes of sequence constraints, this can be thought of as being a snap-action with no preconditions – it suffices to order it after steps in $\pi$ whose preconditions or effects refer to $lit(l)$. To fix the time at which $l$ occurs, an additional STN constraint is added: $t(l) - t_{PS} = time(l)$ – while snap-actions are ordered only relative to other points in the plan, TILs must also occur a specific amount of time after time zero.

State expansion in this way generates candidate successors that are logically feasible; to ensure they are also temporally feasible, only those whose STNs are consistent are kept. Using an all-pairs shortest path algorithm in the STN will both check consistency (with negative cycles corresponding to an inconsistent STN), and give us the earliest and latest possible time at which each snap-action could be applied. We denote these $t_{\min}(x)$ and $t_{\max}(x)$ for each STN variable $t(x)$. Typically, only the former of these is used – to map $\pi$ to a schedule $\sigma$, each start–end snap-action pair $a_{\vdash}$, $a_{\dashv}$ gives a triple $\langle a, t_{\min}(a_{\vdash}), (t_{\min}(a_{\dashv}) - t_{\min}(a_{\vdash}))\rangle$. In other words, apply each action as soon as possible, with the shortest possible duration, thereby minimizing execution time.

Extending this approach to planning while aware of planning and execution time requires a number of modifications, which we now step through.

**No action can start before plan execution starts** – because execution cannot start until a plan has been produced. That is, for each $a_{\vdash}$ in the plan $\pi$, we add a constraint $t_{ES} \leq t(a_{\vdash})$ to the STN, where $t_{ES}$ is the time at which execution will start. We do not know this *a priori*, but can at least say $t_{now} \leq t_{ES}$ is the time since the planner started executing. An STN for a plan produced during successor generation will then be consistent *iff* it is not already too late to start executing the plan.

These additional constraints can be thought of as pushing the earliest actions in the plan to start after now; the effects of which are then propagated through the STN to appropriately delay the later actions, according to the sequence and duration constraints. If an otherwise-consistent STN is made inconsistent by these, then necessarily there must be a snap-action $x$ where $t_{\max}(x) < t_{now}$ – i.e. we are past the latest point at which $x$ could have been applied.

[1]As search progresses in a strictly forward direction, all threats are dealt with by *demotion* – ordering the new step after existing steps.

Planning time particularly matters in the presence of TILs – in the absence of these, we can start executing a plan whenever we like by simply delaying the start of the first action. If TILs are present, though, these anchor the plan to having to fit around absolute time: with reference to state expansion, when a TIL is added to the plan, this fixes it to come after any earlier steps with which it would interfere, thereby constraining their maximum time.

**Automatically applying past TILs** – if we are now past the time at which a TIL has occurred, it is added to $\pi$ before expanding the state.

More formally, immediately before expanding a state $S = \langle \pi, p, STN(\pi) \rangle$, the following TILs are applied:

$$\{l \in T \mid t(now) \geq time(l) \wedge l \notin \pi\}$$

If there are several such TILs, they are applied in ascending order of $time(l)$. The mechanism for applying these TILs is identical to that in OPTIC: each is applied, to yield a successor state $S'$; and then $S'$ replaces $S$. By doing this before expanding the state, we account for time having passed since $S$ having been placed on the open list, and it being expanded – if in this time a TIL will have happened, $S$ is updated accordingly, before expansion.

If this modification was not made, search would be free to branch over what step should next be added to $\pi$. In the case where a TIL $l$ represents a deadline – by deleting a precondition on actions that must occur by a given time – search would be free to apply these actions, even though in reality it is too late. By forcing the application of past TILs, we avoid this behavior: all such actions would then become inapplicable.

**Pruning states where it is too late to start their plan** From the STN for a plan $\pi$, we can note the latest point at which that plan can start executing; and prune any states for which this time has already passed.

As noted earlier, to check if the STN for a state is consistent, we use an all-pairs shortest path algorithm. This incidentally yields the minimum and maximum time-stamps for each snap-action. For snap-actions that are ordered before a TIL – which are fixed in time – these maximum time-stamps are finite. Moreover, because the plan is expanded in a strictly forward direction, the maximum timestamps are monotonically decreasing: it is not possible to somehow order a new action before a plan step, in a way that reduces its maximum time-stamp. Thus, for each state $S = \langle \pi, p, STN(\pi) \rangle$ we identify the start snap-action in $\pi$ that has the earliest possible maximum time-stamp – this is the latest time at which $\pi$ could feasibly be executed:

$$latest\_start(\pi) = \min\{t_{\max}(a_{\vdash}) \mid a_{\vdash} \in \pi\}$$

Then, when $S$ is about to be expanded – after it was generated, placed on the open list, and then removed – it is pruned if $t_{now} > latest\_start(\pi)$.

## Experiments

To gain a concrete sense of the practical import of our technique, we experimentally compared it to the baseline method

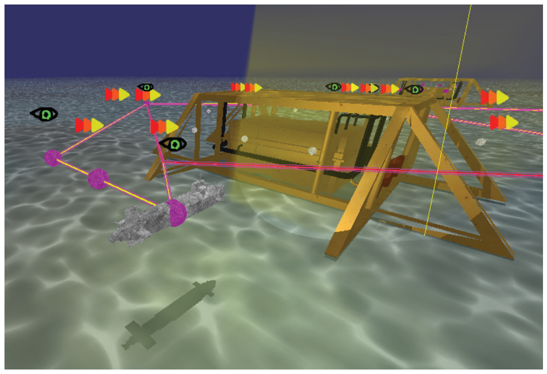


Figure 1: Screenshot of the underwater simulator, in which the AUV is inspecting the structure.

of prespecified planning times. We performed experiments in two types of domains: a realistic AUV simulation, and a set of IPC domains.

As a baseline against which to compare our time-aware planner, we used OPTIC in optimization mode, searching for the best plan within a varying fixed planning time of $T$ seconds. Time windows were considered to be $T$ seconds earlier, to adjust the initial state to the start of execution time. Therefore, a TIL $l$ occurring at time $time(l)$ seconds, using a planning time of $T$ seconds, will occur at time $(time(l) - T)$ (at least 0) in the plan.

## AUVs

We demonstrate the approach in simulation with autonomous underwater vehicles (AUVs). We embed OPTIC and our planner into ROS, using ROSPlan (Cashmore et al. 2015), to control the AUV. The AUV is equipped with a manipulator and placed in an underwater structure, with the task to inspect certain areas and to ensure that valves are turned to correct angles. The valves can only be turned within certain time windows, outside of which the valve is blocked. If the valve cannot be turned to the correct angle within an early time window, then a later window can be used. We generated 41 missions with varying time windows. A screenshot of the simulation is shown in Figure 1

These missions normally form part of a larger, strategic mission, spread out over a number of seabed manifolds. The AUV moves between these manifolds in order to complete the missions. Due to the uncertainty in the environment, it is not known beforehand precisely what time the AUV will arrive at the manifold. Before beginning the task, the AUV must construct a new plan. Plans with shorter durations are considered to be of higher quality, as this eases the time constraints on the remainder of the missions. We use this scenario to show that our approach allows the AUV to make use of earlier time windows, generating plans of higher quality.

The results are summarized in Table 1. The table shows the number of problems solved for each planner, out of a possible 41. Using our approach every problem was solved. Using a fixed planning time, some problems were unsolvable due to a planning time that was too short. The table

| | Time Aware | OPTIC$_{50}$ | OPTIC$_{100}$ | OPTIC$_{200}$ |
|---|---|---|---|---|
| best quality | 34 | 13 | 20 | 19 |
| IPC quality | 40.19 | 25.55 | 26.19 | 26.47 |
| problems solved | 41 | 26 | 34 | 40 |

Table 1: Table comparing the number of problems solved, the number of plans of highest quality, and the IPC quality for each approach.

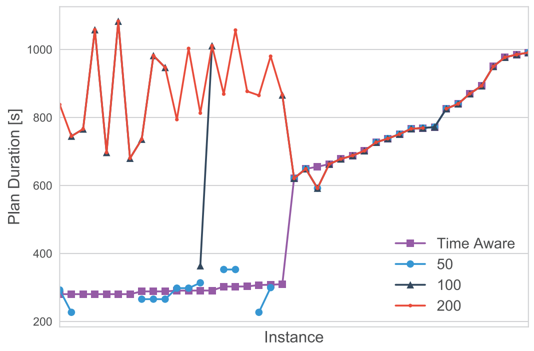


Figure 2: Plan durations per problem for each approach. The time-aware approach solves many problems using an earlier time window. OPTIC using a long planning time solves almost every problem, but only using the later time windows. Other planning time bounds are less reliable.

also shows the number of best plans for each approach. This is the number of problems for which that approach produced the plan of highest quality between the four approaches (possibly jointly). There it can be seen that although increasing the planning time allows for all problems to be solved, the quality is much poorer. The higher absolute number of best plans for the 200 second planning time is due to the greater number of problems solved. Finally, the table shows the IPC quality, calculated for all problems. These results demonstrate the choice between acting quickly, utilizing early time-windows, or producing plans reliably. Using the time-aware approach does both.

This can be seen more clearly in Figure 2. This figure compares the plan duration from each approach per problem. Using OPTIC$_{200}$ almost every problem is solved, at the longest possible plan duration – assuming planning takes 200 seconds forces the planner to have to use the later time windows. Other approaches may generate shorter plan durations, but fail to solve many of the problems.

## IPC Domains

In our IPC experiments, we tested all IPC-4 and IPC-5 domains that contain TILs: airport, pipesworld, satellite, truck, and UMTS. The UMTS domains and half of the airport instances were omitted as none of the planners completed

| | Time Aware | OPTIC$_{0.1}$ | OPTIC$_{1}$ | OPTIC$_{10}$ |
|---|---|---|---|---|
| best quality | 38 | 1 | 0 | 1 |
| IPC quality | 38 | 9.99 | 29.74 | 19.89 |
| problems solved | 38 | 10 | 30 | 21 |

Table 2: Table comparing the number of problems solved, the number of plans of highest quality, and the IPC quality for each approach

these. The planners were given a maximum of $200s$ of CPU time and 4GB of memory.

Table 2 presents results on the modified IPC domains. The fixed planning time planners were outperformed by the time-aware methods in every domain. Several instances were unsolvable by the former due to the fixed planning time constraints. Table 3 shows the planners detailed performance in each relevant domain tested.

In addition to the fixed planning times that are showed in Table 2 and Table 3 we have tested $50s$, $100s$, and $200s$. The performance of the baseline approach with these planning times were lower than the time-aware method and the best presented baseline, thus these results were omitted.

## Conclusions and Future Work

We have presented a domain-independent temporal planner that takes the interaction between the time spent on planning and execution time into consideration. We have demonstrated empirically that this planner achieves much better results in domains with absolute deadlines than our baseline approach. However, our work is merely the first step in addressing this important topic. There remain many exciting avenues for future work.

For example, our planner only looks at the current partial plan, and uses a heuristic to “look” into the future. This heuristic is used to estimate the remaining search depth, but not to obtain more information about future actions and their effects on deadlines. In order to get a more informed view of future actions, and their effect on deadlines, we will explore using temporal landmarks (Karpas et al. 2015). These landmarks could be encoded into the same STN of the partial plan, and thus we believe we will be able to achieve even better pruning of branches of the search tree which will not lead to a solution in time.

More broadly, the problem we are addressing here could benefit from more explicit metareasoning (Russell and Wefald 1991). For example, suppose we had a planning problem with two possible solutions, each of which must be explored on a separate branch of the search tree. Further suppose that each of these solutions has a deadline which leaves just enough time to explore one of the branches, but not both of them. Clearly, a planner with perfect knowledge would choose one of these branches and explore it. On the other hand, the approach we present here will explore both branches until it realizes there is not enough time left, and will then prune both branches — without solving the problem. In future work, we will explore ways of addressing this type of problem by incorporating explicit metareasoning on

| group | planner | solved | time | GAT |
|---|---|---|---|---|
| airport-1 | **Time Aware** | **14** | 6.62 | 193.54 |
| | OPTIC$_{0.1}$ | 2 | 0.06 | 89.61 |
| | OPTIC$_{1}$ | 10 | 0.24 | 167.72 |
| | OPTIC$_{10}$ | 10 | 0.20 | 176.72 |
| pipesworld | Time Aware | 3 | 0.72 | 16.06 |
| | OPTIC$_{0.1}$ | 1 | 0.05 | 12.11 |
| | **OPTIC$_{1}$** | **4** | 0.51 | 15.51 |
| | OPTIC$_{10}$ | 0 | | |
| satellite-1 | **Time Aware** | 1 | 0.03 | 190.23 |
| | OPTIC$_{0.1}$ | 1 | 0.04 | 190.31 |
| | OPTIC$_{10}$ | 1 | 0.02 | 200.21 |
| | OPTIC$_{1}$ | 1 | 0.02 | 191.21 |
| satellite-2 | **Time Aware** | **5** | 0.71 | 181.89 |
| | OPTIC$_{0.1}$ | 1 | 0.03 | 190.31 |
| | OPTIC$_{1}$ | 4 | 0.39 | 182.87 |
| | OPTIC$_{10}$ | 1 | 0.56 | 129.16 |
| satellite-3 | **Time Aware** | **5** | 0.80 | 181.88 |
| | OPTIC$_{0.1}$ | 1 | 0.03 | 190.31 |
| | OPTIC$_{1}$ | 4 | 0.36 | 182.87 |
| | OPTIC$_{10}$ | 1 | 0.56 | 129.16 |
| satellite-4 | **Time Aware** | **4** | 2.20 | 165.20 |
| | OPTIC$_{0.1}$ | 0 | | |
| | OPTIC$_{1}$ | 2 | 0.15 | 155.00 |
| | OPTIC$_{10}$ | 2 | 1.38 | 147.38 |
| truck | **Time Aware** | **6** | 0.21 | 1840.98 |
| | OPTIC$_{0.1}$ | 4 | 0.05 | 1673.95 |
| | OPTIC$_{1}$ | 5 | 0.06 | 1674.20 |
| | OPTIC$_{10}$ | **6** | 0.20 | 1855.97 |

Table 3: Table comparing the number of problems solved, the planning time, and the goal achievement time (GAT) grouped by IPC instance type. The planning time, and the GAT is the mean of all instances in the group solved by the planner.

planning time allocation into the search strategy.

One possible approach for this would be to treat the expression $t_{ES} - t_{now}$ as a variable, which we will denote by *slack*. We can then treat the STN as a mathematical optimization problem, and maximize the slack. The slack for node $n$ can serve as a proxy for the probability of finding a solution in time in the subtree rooted at $n$. Our metareasoning algorithm could then choose the next node to expand based on both heuristic estimates and the slack.

## Acknowledgements

This project has received funding from the European Union's Horizon 2020 Research and Innovation programme under Grant Agreement No. 730086 (ERGO).

## References

Benton, J.; Coles, A. J.; and Coles, A. 2012. Temporal planning with preferences and time-dependent continuous costs. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS)*.

Burns, E.; Ruml, W.; and Do, M. B. 2013. Heuristic search when time matters. *Journal of Artificial Intelligence Research* 47:697–740.

Cashmore, M.; Fox, M.; Long, D.; Magazzeni, D.; Ridder, B.; Carrera, A.; Palomeras, N.; Hurtós, N.; and Carreras, M. 2015. Rosplan: Planning in the robot operating system. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS)*, 333–341.

Coles, A.; Fox, M.; Halsey, K.; Long, D.; and Smith, A. 2009. Managing concurrency in temporal planning using planner-scheduler interaction. *Artificial Intelligence* 173(1):1–44.

Coles, A. J.; Coles, A.; Fox, M.; and Long, D. 2010. Forward-chaining partial-order planning. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS)*, 42–49.

Coles, A. J.; Coles, A.; Fox, M.; and Long, D. 2012. COLIN: planning with continuous linear numeric change. *Journal of Artificial Intelligence Research (JAIR)* 44:1–96.

Cresswell, S., and Coddington, A. 2003. Planning with timed literals and deadlines. In *Proceedings of 22nd Workshop of the UK Planning and Scheduling Special Interest Group*, 23–35.

Cushing, W.; Kambhampati, S.; Mausam; and Weld, D. S. 2007. When is temporal planning really temporal? In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*, 1852–1859.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49(1-3):61–95.

Dionne, A. J.; Thayer, J. T.; and Ruml, W. 2011. Deadline-aware search using on-line measures of behavior. In *Proceedings of the Symposium on Combinatorial Search (SoCS-11)*.

Edelkamp, S., and Hoffmann, J. 2004. PDDL2.2: The language for the classical part of the 4th international planning competition. Technical Report 195, University of Freiburg.

Fernández-González, E.; Karpas, E.; and Williams, B. C. 2015. Mixed discrete-continuous heuristic generative planning based on flow tubes. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*, 1565–1572.

Fernández-González, E.; Karpas, E.; and Williams, B. C. 2017. Mixed discrete-continuous planning with convex optimization. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, 4574–4580.

Fikes, R. E., and Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. In *Proceedings of the 2nd International Joint Conference on Artificial Intelligence (IJCAI)*, 608–620.

Fox, M., and Long, D. 2003. PDDL2.1: an extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research (JAIR)* 20:61–124.

Karpas, E.; Wang, D.; Williams, B. C.; and Haslum, P. 2015. Temporal landmarks: What must happen, and when. In *Pro-*

*ceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS)*, 138–146.

Niemueller, T.; Lakemeyer, G.; and Ferrein, A. 2015. The RoboCup Logistics League as a Benchmark for Planning in Robotics. In *WS on Planning and Robotics (PlanRob) at Int. Conf. on Aut. Planning and Scheduling (ICAPS)*.

Ruml, W.; Do, M. B.; Zhou, R.; and Fromherz, M. P. J. 2011. On-line planning and scheduling: An application to controlling modular printers. *Journal of Artificial Intelligence Research* 40:415–468.

Russell, S. J., and Wefald, E. 1991. Principles of metareasoning. *Artificial Intelligence* 49(1-3):361–395.

# Information-Efficient Model Identification for Tensegrity Robot Locomotion

**Shaojun Zhu, David Surovik, Kostas Bekris, Abdeslam Boularias**
Department of Computer Science, Rutgers University, New Jersey, USA
{shaojun.zhu, david.surovik, kostas.bekris, abdeslam.boularias}@cs.rutgers.edu

## Abstract

This paper aims to identify in a practical manner unknown physical parameters, such as mechanical models of actuated robot links, which are critical in dynamical robotic tasks. Key features include the use of an off-the-shelf physics engine and the data-efficient adaptation of a black-box Bayesian optimization framework. The task being considered is locomotion with a high-dimensional, compliant Tensegrity robot. A key insight in this case is the need to project the system identification challenge into an appropriate lower dimensional space. Comparisons with alternatives indicate that the proposed method can identify the parameters more accurately within the given time budget, which also results in more precise locomotion control.

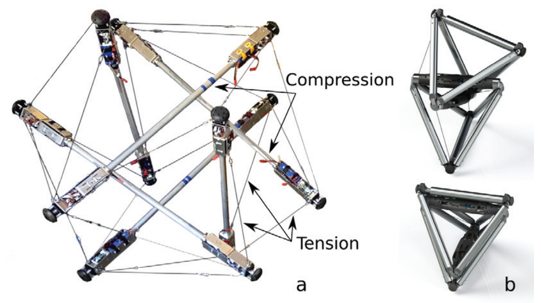


Figure 1: The Tensegrity robot (Caluwaerts et al., 2014).

## Introduction

This paper presents an approach for model identification by exploiting the availability of off-the-shelf physics engines used for simulating dynamics of robots and objects they interact with. There are many examples of popular physics engines that are becoming increasingly efficient (Erez, Tassa, and Todorov, 2015; Bul; MuJ; DAR; Phy; Hav). These physics engines receive as input mechanical and mesh models of the robots in a particular scene, in addition to controls (force, torque, velocity, etc.) applied to them, and return a prediction of the robot's dynamical response.

The accuracy of the prediction depends on several factors. The first one is the limitation of the mathematical model used by the engine (e.g., the Coulomb approximation). The second factor is the accuracy of the numerical algorithm used for solving the equations of motion. Finally, the prediction depends heavily on the accuracy of the physical parameters of the robots, such as mass, friction and elasticity. In this work, we focus on the last factor and propose a method to improve the accuracy of the physical parameters used in the physics engine.

In the context of compliant locomotion systems, the Tensegrity robot of Figure 1 is a structurally compliant platform that can distribute forces into linear elements as pure compression or tension (Caluwaerts et al., 2014). This robot's tensile elements can be actuated, enabling it to effectively adapt to complex contact dynamics in unstructured terrains. A policy for a rolling locomotive gait of the platform has been learned from simulated data (Geng et al., 2016). Tensegrity robots are inherently high-dimensional, highly-dynamic systems, and providing a predictive model requires a physics-based simulator (NTRT). The accuracy of such a solution critically depends upon physical parameters of the robot, such as the density of its rigid elements and the elasticity of the tensile elements. While a manual process can be followed to tune a simulation to match the behavior of a real prototype (Mirletz et al., 2015), it is highly desirable to conduct this calibration using as few observed trajectories as possible. In this work, trajectories generated by a simulation manually tuned to a prototypical robotic platform are used to identify the parameters of a physics engine for tensegrity modeling. Given the high-dimensionality of the parameter space, this is a challenging problem. This work proposes the mapping of the system identification process to a lower dimensional space of parameters. Methods used for dimensionality reduction include Random Embedding (REMBO) (Wang et al., 2016) as well as Variational Auto Encoder (VAE) (Kingma and Welling, 2014). A data-efficient Bayesian optimization technique is used for searching in the lower dimensional space, instead of the original high dimensional parameter space. The proposed method is able to efficiently identify the parameters that produce a simulation that most closely matches the observed ground-truth trajectories of this exciting locomotive platform.

## Foundations and Contributions

Two high-level approaches exist for learning robotic tasks with unknown dynamical models: model-free and model-based ones. Model-free methods search for a policy that best solves the task without explicitly learning the system dynamics (Sutton and Barto, 1998; Bertsekas and Tsitsiklis, 1996; Kober, Bagnell, and Peters, 2013; Levine and Abbeel, 2014). Model-free methods are accredited with the recent success stories of reinforcement learning in video games (Mnih et al., 2015). For robot learning, a relative entropy policy search has been used (Peters, Mülling, and Altün, 2010) to successfully train a robot to play table tennis. The PoWER algorithm (Kober and Peters, 2009) is another model-free policy search approach widely used in robotics.

Model-free methods, however, do not easily generalize to unseen regions of the state-action space. To learn an effective policy, features of state-actions in learning and testing should be sampled from distributions that share the same support. This is rather dangerous in robotics, as poor performance in testing could lead to irreversible damage.

Model-based approaches explicitly learn the dynamics of the system, and search for an optimal policy using standard simulation, planning, and actuation control loops for the learned parameters. There are many examples of model-based approaches for robotic manipulation (Dogar et al., 2012; Lynch and Mason, 1996; Merili, Veloso, and Akin, 2014; Scholz et al., 2014; Zhou et al., 2016), some of which have used physics-based simulation to predict the effects of pushing flat objects on a smooth surface (Dogar et al., 2012). A nonparametric approach was employed for learning the outcome of pushing large objects (furniture) (Merili, Veloso, and Akin, 2014). A Markov Decision Process (MDP) has been applied to modeling interactions between objects; however, only simulation results on pushing were reported (Scholz et al., 2014). For general-purpose model-based reinforcement learning, the PILCO algorithm has been proven efficient in utilizing a small amount of data to learn dynamical models and optimal policies (Deisenroth, Rasmussen, and Fox, 2011).

Bayesian Optimization is a popular framework for data-efficient black-box optimization (Shahriari et al., 2016). In robotics, some recent applications include learning controllers for bipedal locomotion (Antonova, Rai, and Atkeson, 2016), gait optimization (Calandra et al., 2016) and transfer policies from simulation to real world (Marco et al., 2017).

This work is based on a model-based approach, which instead of learning a dynamics model, it utilizes a physics engine, and concentrates on identifying only the mechanical properties of the objects instead of recreating the dynamics from scratch. Furthermore, it utilizes Bayesian optimization and identifies a process for dealing with high-dimensional system identification challenges efficiently.

## Proposed Approach

This work proposes an online approach for robots to learn the physical parameters of their dynamics through minimal physical interaction. Because of the high dimensionality of the parameter space of the tensegrity robot, even with efficient optimization method like Bayesian optimization (BO), it is still challenging to identify all the parameters efficiently. The overall framework of the model identification process is first introduced, then the approaches of dimensionality reduction to decrease the search space of BO in order to achieve efficient optimization are covered in detail.

### Model Identification

For the tensegrity robot, the physical properties of interest correspond to the density, length, radius, stiffness, damping factor, pre-tension, motor radius, motor friction, and motor inertia of the various rigid and tensile elements and actuators.

These physical properties are represented as a $D$-dimensional vector $\theta \in \Theta$, where $\Theta$ is the space of all possible values of the physical properties. $\Theta$ is discretized with a regular grid resolution. The proposed approach returns a distribution $P$ on discretized $\Theta$ instead of a single point $\theta \in \Theta$ since model identification is generally an ill-posed problem. In other terms, there are multiple models that can explain an observed trajectory with equal accuracy. The objective is to preserve all possible explanations for the purposes of robust planning.

The online model identification algorithm (given in Algorithm 1) takes as input a prior distribution $P_t$, for time-step $t \geq 0$, on the discretized space of physical properties $\Theta$. $P_t$ is calculated based on the initial distribution $P_0$ and a sequence of observations $(x_0, \mu_0, x_1, \mu_1, \dots, x_{t-1}, \mu_{t-1}, x_t)$. For the Tensegrity robot, $x_t$ is a state vector concatenating the 3D centers of all rigid elements, i.e., the rods in the corresponding Figure 1, and $\mu_t$ is a vector of motor torques.

The process consists of simulating the effects of the controls $\mu_i$ on the robot in states $x_i$ under various values of parameters $\theta$ and observing the resulting states $\hat{x}_{i+1}$, for $i = 0, \dots, t$. The goal is to identify the model parameters that make the outcomes $\hat{x}_{i+1}$ of the simulation as close as possible to the real observed outcome $x_{i+1}$. In other terms, the following black-box optimization problem is solved:

$$\theta^* = \arg\min_{\theta \in \Theta} E(\theta) \overset{def}{=} \sum_{i=0}^{t} \|x_{i+1} - f(x_i, \mu_i, \theta)\|_2, \tag{1}$$

wherein $x_i$ and $x_{i+1}$ are the observed states of the robot at times $i$ and $i+1$, $\mu_i$ is the control that applied at time $t$, and $f(x_i, \mu_i, \theta) = \hat{x}_{i+1}$, the predicted state at time $t+1$ after simulating control $\mu_i$ at state $x_i$ using physical parameters $\theta$.

The proposed approach consists of learning the error function $E$ from a sequence of simulations with different parameters $\theta_k \in \Theta$. To choose these parameters efficiently in a way that quickly leads to accurate parameter estimation, a belief about the actual error function is maintained. This belief is a probability measure over the space of all functions $E : \mathbb{R}^D \rightarrow \mathbb{R}$, and is represented by a Gaussian Process (GP) (Rasmussen and Williams, 2005) with mean vector $m$ and covariance matrix $K$. The mean $m$ and covariance $K$ of the GP are learned from data points $\{(\theta_0, E(\theta_0)), \dots, (\theta_k, E(\theta_k))\}$, where $\theta_k$ is a vector of physical properties of the object, and $E(\theta_k)$ is the accumulated distance between actual observed states and states that are obtained from simulation using $\theta_k$.

The probability distribution $P$ on the identity of the best physical model $\theta^*$, returned by the algorithm, is computed

```
Input: State-action-state data $\{(x_i, \mu_i, x_{i+1})\}$ for
        $i = 0, \dots, t$
        $\Theta$, a discretized space of possible values of
        physical properties;
Output: Probability distribution $P$ over $\Theta$ according to
        the provided data;
Sample $\theta_0 \sim \text{Uniform}(\Theta)$; $L \leftarrow \emptyset$; $k \leftarrow 0$;
repeat
    $l_k \leftarrow 0$;
    for $i = 0$ to $t$ do
        Simulate $\{(x_i, \mu_i)\}$ using a physics engine with
          physical parameters $\theta_k$ and get the predicted
          next state $\hat{x}_{i+1} = f(x_i, \mu_i, \theta_k)$ ;
        $l_k \leftarrow l_k + \|\hat{x}_{i+1} - x_{i+1}\|_2$;
    end
    $L \leftarrow L \cup \{(\theta_k, l_k)\}$;
    Calculate $GP(m, K)$ on error function $E$, where
      $E(\theta) = l$, using data $(\theta, l) \in L$;
    Sample $E_1, E_2, \dots, E_n \sim GP(m, K)$ in $\Theta$;
    foreach $\theta \in \Theta$ do
        $P(\theta) \approx \frac{1}{n}\sum_{j=0}^{n} \mathbf{1}_{\theta = \arg\min_{\theta' \in \Theta} E_j(\theta')}$
    end
    $\theta_{k+1} = \arg\min_{\theta \in \Theta} P(\theta) \log\big(P(\theta)\big)$ ;
    $k \leftarrow k + 1$;
until Timeout;
```
**Algorithm 1:** Model Identification with Greedy Entropy Search

from the learned GP as

$$P(\theta) \overset{def}{=} P\big(\theta = \arg\min_{\theta' \in \Theta} E(\theta')\big)$$
$$= \int_{E:\mathbb{R}^D \to \mathbb{R}} p_{m,K}(E) \Pi_{\theta' \in \Theta - \{\theta\}} H\big(E(\theta') - E(\theta)\big) \mathrm{d}E \tag{2}$$

where $H$ is the Heaviside step function, i.e., $H\big(E(\theta') - E(\theta)\big) = 1$ if $E(\theta') \geq E(\theta)$ and $H\big(E(\theta') - E(\theta)\big) = 0$ otherwise, and $p_{m,K}(E)$ is the probability of a function $E$ according to the learned GP mean $m$ and covariance $K$. Intuitively, $P(\theta)$ is the expected number of times that $\theta$ happens to be the minimizer of $E$ when $E$ is a function distributed according to GP density $p_{m,K}$.

Distribution $P$ from Equation 2 does not have a closed-form expression. Therefore, a *Monte Carlo* sampling is employed for estimating $P$. Specifically, the process samples vectors containing values that $E$ could take, according to the learned Gaussian process, in the discretized space $\Theta$. $P(\theta)$ is estimated by counting the fraction of sampled vectors of the values of $E$ where $\theta$ happens to have the lowest value, as indicated in Algorithm 1.

Finally, the computed distribution $P$ is used to select the next vector $\theta_{k+1}$ to use as a physical model in the simulator. This process is repeated until the entropy of $P$ drops below a certain threshold, or until the algorithm runs out of the allocated time budget. The entropy of $P$ is given as $\sum_{\theta \in \Theta} -P_{min}(\theta) \log\big(P_{min}(\theta)\big)$. When the entropy of $P$ is close to zero, the mass of distribution $P$ is concentrated around a single vector $\theta$, corresponding to the physical model that best explains the observations. Therefore, the next vector $\theta_{k+1}$ should be selected such that the entropy of $P$ would decrease after adding the data point $\big(\theta_{k+1}, E(\theta_{k+1})\big)$ to train the GP and re-estimate $P$ using the new mean $m$ and covariance $K$ in Equation 2.

The Entropy Search method (Hennig and Schuler, 2012) follows this reasoning and use Monte Carlo again to sample, for each potential choice of $\theta_{k+1}$, a number of values that $E(\theta_{k+1})$ could take according to the GP in order to estimate the expected change in the entropy of $P$ and choose the parameter vector $\theta_{k+1}$ that is expected to decrease the entropy of $P$ the most. The existence of a secondary nested process of Monte Carlo sampling makes this method impractical for online model identification. Instead, this work proposes a simple heuristic for choosing the next $\theta_{k+1}$. In this method, called *Greedy Entropy Search*, the next $\theta_{k+1}$ is chosen as the point that contributes the most to the entropy of $P$, i.e.,

$$\theta_{k+1} = \arg\max_{\theta \in \Theta} -P(\theta) \log\big(P(\theta)\big).$$

This selection criterion is greedy because it does not anticipate how the output of the simulation using $\theta_{k+1}$ would affect the entropy of $P$. Nevertheless, this criterion selects the point that is causing the entropy of $P$ to be high. That is, a point $\theta_{k+1}$ with a good chance $P(\theta_{k+1})$ of being the real model, but with a high uncertainty $P(\theta_{k+1}) \log\big(\frac{1}{P(\theta_{k+1})}\big)$.

## Random Embedding for Model Identification in the High Dimensional Space

For problems where the space $\Theta$ of physical properties has a high dimension $D$, the method presented in Algorithm 1 is not practical because the number of elements in discretized $\Theta$ is exponential in dimension $D$. This is a common problem in global search methods (Wang et al., 2016). In fact, it has been shown that Bayesian optimization techniques do not perform better than a random search when the dimension of the search space is too large (10 dimension in the experiment in (Ahmed, Shahriari, and Schmidt, 2016)). Therefore, Algorithm 1 cannot be directly used for robotic platforms with a large number of joints and parameters, such as the Tensegrity robot or compliant dexterous hands.

Dimensionality reduction is a popular solution to the problem of searching in high-dimensional spaces. This solution is particularly appealing in the context of this work because we are more interested in the accuracy of the predicted trajectory than in identifying the true underlying physical parameters. Mechanical models of motion tie together several parameters of an object. For example, in Coulomb's model, the mass and the friction of an object are used in a linear function to predict the motion of a sliding planar object. Therefore, one can map linearly these two parameters to a single parameter and still make accurate predictions of the motion.

Random embedding is an efficient and effective dimensionality reduction technique (Wang et al., 2016). Given a space of parameters $\Theta$ with dimension $D$, we generate a random matrix $A \in \mathscr{R}^{D \times d}$ that projects points from $\Theta \subset \mathbb{R}^D$ to a lower-dimensional space of parameters $\Omega \subset \mathbb{R}^d$ where $d < D$. Instead of discretizing $\Theta$, we discretize $\Omega$ into a regular grid and map each point $\omega \in \Omega$ to a point $\theta$ in the

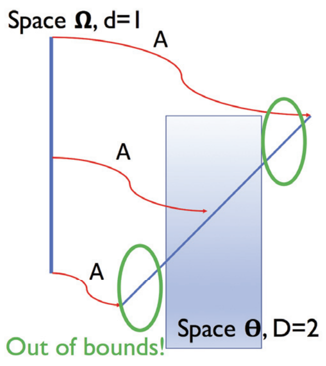


Figure 2: LEFT: A example of 1D-to-2D projection resulting in points outside the original domain. RIGHT: REMBO approaches this issue by projecting the point outside Θ to the nearest boundary point of Θ.

original high-dimensional space by using $A$, i.e. $\theta = A\omega$. One can show (Wang et al., 2016) that with probability one, $\min_{\theta\in\Theta} E(\theta) = \min_{\omega\in\Omega} E(A\omega)$ where $E$ is the error function in Equation 1. Consequently, we run Algorithm 1 using discretized $\Omega$ as input instead of $\Theta$. We project back the low-dimensional vectors $\omega \in \Omega$ to original parameter space $\Theta$ using $\theta = A\omega$ when we need to run the physical simulation to get the trajectory under a sampled value of $\omega$.

However, For a randomly generated matrix $A$ and point $\omega \in \Omega$, the corresponding high-dimensional vector $\theta = A\omega$ is not guaranteed to belong to $\Theta$, but could instead lie anywhere within $\mathbb{R}^D$. The simulator may consider $\theta$ as invalid if it is outside of $\Theta$ as shown in Fig.2. Moreover, just doing a rejection sampling does not always work because most of the points could be rejected for being invalid in some cases. *Random EMbedding Bayesian Optimization (REMBO) (Wang et al., 2016)* addressed this issue simply by projecting the point outside $\Theta$ to the nearest boundary point of $\Theta$.

## Variational Auto Encoder for Model Identification in the High Dimensional Space

An auto encoder is a neural network that learns to reconstruct the input by going through a latent space, which is in a lower dimensional space than the original input space(Vincent et al., 2010). It has shown to be very useful in unsupervised learning of low dimensional representations. A variational auto encoder (VAE) adds an additional constraint that the latent space follows a prior distribution, usually assumed to be Gaussian (Kingma and Welling, 2014). This additional constraint makes the model more useful as a generative model, as it also learns to generate output from the prior distribution in addition to reconstruction.

We adapt the VAE and combine it with the Bayesian optimization process, as shown in Fig. 3. Firstly, the VAE is trained with randomly sampled physical parameter data $\theta$ to learn a low dimension embedding $\alpha$. Once the VAE is optimized, the decoder part is used to project the low dimensional $\alpha$ back to the original physical parameter space $\theta$. Thus, the Bayesian optimization process as detailed in Algorithm 1 can

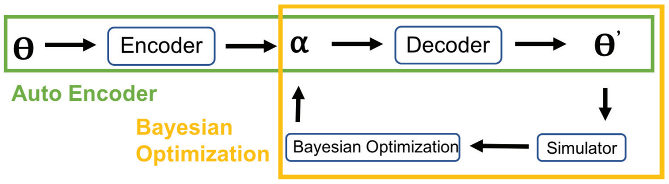


Figure 3: The auto encoder is trained first to learn the latent low dimensional embedding. Then Bayesian optimization is performed in this low dimensional space to search for the optimal parameter. The decoder is used to reconstruct the original 15 dimensional parameter in order to perform physical simulation.

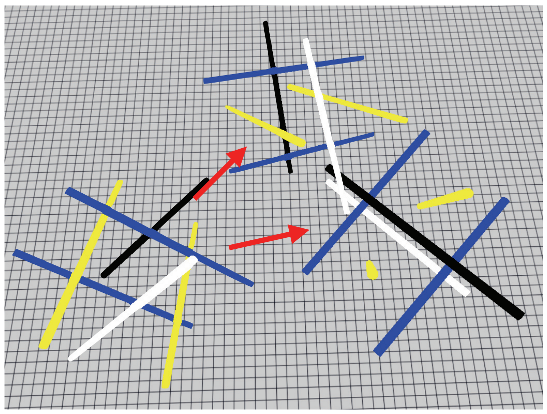


Figure 4: Simulation of the Tensegrity robot resulting in different states when executing the same control for different parameters.

be done efficiently in the low dimensional space. The decoder can be seen as a learned non-linear version of the projection matrix $A$ in REMBO.

## Experimental Results

**Setup:** This experiment aims to identify the 15 parameters of the T6 model of the Tensegrity SuperBall robot in NASA's Tensegrity Robotics Toolkit (NTRT). The complex dynamics and high dimensionality of the robot make this problem very hard. Fig. 4 shows an example of the different results of applying the same control to the robot with 1% difference in the rod length (one of the 15 parameters). In absence of access to the real robot, the default values of the T6 model in NTRT are used as ground-truth. The Guided Policy Search (GPS) algorithm (Levine and Abbeel, 2014) was used to discover fast trajectories of several flops through iterative exploration and refinement (GPS controller).

The Greedy Entropy Search (GES) method is compared against random search, where random values of the parameters are selected within the $\pm10\%$ range. Nevertheless, it is well-known that Bayesian optimization in high dimensions is difficult due to the exponential growth of the search space. To deal with this issue, the two dimensionality reduction methods, REMBO and VAE are used to reduce the dimensionality of the parameter space from 15 to 5.

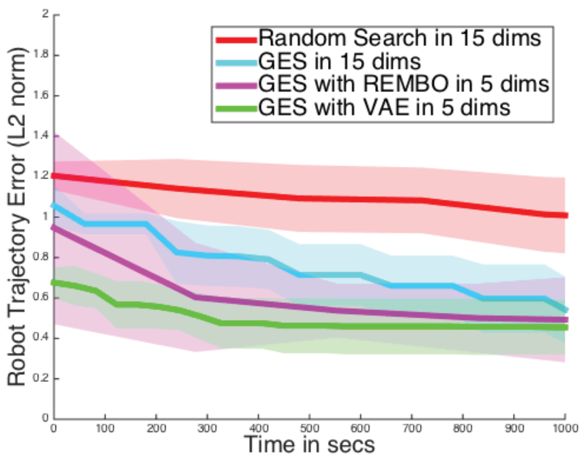


Figure 5: Test trajectory errors of different methods for the Tensegrity robot as a function of time budget for the parameter optimization process. Greedy Entropy Search in the 5-dimensional space using VAE achieves the lowest trajectory error, outperforming random search and Greedy Entropy Search in the original 15 dimensional space, as well as Greedy Entropy Search in the 5-dimensional space using REMBO.

The encoder and decoder of the VAE used in the experiment are both two-layer neural networks. The input dimension of the encoder and the output dimension of the decoder is 15, which is the dimension of the parameter space. The latent space is 5 dimensional. Between them is one layer of 400 dimensions. This dimension is chosen through cross validation by balancing accuracy and network complexity. The prior distribution of the latent space in the VAE is assumed to be $N(0,1)$. Based on the three-sigma rule, when sampling between $[-3,3]$, this interval should cover 99.7% of the latent space when the VAE is optimized. For REMBO, each time a random projection matrix is generated to project the parameters into $[0,1]$.

To train the VAE, 10,000 training trajectories are generated. These trajectories are generated by running the GPS controller in the simulator with different physical parameters and adding random noise of up to $\pm 10\%$ to the default parameter values. This means each trajectory is generated under slightly different physical parameters.

**Results:** Fig. 5 shows the average error between the trajectories using the model parameters identified by different methods and the trajectories generated from the ground-truth simulator. When optimizing in the original 15-dim. space, as a data-efficient global optimization method, Bayesian optimization with Greedy Entropy Search outperformed random search. Further improvements are achieved by dimensionality reduction, making the search more efficient. Greedy Entropy Search in the 5-dimensional space using VAE achieves the lowest trajectory error, outperforming the method using REMBO. This shows that a learned better latent embedding enables more efficient parameter search in the Bayesian optimization process. A video showing examples of the Tensegrity robot locomotion can be found on https://youtu.be/lD31s0c_tqM.

Fig. 6 provides the errors for each of the parameter as a function of time budget for the parameter optimization process. Only the combination of Greedy Entropy Search with VAE achieves close to 1% error for all parameters. Some parameters may have stronger influence on the robot dynamics. An intelligent way to identify these parameters would be helpful to reduce the dimensionality of the parameter space and could be more informative than random embeddings. This will be a direction for future work.

## Conclusion

This work proposes an information and data efficient framework for identifying physical parameters critical for robotic tasks, such as compliant robot locomotion. The framework aims to minimize the error between trajectories observed in experiments and those generated by a physics engine. To minimize the number of needed experiments, a Greedy variant of Entropy Search is proposed, which is shown to be data efficient. To solve high-dimensional challenges, this work integrates Greedy Entropy Search with a projection to a lower-dimensional space through random embedding or learning a latent embedding utilizing variational auto encoder. The evaluation of the proposed method against alternatives is favorable both in terms of identifying parameters more efficiently, as well as resulting in more accurate locomotion trajectories.

An interesting extension of this work would involve the identification of controls during the learning process that help in quickly minimizing the error. This can be a robust control process, which takes advantage of Bayesian Optimization's output in terms of a belief distribution for the identified parameters, so as to minimize entropy and maximize the safety of the experimentation process. Furthermore, it is interesting to compare the generality of the learned models and resulting control schemes that utilize them against completely model-free and end-to-end approaches for reinforcement learning and control.

## References

Ahmed, M.; Shahriari, B.; and Schmidt, M. 2016. Do we need "harmless" bayesian optimization and "first-order" bayesian optimization? In *NIPS BayesOPT Workshop*.

Antonova, R.; Rai, A.; and Atkeson, C. G. 2016. Sample efficient optimization for learning controllers for bipedal locomotion. In *Humanoid Robots (Humanoids), 2016 IEEE-RAS 16th International Conference on*, 22–28. IEEE.

Bertsekas, D. P., and Tsitsiklis, J. N. 1996. *Neuro-Dynamic Programming*. Athena Scientific, 1st edition.

Bullet physics engine. [Online]. Available: www.bulletphysics.org.

Calandra, R.; Seyfarth, A.; Peters, J.; and Deisenroth, M. P. 2016. Bayesian optimization for learning gaits under uncertainty. *Annals of Mathematics and Artificial Intelligence (AMAI)* 76(1):5–23.

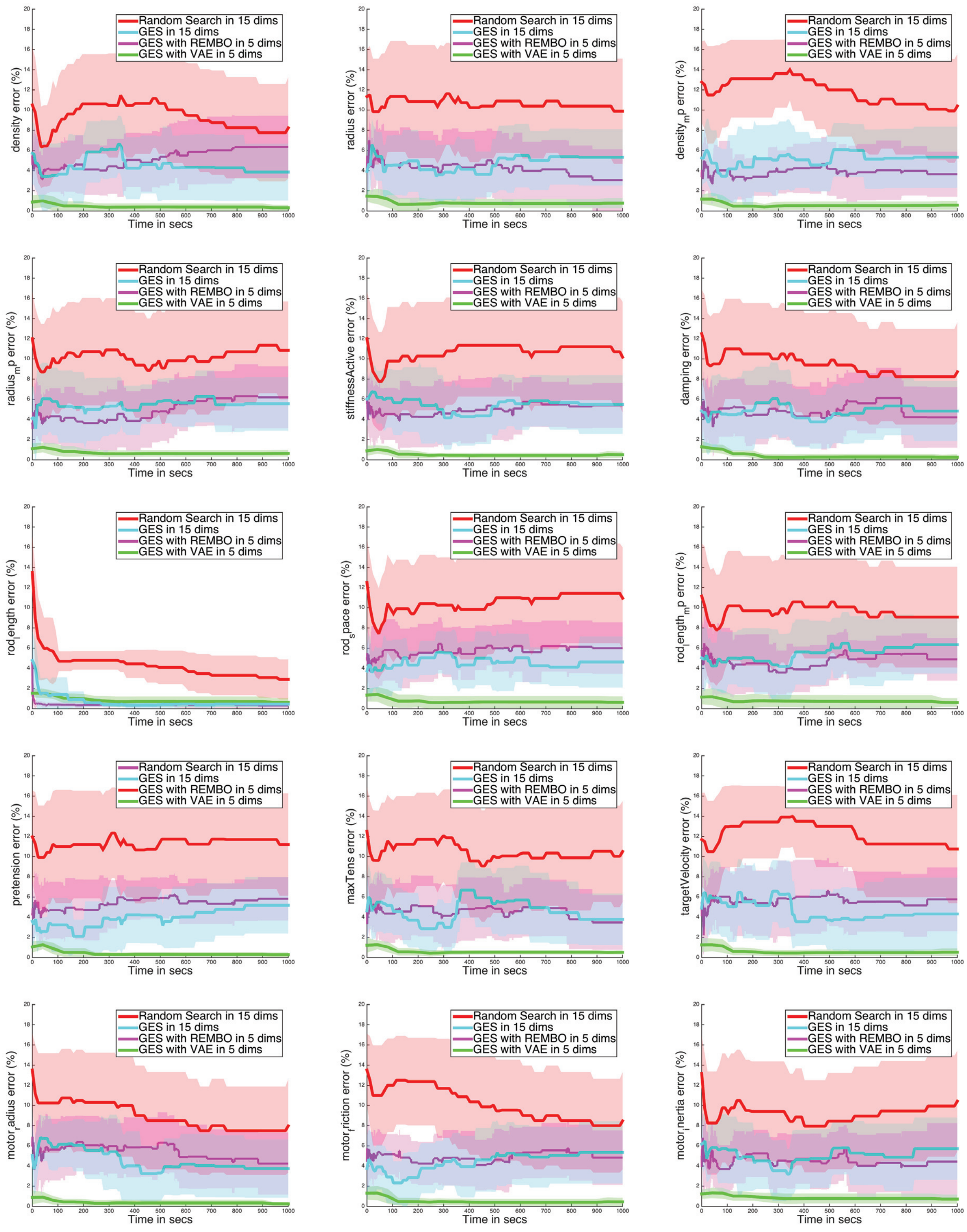


Figure 6: Each of the fifteen parameter error functions for the Tensegrity robot as a function of time budget for the parameter optimization process Greedy Entropy Search in the 5-dimensional space using VAE achieves the lowest error, which is less than 1% for all dimensions.

Caluwaerts, K.; Despraz, J.; Iscen, A.; Sabelhaus, A.; Bruce, J.; Schrauwen, B.; and SunSpiral, V. 2014. Design and control of compliant tensegrity robots through simulation and hardware validation. *Journal of The Royal Society Interface* 11(98).

DART physics egnine. [Online]. Available: http://dartsim.github.io.

Deisenroth, M.; Rasmussen, C.; and Fox, D. 2011. Learning to Control a Low-Cost Manipulator using Data-Efficient Reinforcement Learning. In *Robotics: Science and Systems (RSS)*.

Dogar, M.; Hsiao, K.; Ciocarlie, M.; and Srinivasa, S. 2012. Physics-Based Grasp Planning Through Clutter. In *Robotics: Science and Systems VIII*.

Erez, T.; Tassa, Y.; and Todorov, E. 2015. Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ODE and physx. In *IEEE International Conference on Robotics and Automation, ICRA*, 4397–4404.

Geng, X.; Zhang, M.; Bruce, J.; Caluwaerts, K.; Vespignani, M.; SunSpiral, V.; Abbeel, P.; and Levine, S. 2016. Deep reinforcement learning for tensegrity robot locomotion. *CoRR* abs/1609.09049.

Havok physics engine. [Online]. Available: www.havok.com.

Hennig, P., and Schuler, C. J. 2012. Entropy Search for Information-Efficient Global Optimization. *Journal of Machine Learning Research* 13:1809–1837.

Kingma, D. P., and Welling, M. 2014. Auto-encoding variational bayes. In *ICLR*.

Kober, J., and Peters, J. R. 2009. Policy search for motor primitives in robotics. In *Advances in neural information processing systems*, 849–856.

Kober, J.; Bagnell, J. A. D.; and Peters, J. 2013. Reinforcement learning in robotics: A survey. *International Journal of Robotics Research*.

Levine, S., and Abbeel, P. 2014. Learning neural network policies with guided policy search under unknown dynamics. In *Advances in Neural Information Processing Systems (NIPS)*.

Lynch, K. M., and Mason, M. T. 1996. Stable pushing: Mechanics, control- lability, and planning. *IJRR* 18.

Marco, A.; Berkenkamp, F.; Hennig, P.; Schoellig, A. P.; Krause, A.; Schaal, S.; and Trimpe, S. 2017. Virtual vs. real: Trading off simulations and physical experiments in reinforcement learning with bayesian optimization. In *2017 IEEE International Conference on Robotics and Automation, ICRA 2017, Singapore, Singapore, May 29 - June 3, 2017*, 1557–1563.

Merili, T.; Veloso, M.; and Akin, H. 2014. Push-manipulation of Complex Passive Mobile Objects Using Experimentally Acquired Motion Models. *Autonomous Robots* 1–13.

Mirletz, B. T.; Park, I.-W.; Quinn, R. D.; and SunSpiral, V. 2015. Towards bridging the reality gap between tensegrity simulation and robotic hardware. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; Petersen, S.; Beattie, C.; Sadik, A.; Antonoglou, I.; King, H.; Kumaran, D.; Wierstra, D.; Legg, S.; and Hassabis, D. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529–533.

MuJoCo physics engine. [Online]. Available: www.mujoco.org.

NTRT. NASA tensegrity robotics toolkit (NTRT). https://ti.arc.nasa.gov/tech/asr/intelligent-robotics/tensegrity/NTRT/.

Peters, J.; Mülling, K.; and Altün, Y. 2010. Relative entropy policy search. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI 2010)*, 1607–1612.

PhysX physics engine. [Online]. Available: www.geforce.com/hardware/technology/physx.

Rasmussen, C. E., and Williams, C. K. I. 2005. *Gaussian Processes for Machine Learning*. The MIT Press.

Scholz, J.; Levihn, M.; Isbell, C. L.; and Wingate, D. 2014. A Physics-Based Model Prior for Object-Oriented MDPs. In *Proceedings of the 31st International Conference on Machine Learning (ICML)*.

Shahriari, B.; Swersky, K.; Wang, Z.; Adams, R. P.; and de Freitas, N. 2016. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE* 104(1):148–175.

Sutton, R. S., and Barto, A. G. 1998. *Introduction to Reinforcement Learning*. Cambridge, MA, USA: MIT Press, 1st edition.

Vincent, P.; Larochelle, H.; Lajoie, I.; Bengio, Y.; and Manzagol, P.-A. 2010. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research* 11(Dec):3371–3408.

Wang, Z.; Hutter, F.; Zoghi, M.; Matheson, D.; and de Feitas, N. 2016. Bayesian optimization in a billion dimensions via random embeddings. *Journal of Artificial Intelligence Research* 55:361–387.

Zhou, J.; Paolini, R.; Bagnell, J. A.; and Mason, M. T. 2016. A convex polynomial force-motion model for planar sliding: Identification and application. In *2016 IEEE International Conference on Robotics and Automation, ICRA 2016, Stockholm, Sweden, May 16-21, 2016*, 372–377.

# On Chatbots Exhibiting Goal-Directed Autonomy in Dynamic Environments

**Biplav Srivastava**
IBM Research

## Abstract

Conversation interfaces (CIs), or chatbots, are a popular form of intelligent agents that engage humans in task-oriented or informal conversation. In this position paper and demonstration, we argue that chatbots working in dynamic environments, like with sensor data, can not only serve as a promising platform to research issues at the intersection of learning, reasoning, representation and execution for goal-directed autonomy; but also handle non-trivial business applications. We explore the underlying issues in the context of *Water Advisor*, a preliminary multi-modal conversation system that can access and explain water quality data.

## Introduction

Chatbots (McTear, Callejas, and Griol 2016), which can engage people in natural dialog conversation, have gained popularity recently drawn by numerous platforms to create them quickly for any domain (Accenture 2016). Most common types of such agents deal with a single user at a time and conduct informal conversation, answer the user's questions or provide recommendations in a given domain. They need to handle uncertainties related to human behavior and natural language, while conducting dialogs to achieve system goals. Chatbots have been deployed in customer care in many industries where they are expected to save over $8 billion per annum by 2022 (Juniper 2017).

However, the data sources used by common chatbots are static databases like product catalogs or user manuals. Therefore, for their problem of dialog management, i.e., creating dialog responses to user's utterances, effective approaches include learning policies over predictable nature of data(Young et al. 2013) or reasoning on its abstract representations (Inouye 2004).

The application scenarios become more compelling when the chatbot works in a dynamic environment, e.g., with sensor data, and interacts with groups of people, who come and go, rather than only an individual at a time. In such situations, the agent has to execute actions to monitor the environment, model different users engaged in conversation over time and track their intents, learn patterns and represent them, reason about best course of action given goals and system state, and execute conversation or other multi-modal actions.

We now explore the underlying issues of goal-directed autonomy in dynamic environment in the context of *Water Advisor* (WA) (Ellis et al. 2018), a prototypical multi-modal conversation system that can access and explain water quality data to a variety of stake-holders. We identify opportunities for learning, reasoning, representation and execution in WA and motivate more such applications.

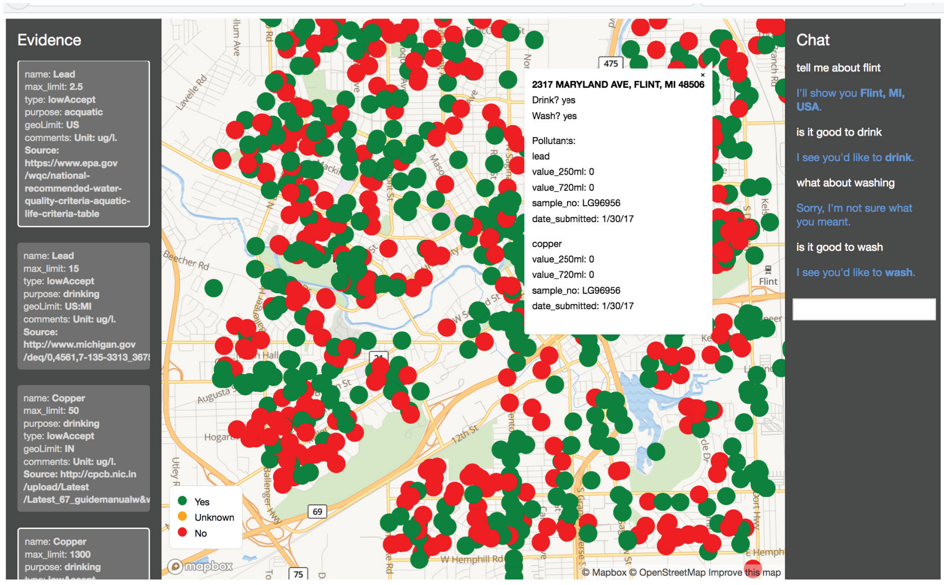
## Decision-Support for Water Usage With a Multi-Modal Conversation Interface

The global situation of water quality around the world is alarming in both developing and developed countries((UNEP) 2016) because water demand continues to rise while existing sources for fresh water are getting polluted. A key strategy for tackling water pollution is engaging people. A person makes many daily decisions touching on water usage activities like for profession (e.g., fishing, irrigation, shipping), recreation (e.g, boating), wild life conservation (e.g., dolphins) or just regular living (e.g., drinking, bathing, washing). Accessible tools for public are particularly useful to handle public health challenges such as the Flint water crisis (Pieper, Tang, and Edwards 2017).

A decision in this space needs to consider the activity (purpose) of the water use; relevant water quality parameters and their applicable regulatory standards for safety; available measurement technology, process, skills and costs; and actual data. There are further complication factors: there may be overlapping regulations due to geography and administrative scope; one may have to account for alternative ways to measure a particular water quality parameter that evolves over time; and water data can have issues like missing values or at different levels of granularity. The very few tools available today target water experts such as WaterLive mobile app for Australia [1], Bath app for UK[2], and GangaWatch for India (Sandha, Srivastava, and Randhawa 2017) and assume technical understanding of sciences.

[1]http://www.water.nsw.gov.au/realtime-data

[2]2https://environment.data.gov.uk/bwq/profiles/



Evidence

name: **Lead**
max_limit: **2.5**
type: **lowAccept**
purpose: **acquatic**
geoLimit: **US**
comments: **Unit: ug/l.**
**Source:**
**https://www.epa.gov/wqc/national-recommended-water-quality-criteria-aquatic-life-criteria-table**

name: **Lead**
max_limit: **15**
type: **lowAccept**
purpose: **drinking**
geoLimit: **US:MI**
comments: **Unit: ug/l.**
**Source:**
**http://www.michigan.gov/deq/0,4561,7-135-3313_3675**

name: **Copper**
max_limit: **50**
purpose: **drinking**
type: **lowAccept**
geoLimit: **IN**
comments: **Unit: ug/l.**
**Source: http://cpcb.nic.in/upload/Latest/Latest_67_guidemanualw&v**

name: **Copper**
max_limit: **1300**
purpose: **drinking**

**2317 MARYLAND AVE, FLINT, MI 48506**
Drink? yes
Wash? yes

Pollutants:
lead
value_250ml: 0
value_720ml: 0
sample_no: LG96956
date_submitted: 1/30/17

copper
value_250ml: 0
value_720ml: 0
sample_no: LG96956
date_submitted: 1/30/17

Chat

tell me about flint

I'll show you **Flint, MI, USA.**

is it good to drink

I see you'd like to **drink.**

what about washing

Sorry, I'm not sure what you meant.

is it good to wash

I see you'd like to **wash.**

Figure 1: A screenshot of *Water Advisor*. See video of it in action at https://youtu.be/z4x44sxC3zA.

## Water Advisor

Water Advisor (WA) is intended to be a data-driven assistant that can guide people without requiring any special water expertise. One can trigger it via a conversation to get an overview of water condition at a location, explore it by filtering and zooming on a map, and seek details on demand (Figure 1) by exploring relevant regulations, data or other locations. The current prototype uses water quality data available from Flint, MI[3] but future extensions will use open water data from US Geological Survey[4] (USGS) that is refreshed for thousands of places in US per day. However, the number of water quality parameters, for which data is available, varies widely between locations and over time, making generation of useful advice challenging. For regulations, WA relies on information provided by multiple agencies at national (US, India) and state levels (Michigan, New York), which has been consolidated for reuse[5].

## Technical Issues

In a water advising application, one or more users may need to interact with the chatbot if handling a complex decision like water contamination. The tool has to detect the user's information goals and meet them at lowest cognitive cost. The system uses a natural language classifier (NLC) to understand user utterance, and its error rate varies with input. The system has to decide whether to ask clarifying questions if it has low confidence and there are many ways to respond. The user may have preferences about how they specify an input (like location) and the kind of response they want (visual v/s textual). We discuss a range of issues below for exposition *but note that the current WA prototype handles only a subset of following integration issues.*

**Learning** plays an important role in understanding user's utterance, finding reliable water data samples in the database based on region and duration of interest, discovering issues in water quality and improving overall performance over time. In the prototype, for utterances, we use trained user models from commercial systems and for water quality, a simple regression method.

**Representation** is needed to map water's usage purpose to quality parameters and model safe limits of pollution parameters with different mathematic properties (e.g., polarity). It also helps map water purpose to regulations and further, aggregate and reconcile the latter when a region falls under overlapping jurisdiction of regulations. We represent this as geographically-scoped attribute-value pair in JSON format and make it publicly available for others to use and extend[6].

**Reasoning** is crucial to keep conversation focused based on system usability goals and user needs. One can model cognitive costs to user based on alternative system response choices and seek to optimize short-term and long-term behavior. Reasoning can further help to short-list regulations based on water activity and region of interest, generate advice and track explanations. We currently use rules on geographical scope and missing values to determine system response.

**Execution** is autonomous as the agent can choose to act by (a) asking clarifying questions about water usage goals or locations, (b) asking user's preference about advice, (c) seeking most reliable water data for region and time interval of interest from available external data sources, and corre-

[3]http://www.michigan.gov/flintwater/0,6092,7-345-76292_76294_76297—,00.html

[4]https://waterdata.usgs.gov/nwis/current/?type=quality

[5]https://github.com/biplav-s/water-info

[6]https://github.com/biplav-s/water-info

sponding subset of compatible regulations (d) invoking reasoning to generate an advice for water usage using filtered water data and regulations, (e) visualizing and explaining its output using water regulations, and (f) using one or more suitable modalities available at any turn of user interaction, i.e., chat, maps and document views. The current prototype uses a simplistic strategy for execution based on error rates, system confidence and usability rules.

**Human Usability Factors** have to be modeled and supported during WA's operation. In the current prototype, the user-interface controller module automatically keeps the different modalities synchronized so that the user is looking at consistent information across them. The system has to be aware of missing data or assumptions it is making, and needs to take them into account while communicating output advice in generated natural language. One avenue for future exploration is to measure and track complexity of interaction (Liao, Srivastava, and Kapanipathi 2017) and use sensed signals to pro-actively improve user experience. Another is to combine close-ended and open-ended questioning strategies for efficient interaction (Zhang, Liao, and Srivastava 2018).

**Ethical Issues** can emerge whenever a piece of technology is used among people at large. In the context of conversations, a recent paper surveys ethical issues (Henderson et al. 2018) like biases, adversarial examples, privacy violations, safety challenges and reproducibility concerns. A water-use chatbot can conceivably create bias among users of different activity subgroups (e.g., preferring recreation over drinking), compromise on privacy of users who submit queries about an activity or a region, and create public safety concerns (e.g., when users find scarcity of good quality water). We have not considered them in the prototype, however.

## Discussion and Conclusion

In this paper, we used decision-support in water as a usecase to demonstrate that chatbots can serve as a promising platform to integrate AI sub-disciplines for goal-directed autonomy. Apart from learning, reasoning, representation and execution, chatbots also need to work with human usability factors and ethical issues. An interesting aspect of these applications is that the chatbot may be helping a group of people take collective decision making, like conducting an interview, and data changes over time. Beyond water and customer support, complex applications are emerging in sciences (astronomy(Kephart et al. 2018)), business (career counseling[7], hospitality[8]) and societal domains (health[9]).

## References

Accenture. 2016. Chatbots in customer service. In *At: https://accntu.re/2z9s5fH*.

Ellis, J.; Srivastava, B.; Bellamy, R.; and Aaron, A. 2018. Water advisor - a data-driven, multi-modal, contextual assistant to help with water usage decisions. In *Proc. 32nd AAAI Conference on Artificial Intelligence (AAAI-18), New Orleans, Lousiana, USA.*

Henderson, P.; Sinha, K.; Angelard-Gontier, N.; Ke, N. R.; Fried, G.; Lowe, R.; and Pineau, J. 2018. Ethical challenges in data-driven dialogue systems. In *Proc. of AAAI/ACM Conference on AI Ethics and Society (AIES-18), New Orleans, Lousiana, USA.*

Inouye, R. B. 2004. Minimizing the length of non-mixed initiative dialogs. In Leonoor van der Beek, Dmitriy Genzel, D. M., ed., *ACL 2004: Student Research Workshop*, 7–12. Barcelona, Spain: Association for Computational Linguistics.

Juniper. 2017. Chatbots: Retail, ecommerce, banking & healthcare 2017-2022. In *At: http://bit.ly/2sJHejY*.

Kephart, J.; Dibia, V.; Ellis, J.; Srivastava, B.; Talamadupula, K.; and Dholakia, M. 2018. Cognitive assistant for visualizing and analyzing exoplanets. In *Proc. 32nd AAAI Conference on Artificial Intelligence (AAAI-18), New Orleans, Lousiana, USA.*

Liao, Q.; Srivastava, B.; and Kapanipathi, P. 2017. A Measure for Dialog Complexity and its Application in Streamlining Service Operations. *ArXiv e-prints*.

McTear, M.; Callejas, Z.; and Griol, D. 2016. Conversational interfaces: Past and present. In *The Conversational Interface. Springer, DOI: https://doi.org/10.1007/978-3-319-32967-3_4*.

Pieper, K. J.; Tang, M.; and Edwards, M. A. 2017. Flint water crisis caused by interrupted corrosion control: Investigating "ground zero" home. *Environmental Science & Technology* 51(4):2007–2014.

Sandha, S. S.; Srivastava, B.; and Randhawa, S. 2017. The gangawatch mobile app to enable usage of water data in every day decisions integrating historical and real-time sensing data. *CoRR* abs/1701.08212.

(UNEP), U. N. E. P. 2016. A snapshot of the worlds water quality: Towards a global assessment. In *Nairobi, Kenya. Online at: https://uneplive.unep.org/media/docs/assessments/unep_wwqa_report_web.pdf*.

Young, S.; Gašić, M.; Thomson, B.; and Williams, J. D. 2013. Pomdp-based statistical spoken dialog systems: A review. *Proceedings of the IEEE* 101(5):1160–1179.

Zhang, Y.; Liao, V.; and Srivastava, B. 2018. Towards an optimal dialog strategy for information retrieval using both open-ended and close-ended questions. In *Proc. Intelligent User Interfaces (IUI 2018, Tokyo, Japan, March.*

[7]https://www.ibm.com/talent-management/career-coach

[8]https://www.bebot.io/hotels

[9]https://www.healthtap.com/

# Interaction and Learning in a Humanoid Robot Magic Performance

**Kyle Morris, John Anderson, Meng Cheng Lau, Jacky Baltes**
University of Manitoba, Winnipeg, MB R3T2N2, Canada

**Abstract**

Magicians have been a source of entertainment for many centuries, with the ability to play on human bias, and perception to create an entertaining experience. There has been rapid growth in robotics throughout industrial applications; where primary challenges include improving human-robot interaction, and robotic perception. Despite preliminary work in expressive AI, which aims to use AI for entertainment; there has not been direct application of fully embodied autonomous agents (vision, speech, learning, planning) to entertainment domains. This paper describes preliminary work towards the use of magic tricks as a method for developing fully-embodied autonomous agents. A card trick is developed requiring vision, communication, interaction, and learning capabilities all of which are coordinated using our script representation. Our work is evaluated quantitatively through experimentation, and qualitatively through acquiring 2nd place at the 2016 IROS Humanoid Application Challenge. A video of the live performance can be found at https://youtu.be/OMpcmcPWAVM.

## Introduction

Humans have long enjoyed the clever trickery that comes from a good magic show. Magic tricks embody the primary features desired for an intelligent agent. These include **reactivity**: the ability to quickly perceive and respond to changes in the environment; **proactivity**: being goal-driven and acting towards reaching some desired goal; and **social ability**: the ability to communicate with others to further reach their goal (Wooldridge 2009).

Non-deterministic and dynamic environments pose challenges in developing robust autonomous agents that possess these features. This difficulty lies in balancing the proactive and reactive behaviour (Wooldridge 2009). An agent that is purely reactive may fail to reach a desired goal, whereas a purely proactive (goal driven) agent may not spend enough time acting to reach a goal (Wooldridge 2009).

During a live performance, reactivity is desired to provide authentic response time for each event in the script. Proactiveness involves seeking an end-performance goal that logically entails from the events in the script. The script is central to both reactivity and proactiveness. Lastly, social ability is required to leverage off the audience and guide a performance to cater towards their demographic and play off of their bias. For example, non-explicit humorous remarks are prioritized for an audience containing youth. Our work presents an autonomous agent that performs a magic card trick. We created motion, speech, and vision components on top of our custom DARwIn OP2 framework. These components utilize PocketSphinx for speech recognition, and OpenCV2 for playing card classification. The use of a finite state machine gives structure to the performance and allows the agent to seek an end-performance goal that accounts for potential problems that may arise during the show. Lastly, an easily adjustable design of events allows for a unique performance and user experience.

## Related Work

Live performance takes many forms. Humanoid robotics competitions have explored the development of robust, versatile agents that perform multiple distinct sporting events autonomously (Baltes et al. 2016). Furthermore, teams of robots are used to research how cooperation techniques are used in reaching a desired goal (Ashar et al. 2015). Such competitions have grown in popularity and have evolved to use more entertaining events that remain as useful benchmarks (Gerndt et al. 2015), but do not yet cater easily to a non-research audience.

Expressive AI has explored artificial intelligence for pure entertainment purposes in domains that include games (Mateas 2003) and music (De Mántaras and Arcos 2002); but lacks a robotics implementation. In the domain of Robotics, work has been done on incorporating entertainment (Kuroki 2001) with further specialization into card magic (Koretake, Kaneko, and Higashimori 2015). This work however puts focus on card manipulation, and mechanical aspects rather than timing and interaction. There has been growing discussion of the need for timing and human-robot interaction for effective live performance (Nuñez et al. 2014; Tamura, Yano, and Osumi 2014); but this discussion has been purely theoretical. Our work outlines a new application of robot entertainment for live magic that incorporates computer vision, machine learning, speech recognition and motion in order to deliver an authentic and robust perfor-

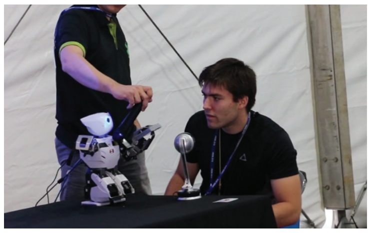


Figure 1: The live performance at IROS 2016. The robot is about to reveal the cards.

mance.

Employing template-matching for playing card recognition has demonstrated higher overall classification accuracy, but only in settings where the card is viewed from a fixed distance and angle (Brinks and White 2007). Similarly, this approach had significant latency (6 seconds) using a client-server architecture and has not yet been tested on a localized model (Brinks and White 2007). Work from (Zheng and Green 2007) demonstrated higher rank classification accuracy along with robustness to card rotation and scale, however there is no evaluation of the overall classification accuracy. Furthermore we achieved higher accuracy on Jack, Queen, and King cards, along with higher suit accuracy. Other approaches such as (Martins, Reis, and Teófilo 2011) achieved higher rank classification; but share similar challenges in suit classification. Despite marginally lower performance on rank classification, our system demonstrates significant overall classification accuracy while being robust to card rotation, translation, and scale.

## The Magic Trick

The trick is based on the classic straight-man act, in which a stern robot assistant contrasts with a charismatic but condescending human magician. A DARwIn-OP2 robot is asked to select and observe 3 cards from a deck. Vocal cues from the human magician provoke responses from the robot. Throughout the performance the robot grows impatient with the magicians' rude gestures and treatment, and takes over the magic performance by knocking the deck out of the magicians' hand. After the robot acquires the deck, the robot explains the simplicity of the magic trick, and reveals the 3 cards that were originally chosen, from the face-down deck.

### Problem Representation

We represent a performance as a collection of ordered phases. A phase is some discrete set of events that must take place together within a limited time. For example one phase may involve multiple listen-response events where an agent uses speech recognition and speech synthesis to follow dialog with a human magician. Another phase may rely on both motion gestures to hold a deck of cards, and computer vision to recognize playing cards.

Grouping events into phases allows for a graceful recovery from potential interrupts in the performance. If, for example, a dialog-only phase is taking place, and noise inter-

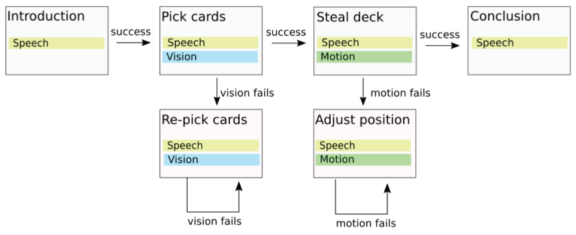
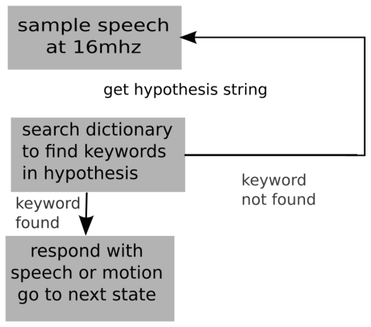


Figure 2: Phases of the performance


sample speech at 16mhz
get hypothesis string
search dictionary to find keywords in hypothesis
keyword not found
keyword found
respond with speech or motion go to next state
Figure 3: Control flow of speech processing

ference occurs, the agent may transfer to a backup phase which involves asking where the noise is coming from. During a card recognition phase that uses only the vision and motion components, it would not make sense for the agent to stop reading cards, or freeze up; because of the noise. It would make sense to have a backup phase in case the lighting is poor, in which the agent may ask for better lighting. The use of a state machine guides the performance by transitioning through pre-designed phases which together form a coherent story.

## Implementation

### Speech Recognition and Synthesis

Voice audio was recorded using a NESSIE Adaptive USB Condenser Microphone at 16kHz. Incoming audio is processed using PocketSphinx in order to generate a hypothesis string. This hypothesis string is checked against a custom language dictionary containing 89 keywords from the magic show script. If selected keywords are found in said string, this will trigger a response from the robot. Each dialog event may be customized to require multiple distinct keywords.

### Vision

Input images are captured using the built-in DARwIn-OP2 Logitech camera and passed to a custom vision module. The vision module was built with C++ and OpenCV2. The input image is first preprocessed by gray scaling, applying blur, and then applying a binary threshold. Contours are then extracted from the image and organized into a hierarchical tree

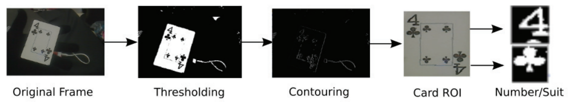


Figure 4: The vision pipeline

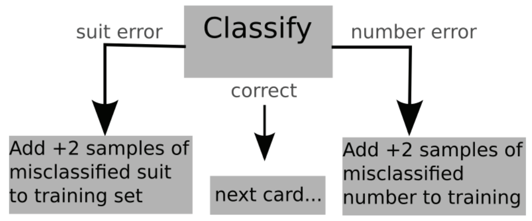

Classify
suit error
number error
correct
Add +2 samples of misclassified suit to training set
next card...
Add +2 samples of misclassified number to training

Figure 5: The training process

and compressed with OpenCV's simple chain approximation to gather only end-points of the contours. Polygon approximation is used on the contour to gather estimated corner points for a playing card. In order to eliminate false detection, the points are checked to be rectangular(based on the ratio between them). An affine transformation is used on the card ROI. Due to symmetry of playing cards, the bottom left corner is checked for a card symbol. If this symbol is missing, the card is assumed to be mirrored, and will be reflected to the correct orientation.

**Card Classification** Card suit (Diamonds, Hearts, Spades, Clubs) and rank (1-10, Jack, King, Queen, Ace) ROI are extracted. These ROI are then either dilated or eroded according to lighting in the environment. The suit and rank ROI are then classified using the K-Nearest Neighbours algorithm (Cover and Hart 1967).

## Machine Learning

The training process took place using a deck of 52 cards. The initial training set contained $5_{images} \times 4_{suits} \times 13_{cards} = 260$ samples collected using the robots built-in camera. Each sample is stored as a 30x30 gray-scale image in csv format as a $1 \times 900$ matrix of pixel brightness values [0-255]. The K-Nearest Neighbours algorithm (Cover and Hart 1967) is used to classify each suit and rank. An iterative training process is used. Initially each card within the full deck is shown in front of the robot. If the card is correctly classified, it is placed in a success pile. Misclassification may take place on either the card rank or suit. In either case, the misclassification is recorded and 2 positive samples of this rank or suit are added to the training set. The card will then be placed in a fail pile. For example if a Two of Hearts is misclassified as a Two of Diamonds, we will add 2 positive samples of the Hearts suit to the training set. The next iteration will begin using cards from the fail pile. This iterative process terminates when the fail pile is empty.

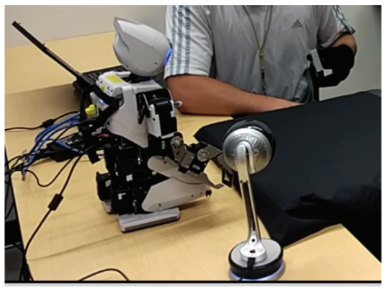


Figure 6: The dynamic evaluation setup.

## Evaluation

Our iterative training process was used, yielding the final training set. The test set was then created by randomly shuffling the deck and placing each card in front of the robot. This process was repeated 5 times to create a total of $5_{samples} \times 13_{ranks} = 65$ test samples for each suit, and $5_{samples} \times 4_{suits} = 20$ test samples for each rank. Evaluation was first completed in a dynamic setting. This included exposure to daylight, and randomization from a human holding the card in front of the robot. A second controlled evaluation consisted of static lighting, and a fixed placement of each card on a black surface.

A rank classification accuracy of 89.23% across the 13 card ranks was achieved using the dynamic setting. This surpassed the controlled setting which achieved 83.46% accuracy. Similarly the dynamic setting achieved a higher classification accuracy (90.38%) than the controlled setting (83.46%) on card suits. It is interesting to note the difference in spread between the two evaluations. The controlled setting has a higher standard deviation (10.76% for card rank, 15.99% for card suit) than the dynamic setting (4.07% for card rank, 11.15% for card suit). We believe this is due to our system being trained in a more dynamic setting.

## Conclusions and Future Work

This work explored the use of live entertainment in agent-based research. Specifically live magic performance was chosen as an avenue for developing a fully-embodied autonomous agent. Our card trick incorporates on-board vision, communication, interaction, and learning capabilities that allow for robust performance. This work may be greatly enhanced with improvements to the vision and machine learning components. Overall classification accuracy is dependent on both rank and suit accuracy. Our method demonstrated robustness to card rotation, translation and scale; but fell short in overall accuracy. We share similar challenges to other aforementioned vision techniques (Brinks and White 2007; Zheng and Green 2007; Martins, Reis, and Teófilo 2011), and believe improvements to image resolution would combat these challenges. Similarly, we see the use of colour recognition as a simple and promising approach to improve suit classification accuracy (Martins, Reis, and Teófilo 2011). Such improvements are challenging to acquire under time and space constraints imposed by on-board hardware. Lastly, we are interested in generalizing our work

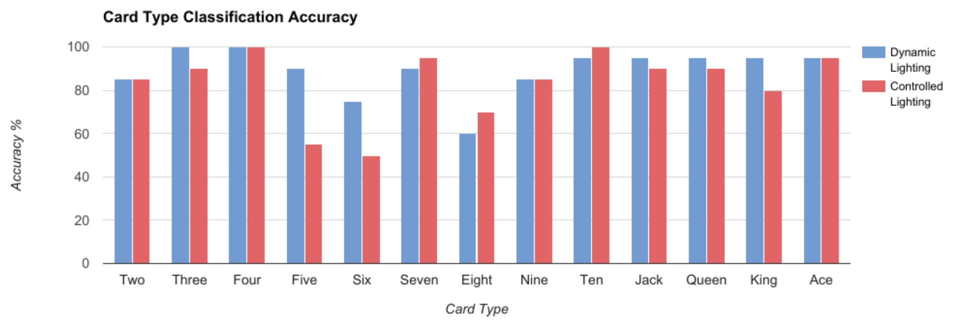


Figure 7: Classification results for card ranks. Taken from 20 samples of each card rank.

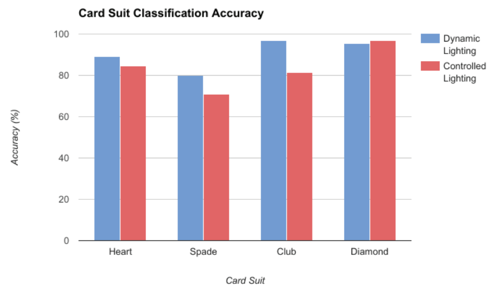


Figure 8: Classification results for card suits. Taken from 60 samples of each card suit.

into a framework for building agents capable of live performance. We believe this framework would provide easier entry, and thus encourage agent-based research using live entertainment.

## References

Ashar, J.; Ashmore, J.; Hall, B.; Harris, S.; Hengst, B.; Liu, R.; Mei (Jacky), Z.; Pagnucco, M.; Roy, R.; Sammut, C.; Sushkov, O.; Teh, B.; and Tsekouras, L. 2015. *RoboCup SPL 2014 Champion Team Paper*. Cham: Springer International Publishing. 70–81.

Baltes, J.; Tu, K.-Y.; Sadeghnejad, S.; and Anderson, J. 2016. Hurocup: competition for multi-event humanoid robot athletes. *The Knowledge Engineering Review* 1–14.

Brinks, D., and White, H. 2007. Texas hold'em hand recognition and analysis.

Cover, T., and Hart, P. 1967. Nearest neighbor pattern classification. *IEEE transactions on information theory* 13(1):21–27.

De Mántaras, R. L., and Arcos, J. L. 2002. Ai and music: From composition to expressive performance. *AI Magazine* 23:43–58.

Gerndt, R.; Seifert, D.; Baltes, J. H.; Sadeghnejad, S.; and Behnke, S. 2015. Humanoid robots in soccer: Robots versus humans in robocup 2050. *IEEE Robotics & Automation Magazine* 22(3):147–154.

Koretake, R.; Kaneko, M.; and Higashimori, M. 2015. The robot that can achieve card magic. *ROBOMECH Journal* 2(1):5.

Kuroki, Y. 2001. A small biped entertainment robot. In *MHS2001. Proceedings of 2001 International Symposium on Micromechatronics and Human Science (Cat. No.01TH8583)*, 3–4.

Martins, P.; Reis, L. P.; and Teófilo, L. 2011. Poker vision: playing cards and chips identification based on image processing. In *Iberian Conference on Pattern Recognition and Image Analysis*, 436–443. Springer.

Mateas, M. 2003. Expressive ai: Games and artificial intelligence. In *DiGRA '03 - Proceedings of the 2003 DiGRA International Conference: Level Up*.

Nuñez, D.; Tempest, M.; Viola, E.; and Breazeal, C. 2014. An initial discussion of timing considerations raised during development of a magician-robot interaction. In *Proc. ACM/IEEE Workshop on Timing in Human-Robot Interaction HRI*.

Tamura, Y.; Yano, S.; and Osumi, H. 2014. Modeling of human attention based on analysis of magic. In *Proceedings of the 2014 ACM/IEEE International Conference on Human-robot Interaction*, HRI '14, 302–303. New York, NY, USA: ACM.

Wooldridge, M. 2009. *An introduction to multiagent systems*. John Wiley & Sons.

Zheng, C., and Green, R. 2007. Playing card recognition using rotational invariant template matching. In *Proc. of Image and Vision Computing New Zealand*, 276–281.

# Exploiting Micro-Clusters to Close The Loop in Data-Mining Robots for Human Monitoring

**Einoshin Suzuki**
Dept. Informatics, ISEE, Kyushu University
744 Motooka, Nishi, Fukuoka, 819-0395 Japan

## Abstract

This paper describes our approach to integrating representation, reasoning, learning, and execution in our data-mining robots by exploiting micro-clusters to close the loop of the KDD process model. Based on our several kinds of autonomous mobile robots that monitor humans with Kinect and discover patterns, we are working on designing data-mining robots, each of which makes trials and errors in its data observation, data processing, pattern extraction, and mobile explorations. In other words, the robots continuously refine their goals at the micro-cluster level. We briefly discuss our four research directions, i.e., the balance between the exploitation and the exploration, the use of weak labels, the anytime algorithm, and the countermeasure to the concept drift, and describe potential, promising approaches for some of them.

## Data-Mining Robots for Human Monitoring

We have constructed several kinds of autonomous mobile robots that monitor humans with Kinect and discover patterns. For instance, one to three robots, either a TurtleBot 2 or a hand-crafted robot each with Kobuki, jointly monitor a walking human, typically with elderly-experience equipment, to discover fall risks by clustering his/her skeletons (Deguchi et al. 2017; Takayama et al. 2014). Another example is a TurtleBot 2 with Kobuki that clusters facial expressions to discover smiling, yawning, and reading clusters of a desk worker (Kondo, Deguchi, and Suzuki 2014). This robot was later used to detect his/her hidden fatigue by clustering classifiers of neutral faces and smiling faces, which were observed every 30 minutes with their weak class labels input through a wireless mouse (Deguchi and Suzuki 2015). Figure 1 shows snapshots of these robots in the respective series of experiments.

All these robots represent the monitored person with micro clusters, which are learnt based on procedures similar to BIRCH, a hierarchical clustering algorithm (Zhang, Ramakrishnan, and Livny 1997; Han, Kamber, and Pei 2012). A micro cluster, which represents a group of similar examples each described with a set of numerical features, in its original form is a triplet $(n, \mathbf{v}, \mathrm{s})$, where $n$, $\mathbf{v}$, and $s$ respectively represent the number of examples in the micro cluster, the add-sum of the examples in the micro cluster, and the add-sum of the squared L2-norm of the examples in the micro cluster (Zhang, Ramakrishnan, and Livny 1997). This triplet is called a Clustering Feature (CF) vector and has virtues of enabling an exact, incremental update and a reproduction of various cluster-wise distances without using the original examples. We initially adopted this approach to cluster colors of subimages observed by an autonomous mobile robot (Suzuki, Matsumoto, and Kouno 2012), and then extended the idea to cluster skeletons (Deguchi et al. 2017; Takayama et al. 2014), facial expressions (Kondo, Deguchi, and Suzuki 2014), and linear classifiers (Deguchi and Suzuki 2015). In these applications, an example is represented by a point in an Euclidean space spanned by the vectors of features, e.g., instability features described with skeleton joints inferred by Kinect (Deguchi et al. 2017; Takayama et al. 2014), action units inferred by Kinect to code emotional facial expressions (Kondo, Deguchi, and Suzuki 2014), coefficients of a logistic repression classifier to discriminate between neutral faces and smiling faces (Deguchi and Suzuki 2015).

Currently, we are working on extending our robots to data-mining robots, each of which makes trials and errors in its data observation, data processing, pattern extraction, and mobile explorations. The idea comes from the Knowledge Discovery in Databases (KDD) process model (Fayyad, Piatetsky-Shapiro, and Smyth 1996) shown in Figure 2. The Knowledge Discovery in Databases (KDD) process model states that a data mining process can be modeled as a series of several kinds of pre-/post-processing and pattern extraction. Our application domain is on a TurtleBot with Kobuki equipped with Kinect ver. 2 that continuously navigates inside a 90-m$^2$ room, observes desk workers, report discovered patterns to them, and receives their comments as rewards through its mouse. We believe that our data-mining robots are still goal-oriented, though their goals are unclear at the pattern level during their operations due to the nature of the KDD process model.

## Exploiting Micro-Clusters to Close The Loop

Our previous robots either neglect the discovered patterns and micro-clusters or use them through static proce-

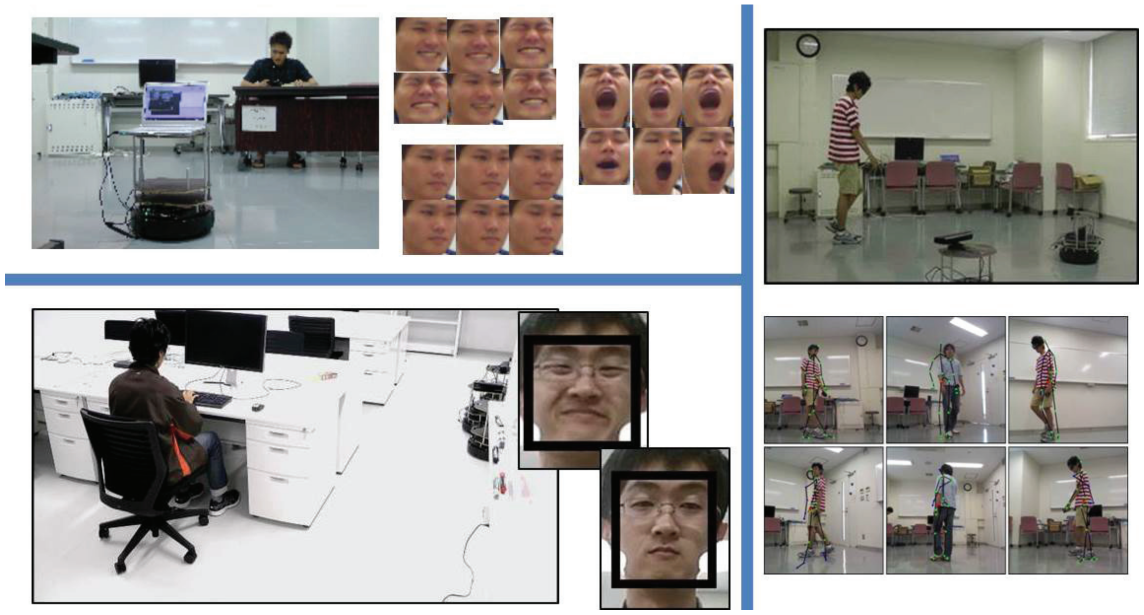


Figure 1: Snapshots of our autonomous mobile robots that monitor humans with Kinect and discover patterns. (Top left) TurtleBot 2 with Kobuki clusters facial expressions to discover smiling, yawning, and reading clusters of a desk worker (Kondo, Deguchi, and Suzuki 2014). (Right) Two TurtleBots 2 with Kobuki jointly monitor a walking human with elderly-experience equipment to discover fall risks by clustering his/her skeletons (Deguchi et al. 2017; Takayama et al. 2014). (Bottom left) TurtleBot 2 with Kobuki detects hidden fatigue of a desk worker by clustering classifiers of neutral faces and smiling faces, which were observed every 30 minutes with their weak class labels input through a wireless mouse (Deguchi and Suzuki 2015).

dures (Deguchi et al. 2017; Takayama et al. 2014; Kondo, Deguchi, and Suzuki 2014; Deguchi and Suzuki 2015). On the other hand, our intended data-mining robots closes "The Loop", i.e., realizes the trials and errors of the KDD process model especially by exploiting their results of the pattern discovery in their data observation and mobile explorations. In other words, the robots continuously refine their goals at the micro-cluster level. We have adopted four research directions: the balance between the exploitation and the exploration, the use of weak labels, the anytime algorithm, and the countermeasure to the concept drift.

Realizing the balance between the exploitation and the exploration requires care in our application due to the difficulty in estimating the interestingness of a discovered pattern in data mining. Though we have already built naive methods, e.g., moving to observe from a different angle when the set of micro clusters reaches a pre-defined degree of stability, the reward given by humans is not necessarily related to such diversity and how to estimate the correct, new angle for observation is unclear. Note that we are mostly faced with signal data, as the symbol grounding problem is far from being resolved. Modeling the diversity related to the interestingness would be the next step, though the exploration for new data would remain hard-wired.

We define a weak label as a piece of information related with supervisory signal, or the desired output value. It could be a class label of a bag of examples in the multiple instance learning, a class label in relevant learning tasks in multi-task or transfer learning, a (probabilistic) constraint on the target class labels in classification. See for instance (Mann and McCallum 2010). In our problem, the reward by a desk workers is rarely given, even if our robot reports an interesting pattern. We have recently developed a one-class selective transfer machine for personalized anomalous facial expression detection (Fujita, Matsukawa, and Suzuki 2018), which would be useful in both designing how to exploit weak labels and using the detected anomalous facial expressions as weak labels.

Naturally, our robot has to adopt an anytime algorithm, e.g., (Ueno et al. 2006), which can return the so-far best output anytime by using the available resources, especially the computation time. In BIRCH (Zhang, Ramakrishnan, and Livny 1997; Han, Kamber, and Pei 2012) and our discovery robots (Deguchi et al. 2017; Takayama et al. 2014; Kondo, Deguchi, and Suzuki 2014; Deguchi and Suzuki 2015), the micro-clusters are managed by a Clustering Feature (CF) tree, which may be viewed as a result of hierarchical clustering (Han, Kamber, and Pei 2012). Handling and reporting the micro-clusters in an intermediate level of the CF tree is a naive but natural solution. The closing the loop problem dictates that this research direction is deeply related with the first one: the balance between the exploitation and the exploration. Combined with the other two problems, designing an adequate anytime algorithm for our robots raises numerous challenges, even if partial solutions exist in the literature, e.g., (Ivanov, Blumberg, and Pentland: 2001).

Last but not least, our robot has to take a countermeasure to the concept drift, which is inherent in data stream

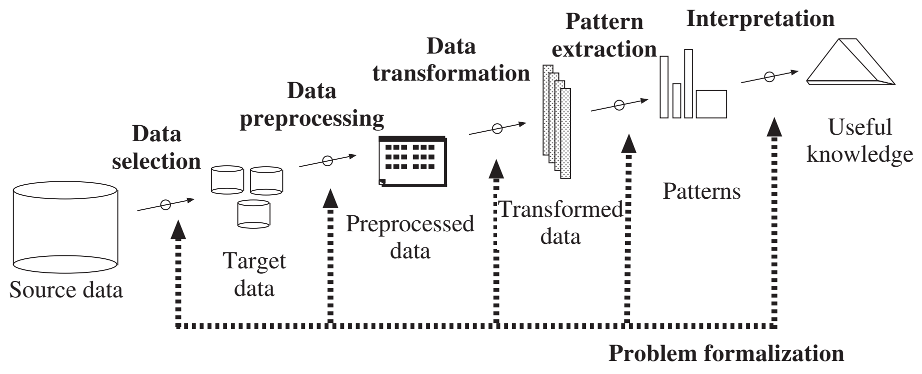


Figure 2: KDD process model (adopted and modified from (Fayyad, Piatetsky-Shapiro, and Smyth 1996)).

mining (Krempl et al. 2014). The statuses of desk workers change gradually or abruptly, though our robot platform including its batteries and sensors is reliable and can be regarded as static. Comparing CF trees (Boubou, Hafez, and Suzuki 2015) is in fact a nontrivial procedure and thus we are rather seeking for another approach of managing a set of micro-clusters.

## Acknowledgments

I deeply appreciate all co-authors and collaborators. A part of this research was supported by Grant-in-Aid for Scientific Research JP24650070, JP25280085, and JP15K12100 from the Japan Society for the Promotion of Science (JSPS); a Bilateral Joint Research Project between Japan and France funded by JSPS and CNRS (CNRS/JSPS PRC 0672); and Strategic International Cooperative Program funded by Japan Science and Technology Agency (JST) and ANR.

## References

Boubou, S.; Hafez, A. H. A.; and Suzuki, E. 2015. Visual Impression Localization of Autonomous Robots. In *Proc. 2015 IEEE International Conference on Automation Science and Engineering (CASE)*, 328–334.

Deguchi, Y., and Suzuki, E. 2015. Hidden Fatigue Detection for a Desk Worker Using Clustering of Successive Tasks. In *Ambient Intelligence*, volume 9425 of *LNCS*, 263–238. Springer-Verlag.

Deguchi, Y.; Takayama, D.; Takano, S.; Scuturici, V.-M.; Petit, J.-M.; and Suzuki, E. 2017. Skeleton Clustering by Multi-Robot Monitoring for Fall Risk Discovery. *Journal of Intelligent Information Systems* 48(1):75–115.

Fayyad, U. M.; Piatetsky-Shapiro, G.; and Smyth, P. 1996. From Data Mining to Knowledge Discovery: An Overview. In *Advances in Knowledge Discovery and Data Mining*. Menlo Park, Calif.: AAAI/MIT Press. 1–34.

Fujita, H.; Matsukawa, T.; and Suzuki, E. 2018. One-Class Selective Transfer Machine for Personalized Anomalous Facial Expression Detection. In *Proc. Thirteenth International Conference on Computer Vision Theory and Applications (VISAPP)*. (accepted for publication).

Han, J.; Kamber, M.; and Pei, J. 2012. *Data Mining, Concepts and Techniques*. Morgan Kaufmann.

Ivanov, Y. A.; Blumberg, B.; and Pentland:, A. 2001. Expectation Maximization for Weakly Labeled Data. In *Proc. ICML 2001*, 218–225.

Kondo, R.; Deguchi, Y.; and Suzuki, E. 2014. Developing a Face Monitoring Robot for a Deskworker. In *Ambient Intelligence*, volume 8850 of *LNCS*, 226–241. Springer-Verlag.

Krempl, G.; Žliobaite, I.; Brzeziński, D.; Hüllermeier, E.; Last, M.; Lemaire, V.; Noack, T.; Shaker, A.; Sievi, S.; Spiliopoulou, M.; and Stefanowski, J. 2014. Open Challenges for Data Stream Mining Research. *SIGKDD Explorations* 16(1):1–10.

Mann, G. S., and McCallum, A. 2010. Generalized Expectation Criteria for Semi-Supervised Learning with Weakly Labeled Data. *Journal of Machine Learning Research* 11:955–984.

Suzuki, E.; Matsumoto, E.; and Kouno, A. 2012. Data Squashing for HSV Subimages by an Autonomous Mobile Robot. In *Discovery Science (DS)*, volume 7569 of *LNAI*, 95–109. Springer-Verlag.

Takayama, D.; Deguchi, Y.; Takano, S.; Scuturici, V.-M.; Petit, J.-M.; and Suzuki, E. 2014. Multi-view Onboard Clustering of Skeleton Data for Fall Risk Discovery. In *Ambient Intelligence*, volume 8850 of *LNCS*, 258–273. Springer-Verlag.

Ueno, K.; Xi, X.; Keogh, E. J.; and Lee, D.-J. 2006. Anytime Classification Using the Nearest Neighbor Algorithm with Applications to Stream Mining. In *Proc. ICDM 2006*, 623–632.

Zhang, T.; Ramakrishnan, R.; and Livny, M. 1997. BIRCH: A New Data Clustering Algorithm and its Applications. *Data Mining and Knowledge Discovery* 1(2):141–182.

# SiRoK: Situated Robot Knowledge — Understanding the Balance Between Situated Knowledge and Variability

**Angel Daruna,[1] * Vivian Chu,[1] Weiyu Liu,[1] Meera Hahn,[1]**
**Priyanka Khante[2] Sonia Chernova,[1] Andrea Thomaz[2]**
[1]Institute for Robotics and Intelligent Machines, Georgia Institute of Technology, Atlanta, GA 30332, USA.
[2]Electrical and Computer Engineering, The University of Texas at Austin, Austin, TX 78712, USA.

## Abstract

General-purpose robots operating in a variety of environments, such as homes or hospitals, require a way to integrate abstract knowledge that is generalizable across domains with local, domain-specific observations. In this work, we examine different types and sources of data, with the goal of understanding how locally observed data and abstract knowledge might be fused. We introduce the Situated Robot Knowledge (SiRoK) framework that integrates probabilistic abstract knowledge and semantic memory of the local environment. In a series of robot and simulation experiments we examine the tradeoffs in the reliability and generalization of both data sources. Our robot experiments show that the variability of object properties and locations in our knowledge base is indicative of the time it takes to generalize a concept and its validity in the real world. The results of our simulations back that of our robot experiments, and give us insights into which source of knowledge to use for 31 types of object classes that exist in the real world.

## Introduction

Robotics is undergoing a transition from the development of specialized, single-task robots to general-purpose platforms expected to operate in diverse and changing environments, such as hospitals and homes. Operation in unconstrained human environments introduces many new challenges, one of which is that of knowledge acquisition. On the one hand, the diversity of target environments makes it impossible to precode the robot with all the required knowledge (e.g., where the towels are kept, that a particular bowl is made of metal), requiring the robot to learn from observations on-site. On the other, information often referred to as "common sense knowledge", can be transferred across domains (e.g., towels are often found in bathrooms and closets, bowls are containers) (Speer and Havasi 2012). In this work, we examine different types and sources of such data, to understand how

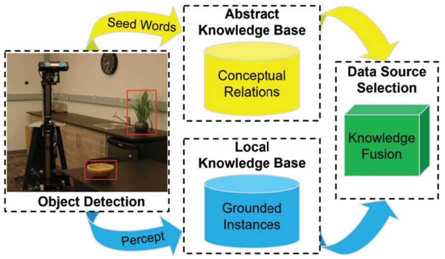


Figure 1: High-level view of SiRok framework.

locally observed data and abstract knowledge can be fused to enable a robot to most effectively reason about its world.

As a motivating example, consider a robot placed in a new home and tasked with fetching a glass of water. One approach is for the robot to rely entirely on local observations, and to exhaustively search the environment for a glass and sink. A human visitor to the home, however, would instead be likely to first find a kitchen, then begin to open cabinets (and not drawers) in order to find the glass. This behavior would be guided by semantic, domain-independent knowledge gathered from prior experiences, and a similar capability would enable robots to more effectively adapt to new environments. However, local knowledge must also be incorporated into this reasoning, allowing adaptation to domain-specific patterns or the current state of the world, such as when the glasses have already been set out on the table, or in houses with unconventional item storage areas. In order to support a robust deployment model, we must better understand the limits of both local and abstract data.

In this work, we consider two sources of knowledge: abstract knowledge and local knowledge. We characterize *abstract knowledge* as domain-independent information that generalizes across many environments (e.g., food in typical homes can be found in the refrigerator in the kitchen). Specifically, we use commonsense information from ConceptNet (Speer and Havasi 2012) and WordNet (Miller 1995) to allow the robot to reason about novel objects and environments. We characterize *local knowledge* as information the robot has perceived in its current environment. This includes information obtained from its sensors (e.g., camera, laser, etc.), including object recognition, semantic lo-

*This material is based upon work supported by the National Science Foundation Graduate Research Fellowship Program under Grant No. DGE-1650044. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

cations, and object properties. From these data sources we generate two separate knowledge bases, the Abstract Knowledge Base (AKB) and the Local Knowledge Base (LKB), which the robot uses to reason about the world. Combined, these components make up the Situated Robot Knowledge (SiRoK) framework (Fig. 1).

Our work makes the following contributions. First, we introduce a domain-independent framework for automatically retrieving common-sense knowledge for a given environment. We use object labels, obtained from object recognition, to generate seed words, which are then used to query existing semantic knowledge bases to construct a probabilistic model representing object type, location, and property data. Second, in a series of robot and simulation experiments we examine in what situations the abstract and local knowledge sources are most reliable for objects with both mutable and immutable properties. Our results show that variability is a key heuristic to take into account when evaluating knowledge sources. In particular, as variability increases, we should emphasize sources of general knowledge. For cases with extreme levels of variability, a robot should rely on direct observations or chance. Our simulations validate the trends we see in our robot experiments, and extend our conclusions to 31 different classes of objects found in real-world households.

## Related Work

Numerous projects across the AI community have sought to make use of commonsense and semantic knowledge. Three large-scale commonsense knowledge networks used across a wide range of applications are WordNet (Miller 1995), ConceptNet (Speer and Havasi 2012), and ResearchCyc (Lenat 1995; Matuszek et al. 2006). WordNet consists of a collection of synsets, which connect concepts hierarchically through the *IsA* relation. WordNet also distinguishes between different senses of the same word and provides glosses, or definitions, for each sense. While WordNet is clean and hand-coded, it also lacks diversity in the types of relations it contains. ConceptNet, on the other hand, contains several dozen different relations, but it does not distinguish between word senses and is largely crowdsourced, leading to a large amount of noise. ResearchCyc uses an even larger number of relations (currently around 17,000) to connect concepts. For the purposes of this work, we choose to use data from WordNet and ConceptNet to take advantage of the complimentary benefits of each.

In other work, Zhu, et al. (Zhu, Fathi, and Fei-Fei 2014a) perform affordance prediction on a set of images by using a Markov Logic Network (MLN) (Richardson and Domingos 2006a) to represent affordance knowledge. This work also does not deal with context and used hand-selected objects and affordances in the network. In (Chen and Liu 2011), contextual noise is addressed by disambiguating the concepts in ConceptNet to enrich the WordNet senses with more diverse knowledge for improved performance on word sense disambiguation tasks. While disambiguating ConceptNet helped provide context for each of its concepts, the resulting knowledge base contained only abstract information. In contrast to this approach, (Stoica and Hearst 2004) did

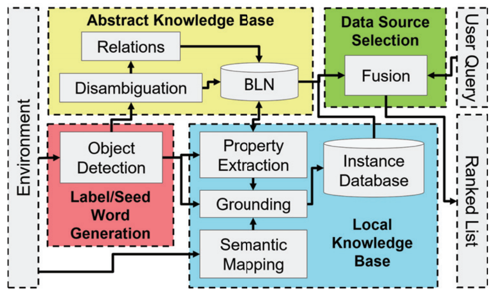


Figure 2: System architecture for the Situated Robot Knowledge (SiRoK) framework. The pipeline starts with environment data that is used to populate the AKB and LKB

construct a situated knowledge hierarchy in a (nearly) automated way, however, the resulting model only included hypernyms (the *IsA* relation).

Within robotics, the KnowRob (Tenorth and Beetz 2009) and RoboBrain (Saxena et al. 2014) projects are most closely related to our work. In KnowRob, the authors create a knowledge network from a variety of encyclopedic sources and represented the network using Prolog rules and the Web Ontology Language. This network is then used to repair robot task plans by filling in missing low-level details from high-level task descriptions. In RoboBrain, the authors generate a multimodal knowledge network for robotics using data collected automatically from the web. The resulting network is abstract and does not account for the domain-specific details relevant to the situational context of the robot. The RoboEarth project focused on the creation of a cloud repository of generalizable robot knowledge, including object models and robot task descriptions, that could be transferred across robot platforms and domains (Waibel et al. 2011). While these works deal with both abstract and situated knowledge, none of them investigate which knowledge source to leverage when. Our efforts focus on understanding which knowledge source a robot should use given some query (e.g. where is the plant) which may be part of a higher-level task. We conclude that the variability of a given piece of information impacts the reliability of obtaining it from either local or abstract sources.

## SiRoK System Architecture

The SiRoK framework is implemented as a system of interconnected modules, which communicate using ROS. The system has three main components (Fig. 2): AKB, LKB , and Data Source Selection, each of which contains a series of subsystems that aggregate and process data. At a high-level, the pipeline begins by performing object detection, where objects in the environment are assigned an object class labels (e.g., cups, bowls, etc.). These generated class names become seed words that are used to extract information from online commonsense networks to build an AKB. These object class labels are also used during grounding, where specific object information is stored into the LKB. In Data Source Selection, the robot uses specific queries to ask

| Data Types | Possible labels |
|---|---|
| Object Class | apple, banana, book, bottle, bowl, broccoli, cake, carrot, chair, clock, couch, cup, donut, fork, glass, knife, laptop, microwave, orange, oven, phone, pizza, plant, refrigerator, sandwich, sink, spoon, table, toaster, tv, vase |
| Colors | black, blue, brown, gray, green, orange, pink, purple, red, transparent, white, yellow |
| Materials | cloth, glass, metal, organic, paper, plastic, wood |
| Weights | light, medium, heavy |
| Shapes | arch, cylindrical, rectangle, spherical |

Figure 3: Classes and object data in the AKB and LKB

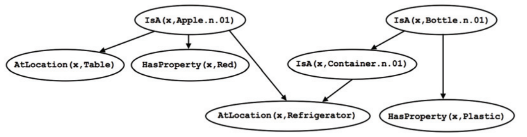


Figure 4: An example of abstract knowledge represented using a Bayesian Logic Network (BLN)

for information from AKB and LKB and fuses the results to respond to the queries. In the remainder of this section, we describe each subsystem in detail and the full system diagram can be found in Fig. 2. The colors of each component in Fig. 2 match the high-level view in Fig. 1.

## Object Detection

For object detection, we used the open source real-time object detection system YOLOv2 (Redmon et al. 2016). YOLOv2 uses a convolutional neural network and computes the location and classification of each object in an image in a single pass. It does this by dividing the image into cells, calculating an objectness score and then object classification probabilities over the individual cells, it then using anchor boxes to predict the object bounding boxes. We tested YOLOv2 on PASCAL VOC2012, achieving a mAP (mean average precision) score of 73.4. For our robot experiments, we trained YOLOv2 on the subset of COCO (Lin et al. 2014) object classes which are specific to the home environment (Fig. 3). Each time the system recognizes the object, the object label, bounding box of the object, and raw rectangle segment of the object is sent to the LKB. The object labels are also passed to the AKB.

## Abstract Knowledge Base

We represent the robot's AKB as a Bayesian Logic Network (BLN) (Jain, Waldherr, and Beetz 2009), a directed statistical relational model in which the variables under consideration are represented as first-order terms or predicates with arguments. BLNs allow logical constraints, represented as first-order logic rules, to be imposed on the network. Prior work in computer vision has utilized Markov Logic Networks (Richardson and Domingos 2006b), a representation that unifies Markov Random Fields and first-order logic, for modeling object attributes and affordances (Zhu, Fathi, and Fei-Fei 2014b). However, parameter learning in MLNs is an ill-posed problem (Jain, Kirchlechner, and Beetz 2007) and approximate inference is expensive even for simple queries. In contrast, BLNs are easy to train, more efficient and have scaled better to our application. Fig. 4 shows a small example BLN, which, once constructed, can be used to perform inference using likelihood weighting (Fung and Chang 2013) to answer queries such as *AtLocation*$(Object_i, x)$ or *HasProperty*$(Object_i, x)$.

To construct the BLN, we leverage information from two online sources of semantic knowledge, WordNet (Miller 1995) and ConceptNet (Speer and Havasi 2012). WordNet is a low-noise hand-crafted collection of sets of cognitive synonyms (synsets), each expressing a distinct concept (e.g., *spoon*) and related to other concepts through hypernym (the *IsA* relation, e.g., *IsA(spoon, utensil)*). ConceptNet is an auto-generated commonsense knowledge bank; it does not differentiate between word senses but groups all within a single concept node related to others through multiple possible relations. For example, for the object *mouse*, ConceptNet returns *AtLocation(mouse,office)* and *HasProperty(mouse, organic)*, highlighting the need to perform sense disambiguation to correctly parse this data.

Given seed words obtained from object recognition labels, we first perform sense disambiguation using the technique in (Tsatsaronis, Varlamis, and Vazirgiannis 2008), by finding the sense of each word that maximizes the overall similarity between the seed words (leveraging the fact that the words come from the same context). We then query WordNet and ConceptNet for semantic data related to each disambiguated word. Importantly, the seeds words not only provide a starting point for data retrieval, but together act as context for the robot's specific environment. Currently, we retrieve data for three relations, which we selected due to their usefulness in robot task execution.

- *IsA*: determines the relationship between an object and its hypernym (e.g., *IsA(bowl, container)*), allowing the robot to reason over object categories.
- *AtLocation*: determines the relationship between an object and locations in the world. (e.g., *AtLocation(bowl, sink)*, allowing the robot to query likely object locations.
- *HasProperty*: determines the relationship between an object and properties such as materials, shape, and colors (e.g., *HasProperty(bowl, ceramic), HasProperty(bowl, red)*, aiding in recognition and allowing the robot to reason about possible object uses (e.g. metal objects should not be placed in the microwave).

For each relation, we calculate a likelihood based on a weighted combination of the relation score from ConceptNet and the Explicit Semantic Analysis relatedness measure (Gabrilovich and Markovitch 2007) between the two concepts in the relation. This likelihood provides an initial estimate for the real-world probability of a given relationship and enables us to generate training evidence for BLN based on the distribution. Relations that cannot be sampled directly are inferred logically using transitive prolog rules. For additional details, see (Garrison and Chernova 2016).

## Local Knowledge Base

**LKB Data Structure** We represent the robot's local environment through a collection of object instances, forming a

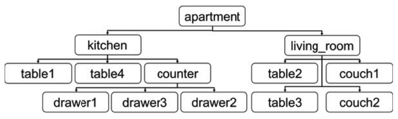


Figure 5: Topological map.

memory of encountered items, and their locations and properties. For $o \in O$, each object class out of the set of objects known to the robot (listed in Fig. 3), we store $i$ instances of that object within the LKB, where an instance is defined as a unique object.

The LKB is implemented using PyTables and HDF5; each object class $o$ is stored as a database, with a table generated for each object instance. For each instance, we currently store the object label, previously seen locations (pose and semantic label), image region corresponding to the bounding box from object recognition, visual information (RGB-D values), and all properties known about the instance (e.g, color, material). The resulting representation provides a scalable memory system that allows for efficient retrieval of all of its recent memories of instances.

**Grounding** In addition to using object recognition for object class labels (e.g., *bottle*), the robot must distinguish different instances of the same class (e.g., *red bottle* vs *yellow bottle*). The grounding component of SiRoK uses features distinct to instances of an object class to distinguish among multiple instances. This form of grounding, from here on referred to as *instance grounding*, was implemented using a K-Nearest Neighbors (KNN) classifier with a threshold distance to accommodate new instances of a class. Our implementation relies on color properties, extracted from the bounding box region of the image using the GrabCut algorithm (Rother, Kolmogorov, and Blake 2004) and uses KNN to determine whether an object is a new instance. Grounding enables the robot to perform color-based differentiation of objects, which we leverage in our study. In future work, we will expand instance grounding to incorporate spatial and temporal information about objects, as well as a wider variety of features.

**Semantic Location** In order to effectively generalize local information and relate it to abstract knowledge, we require a method for converting the robot's world coordinates to semantic location labels (e.g., *kitchen counter*). To provide a semantic location for an object, we utilize a hybrid map (Buschka and Saffiotti 2004), which links a topological map, consisting of a tree graph representing human domain knowledge, with a metric map of spatial locations in the environment. Fig. 5 and Fig. 6 show the topological and metric maps used in this work. The links between the topological map and metric map are expressed directly in the topological map nodes; association of each node with a volume in the metric map. This map structure enables the robot to obtain a semantic label for any 3D point that is hierarchical (e.g., object $o$ is in a *drawer* in the *kitchen* in the *apartment*).

**Property Extraction** As discussed above, SiRoK enables the robot to reason about a range of object properties, including color, weight, material and shape. Through local observation, the robot is able to obtain some properties (e.g., color), while other important object characteristics (e.g., material) are very difficult to determine for existing platforms. Some complementary information, however, can often be obtained from the AKB, which obtains property information through ConceptNet. For each object, we assign a set of object properties commonly learned and used by robots (Hermans, Rehg, and Bobick 2011; Sun, Bo, and Fox 2013; Sinapov et al. 2014). These include color, shape, material, and weight. The individual values that each object can take on (e.g. blue, heavy, metal, etc.) can be found in Fig. 3.

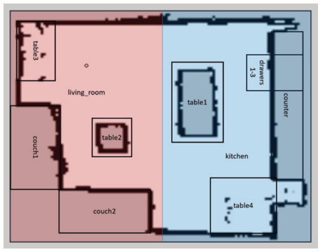


Figure 6: Metric map with an overlay of the spatial volumes associated with nodes in the topological map.

While color is obtained using a simple color classifier, we hand-label the shape and weight of the objects. With the current state of the art we assume that these properties can be obtained easily with good accuracy via existing machine learning algorithms and the use of pre-trained classifiers (Chu, Fitzgerald, and Thomaz 2016; Sun, Bo, and Fox 2013; Sinapov et al. 2014). Future work will include exploration of the objects using the robot's arm and visual information from the RGB-D camera to learn the object properties. However, material still remains to be one of the harder properties to be learned. In this work, we can leverage a human in the environment to extract the material properties of the objects.

In its existing form, the BLN contains far too many property edges to simply verify each one with the human. Thus we present an algorithm, which takes the existing BLN generated from ConceptNet and WordNet, and actively selects a subset of property relations to verify with the human. This results in a pruned representation that is consistent with the specific objects in the current environment.

We first modify the BLN to include inter-property edges. For all properties in the BLN, we add an edge if a relation exists between them in ConceptNet. We then generate three tables. $T_{material}$: all material properties present in our BLN (i.e., holds a relation with *Material* in the ConceptNet). For the next two tables, we use the association index in ConceptNet, a measure between 0 to 1 of how related two words are. $T^{O_N}_{assoc}$: holds all the association indices between an $O_N$ and every property belonging to that object (we ignore properties with index < 0.07).$T_{interprop}$: Let $P_O$ be a set such that each $p \in P_O$ is a property of $O$, this table holds the inter-property association indices between any two properties in $P_O$.

Next, we systematically pick the properties to query an expert for verifications. For each object, we query the expert about property, $p \in P_O$ with the highest association index in $T^{O_N}_{assoc}$. If it is verified *true* and exists in $T_{material}$, then all other material properties belonging to that object are assumed to be *false* and are not queried. We can also assume the predecessors of that property are true for $O_N$ (e.g., if Aluminum is true, then Metal can be assumed true). For the successors, we assume their *hasProperty* relations are true (e.g., Metal true, then Opaque true), but need to query the successors with an *IsA* (e.g., if Metal true, still need to ask about Aluminum). If a node in this *isA* set is verified to be *true*, the rest are assumed to be *false*.

Next, query with the a property with the minimum inter-property association index with $p$, to ask the most different question next. Repeat this process until all the properties are verified as *true/false*. We construct an expert-verified BLN, *vBLN*, with all verified *true* properties. For evaluation we will look to compare this verified BLN with a ground truth BLN with a dissimilarity index, $I_{dissimilarity}$, defined as:

$$\frac{\text{Uncommon edges between ground truth and vBLN}}{\text{Total number of unique edges in ground truth and vBLN}}$$

## Data Source Selection

SiRoK uses knowledge from the AKB and LKB to handle object *queries* related either to (1) what the object is, (2) where it is located, or (3) what properties it has. Within the AKB, the BLN is queried for *IsA*, *AtLocation*, and *HasProperty* information, and the results sorted by probability value. The LKB answers *AtLocation*, and *HasProperty* queries by using the stored outputs from semantic mapping and property classification, returning a ranked list of the most frequently encountered property. We note that, in general, location and property information have different characteristics. A specific object is likely to change location, possibly even frequently, whereas most of the properties we consider, such as color, are likely to change less often. Locations and properties also often generalize across instances (e.g., cups of the same color or cups stored in the same cabinet), but this depends on the variability of the object. In the next section, we evaluate how our inference performs across these different data types.

## Robot Experiments

To evaluate the SiRoK system and examine the relative applicability of abstract knowledge and local knowledge, we designed a series of experiments testing the robot's ability to predict object locations and properties. Our test environment resembles a simple apartment containing furniture and different use areas, as seen in Fig. 6. For all experiments, we use the robot platform, Prentice (Fig. 1). Prentice is an omnidirectional mobile robot and has a horizontally mounted lidar for navigation and a Microsoft Kinect2 RGB-D camera mounted on a pan/tilt unit for visual sensing.[1]

[1]Note that we do not evaluate *IsA* queries on the robot due to the highly abstract nature of the data. *IsA* results are reported in the simulation section.

### Building the Knowledge Bases

We populate an AKB by using the 31 possible class labels shown in Fig, 3 to seed a BLN using ConceptNet and WordNet. As described in *SiRoK System Architecture*, these class labels come from the COCO image dataset that are associated with kitchen and living rooms. We removed one label, hot dog, due to WordNet disambiguating hot dog to sandwich. This is due to WordNet characterizing that hot dogs are sandwiches, which is partially true (i.e., a hot dog is a piece of meat between bread). Future work will address how to take into account words that are part of the same hypernym hierarchy. The constructed BLN contains 257 nodes and 358 edges.

To gather data for the LKB, we used the following experimental steps: (1) put object(s) in our testing environment, (2) allow the robot to observe the environment and update the LKB, (3) update the state of the object(s) in our environment, then repeat this process for the desired number of observations. After each observation, we evaluate the accuracy for finding objects or naming object properties on a fixed test set. To populate the semantic locations, we provide an expert labeled semantic map that correlates to the described scene in Fig. 6. We use a color classifier to label each object in the test environment and the BLN for the object material. The average classifier accuracy is 70% and average clarifications needed for object property is 2.

If a human is available, SiRoK has the option to interactively validate properties in the BLN. We performed 84 clarifications to prune 50 edges in the vBLN from 195 property edges using the human-verification algorithm mentioned in Section III-C.4. While this is a large number of clarifications, during a deployment such queries could occur over a length of time (multiple days) as the robot spends time learning about its environment. Moreover, our algorithm is currently limited by ConceptNet. ConceptNet lacks rich inter-property knowledge (i.e. if an *apple* is sweet, one can assume it is also *juicy*) and the notion of classes (i.e. *sweet, sour, spicy, tangy* all belong to the same class of *taste*), the number of queries is large. However, knowledge of *material* class and good inter-property knowledge, it fared well for *bottle* where only 3 queries were asked for 9 properties or only 1 for 5 properties of *cup*. The final dissimilarity score of the vBLN to ground truth object properties is 0.11 (6 edges difference). This means that the BLN is only 6 edges (an edge is between an object and a property) away from the ground truth and managed to learn 50 out of the total 53 edges from the ground truth.

### Experiments

We break down this section into two experiments: (1) finding objects in the scene and (2) determine the properties of objects. For both, we hypothesize that the role of variability in object options is a primary factor in deciding when to use abstract vs. locally learned knowledge. If an object moves around more frequently, we should rely on reasoning about where we might find the object as opposed to remembering where was the last time or most frequently seen location. For object properties, we expect to see a similar trend.

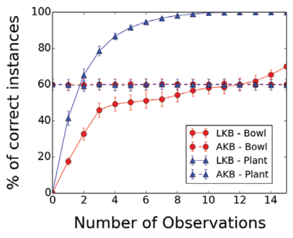


Figure 7: Average accuracy (AKB vs. LKB) across 5000 permutations to predict the top 3 locations of *potted plant* and *bowl*

**Finding Objects** For object location, we collect two separate sets of observations, one using a *bowl* and one using a *potted plant*. The two objects can be seen in Fig 1. For each object we collected 20 observations of the objects in various locations in the kitchen and living room. We determined the locations for each object using two different distributions for each object, one with more movement and one with less movement. The *bowl* was on the higher end of a variability spectrum (table1: 20%, table4: 20%, counter: 12%, table2: 12%, table3: 12%, drawer1: 12%, drawer2: 12%), while the *potted_plant* object was on the lower end (table3: 50%, table2: 25%, counter: 25%). Each time an object is detected by the robot, the object's semantic location is written to LKB.

To test and compare AKB and LKB, we randomly select 25% of the observations to leave out as the test set. This results in five observations in the test set and 15 in the train set. We test the accuracy AKB and LKB incrementally by introducing each observation separately. Specifically, we ask AKB and LKB to predict the location of the 5 observations in the test set after seeing one observations, two observations, and so on. AKB and LKB predict the locations by providing a ranked list of possible locations as described in *Data Source Selection*. We randomly select 5000 different permutations of the observation order and report the average accuracy and standard deviation to account for orderings effects. Note that the AKB is generated prior to seeing the observations as it represents general domain-free knowledge, so the accuracy of the AKB does not change over observations.

The results of this test can be seen in Fig. 7 where the robot turns the top three locations from its ranked list (simulating if the robot were allowed to look at three different locations to find the object). We can see that for the *potted plant*, the LKB reaches 80% accuracy by the fourth observation. However, for the *bowl*, the overall accuracy of the LKB reaches only 65% for top three locations, which is only slightly better than chance. When comparing AKB to LKB, it is clear that in cases where there is low variability in the current environment, learning about the object's location is superior to using general knowledge. However, for the bowl, where locations are more varied, the AKB does a better job of reasoning where in general might bowls be located. Furthermore, for both cases, when there is little to no knowledge of the scene, AKB still offers some insight to where the object might be located as opposed to LKB. We observed the same trends when testing the top-one results, although with lower overall performance rates.

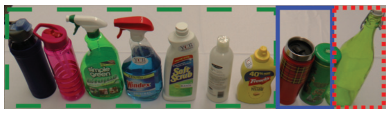


Figure 8: The bottle outlined in long green dashes, solid blue lines, and dotted red lines are plastic, metal, and glass respectively. The bottles are colored from left-to-right as blue, pink, green, blue, white, white, yellow, red, green, and green.

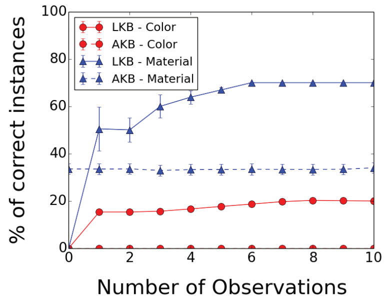


Figure 9: Average accuracy (AKB vs. LKB) across 5000 permutations for predicting the top property of 10 different bottles for two different properties (color and material)

**Object Properties** As described in *Data Source Selection*, object properties are fixed to a specific instance. As a result, we test the robot's ability to predict object properties by using a fixed test set that is also the observation set. As the robot observes its environment over time (similar to how one gets acquainted to a new environment), all of the objects in the environment will be added to its observation set. We select objects of the same class type (e.g., all bottles), to determine if knowledge properties of specific objects can provide insight on the general class of objects. Similar to object locations, we hypothesize that the variability of possible values for a property affects when and how we use our knowledge base. As a result, we select bottles with varying levels of variance within its properties (i.e., color is highly variable while materials is not). For this specific experiment, we selected 10 *bottles* (Fig. 8). Specifically, they ranged in color (green: 3, blue: 2, white: 2, yellow: 1, red: 1, pink:1) and materials (plastic: 7, metal: 2, glass: 1) with color more variable and material less.

The results of the test across the 10 bottles can be found in Fig. 9 for both color and material. We limit the AKB and LKB to just one guess as opposed to three for locations because for object properties, there is a higher threshold for errors. While searching three different locations in a home environment might take slightly longer, it is not unreasonable or dangerous for the robot to do so. On the other hand, predicting that an object is not metallic and putting it in the microwave could have dire consequences. As expected, the LKB performs poorly at predicting highly varied object properties. This makes intuitive sense as knowing that one

| Class | Location | | | | Color | | | | Material | | | | Type |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | | 15 | | 1 | | 15 | | 1 | | 15 | | |
| | AKB | LKB | AKB | LKB | AKB | LKB | AKB | LKB | AKB | LKB | AKB | LKB | AKB |
| Apple | 13 | 15 | 13 | 63 | 73 | 29 | 73 | 40 | 0 | 100 | 0 | 0 | Food, Produce, Edible fruit, Fruit, ... |
| Banana | 10 | 13 | 10 | 37 | 47 | 36 | 47 | 47 | 0 | 100 | 0 | 0 | Food, Produce, Edible fruit, Fruit, ... |
| Book | 0 | 8 | 0 | 23 | 0 | 14 | 0 | 20 | 100 | 100 | 100 | 100 | Product |
| Bottle | 3 | 6 | 3 | 20 | 0 | 20 | 0 | 27 | 73 | 27 | 73 | 73 | Container, Vessel |
| Bowl | 3 | 9 | 3 | 27 | 0 | 13 | 0 | 20 | 0 | 24 | 0 | 0 | Stadium |
| Broccoli | 17 | 16 | 17 | 60 | 80 | 69 | 80 | 80 | 0 | 100 | 0 | 0 | Food, Produce, Solid |
| Cake | 7 | 7 | 7 | 33 | 0 | 16 | 0 | 20 | 0 | 100 | 0 | 0 | Patty, Food, Dish |
| Carrot | 0 | 10 | 0 | 33 | 53 | 40 | 53 | 53 | 0 | 100 | 0 | 0 | Plant organ, Plant part |
| Chair | 17 | 8 | 17 | 40 | 0 | 14 | 0 | 20 | 33 | 28 | 33 | 33 | Instrument |
| Clock | 0 | 41 | 0 | 100 | 0 | 13 | 0 | 20 | 0 | 28 | 0 | 0 | Instrument |
| Couch | 0 | 19 | 0 | 70 | 0 | 13 | 0 | 20 | 0 | 32 | 0 | 0 | Coloring material, Covering |
| Cup | 13 | 5 | 13 | 37 | 0 | 14 | 0 | 20 | 20 | 21 | 20 | 20 | Food, Drug, Agent, Fluid |
| Donut | 7 | 4 | 7 | 30 | 0 | 21 | 0 | 33 | 0 | 100 | 0 | 0 | Food, Doughnut, Solid |
| Fork | 0 | 8 | 0 | 37 | 0 | 14 | 0 | 20 | 0 | 36 | 0 | 0 | Article, Cutlery |
| Glass | 10 | 4 | 10 | 30 | 0 | 16 | 0 | 20 | 0 | 27 | 0 | 0 | Methamphetamine, Drug, Agent |
| Knife | 0 | 8 | 0 | 13 | 0 | 15 | 0 | 20 | 0 | 40 | 0 | 0 | Instrument |
| Laptop | 0 | 10 | 0 | 30 | 0 | 22 | 0 | 40 | 33 | 33 | 33 | 33 | Machine |
| Microwave | 0 | 31 | 0 | 87 | 0 | 34 | 0 | 40 | 0 | 55 | 0 | 0 | Commodity, (Home, Kitchen) appliance |
| Orange | 0 | 7 | 0 | 27 | 0 | 33 | 0 | 33 | 0 | 100 | 0 | 0 | Coloring material |
| Oven | 77 | 32 | 77 | 100 | 0 | 40 | 0 | 53 | 0 | 61 | 0 | 0 | Commodity, (Home, Kitchen) |
| Phone | 0 | 9 | 0 | 23 | 0 | 16 | 0 | 27 | 0 | 34 | 0 | 0 | Language unit |
| Pizza | 3 | 14 | 3 | 40 | 0 | 24 | 0 | 33 | 0 | 100 | 0 | 0 | Food, Dish |
| Plant | 23 | 8 | 23 | 37 | 0 | 14 | 0 | 20 | 0 | 40 | 0 | 0 | Building complex |
| Refrigerato | 0 | 39 | 0 | 100 | 33 | 34 | 33 | 40 | 0 | 50 | 0 | 0 | Commodity, Home appliance |
| Sandwich | 30 | 11 | 30 | 37 | 13 | 22 | 13 | 33 | 0 | 100 | 0 | 0 | Food, Dish |
| Sink | 30 | 27 | 30 | 100 | 0 | 34 | 0 | 40 | 100 | 50 | 100 | 100 | Container, Vessel, Cesspool, Excavation |
| Spoon | 13 | 10 | 13 | 20 | 0 | 10 | 0 | 13 | 67 | 33 | 67 | 67 | Container, Article, Cutlery |
| Table | 20 | 13 | 20 | 40 | 7 | 16 | 7 | 27 | 27 | 22 | 27 | 27 | Food, Board |
| Toaster | 0 | 11 | 0 | 43 | 0 | 37 | 0 | 47 | 0 | 52 | 0 | 0 | Commodity, (Home, Kitchen) appliance |
| Tv | 0 | 25 | 0 | 100 | 0 | 29 | 0 | 40 | 0 | 32 | 0 | 0 | Television, Medium |
| Vase | 17 | 19 | 17 | 30 | 0 | 14 | 0 | 20 | 40 | 31 | 40 | 40 | Container, Vessel |

**Note**: Darker shading equates to higher scores. top three location, top one color property, top one material property

Figure 10: Accuracy of all class labels for location, color, material, and type.

cup is blue does not guarantee the next is blue. For material, the LKB performs well at predicting material as it captures that most bottles in the environment are plastic.

However, it is when we look at the color, that we gain interesting insight about object properties. We see that the AKB follows a slightly different trend than we observed in the object location experiment. We expected that with highly varying properties that the AKB could provide more insight than the LKB. However, if we look deeper at the results of the AKB, we discover that for the class bottle, the AKB has no prediction for color. We believe this points to an important distinction between the variability of an object property and the variability of an object location. When an object's property can take on almost any value (e.g., bottles can be pretty much any color), general knowledge offers little to no insight as to what property the object might have. Furthermore, this situation is also difficult for LKB to learn as the best we can hope for is chance. This suggests that for certain object properties, the only approach to predicting object properties that are highly variable is to remember the exact properties of the instance or perform directly reasoning using lower level features of the object. For both location and properties, we variability effects various accuracy levels of the AKB and LKB. To fully understand the extent in which this insight can be extended to a larger number of classes and properties, we perform a simulated experiment that looks at the variance accuracies across all described classes.

## Simulated Evaluation

We exhaustively evaluate how different sources of information impact the various queries listed in *Abstract Knowledge Base* using a similar procedure and experimental setup for each query to *Building the Knowledge Bases*. Specifically, we populated a simulated world of object instances, and randomly assigned attribute values (seen in Fig. 3) and locations (seen in Fig. 6). Properties and locations were made class specific to better capture the real-world (e.g., no couch instances could be located in a drawer and televisions cannot be made of paper). While the rules set in simulation may not capture the rules of a specific real-world environment, they do capture the relationship between class variability and LKB accuracy and can be viewed as a unique layout of a specific home.

### Evaluation Metric and Results

To test each query type, we start with a set of simulated instances. This set is taken as the true state of the world. Then a set of world state observations are created by randomly selecting locations and properties for each instance in the world and repeating the process for the number of world state observations. This set of world state observations were used as actual data for the LKB to process and store. To validate our hypothesis in *Experiments*, the evaluation was done similarly to that of the robot experiment where we report the top three locations and top one property. For the last query, object types (*IsA*), was tested by comparing the results of the returned values to three sets of human generated labels base on common sense for the home environment (e.g., *IsA*(Apple, Fruit) is true whereas *IsA*(Bowl, Stadium) is false).

In *Experiments*, we see a limited view of object locations and classes. By doing the simulated evaluation, we can look at if the trends seen in the robot experiment were reflected in the 31 different class types. The results of this evaluation are in Fig. 10. The table shows the accuracy of the AKB and LKB for location, color, and material by class. They are further broken down into accuracy values after seeing one observation vs seeing all 15 observations. The table also includes the different *IsA* relations for each object class.

We can see that several of the trends observed in the robot experiment hold true. For example, ovens, which are less variable in location, have a higher initial AKB accuracy than the LKB. The LKB learns the oven location perfectly after 15 observations. In general, color, which varies highly does poorly for both ABK and LKB unless the object has a notion of a color (e.g., carrot and broccoli). We see that the AKB does well on the material property if the class has a typical material it is made out of (e.g., books, sink, spoon). We test this on an aggregate scale in the next section. For the *IsA* queries, the average accuracy of the relations was 72%. Between the three sets of human labels, there was an 83.17% average pairwise percent agreement.The accuracy values between all three users were within 2% of each other. We can look at Fig. 10 to see that this accuracy can be reflected in the labels produced. It correctly identifies useful types such as apple is a food and bottle is a container. The few cases where *IsA* does not perform well can be seen with bowl being related to stadium and glass to drug.

### Role of Variability

The results show that taking into account variability of local knowledge history will be essential for reasoning about new situations. The general trend is that as variability increases, a discount factor should be used to emphasize sources of general knowledge that are resistant to such effects. Fig. 11 was generated by categorizing each simulation output seen in Fig. 10 as either low (1-3 alternatives), medium (4-6 alternatives), or high (7+ alternatives) variability and averaging

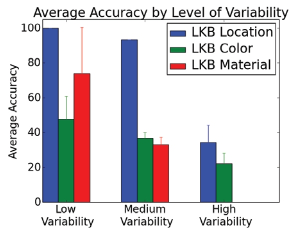


Figure 11: Relationship between LKB accuracy and variability.

all the results for each category. It shows that as variability increases, the LKB accuracy drops. For extreme levels of variability similar to in *Object Properties*, even a general knowledge systems fails. In these situations, a robot should rely on direct observations or chance.

## Conclusions

In this work we introduce the SiRoK framework and systematically evaluate it through robot experiments and simulation. We use SiRoK to better understand the trade offs between general knowledge bases that store symbols and concepts and local knowledge bases that store perceptual data. We find that variability is a key heuristic to take into account when evaluating knowledge. In future works, we hope to find methods of fusing the disparate knowledge sources, improving the quality of the BLN in our AKB, and utilizing the *IsA* query.

## References

Buschka, P., and Saffiotti, A. 2004. Some notes on the use of hybrid maps for mobile robots. In *Proc. of the 8th Int. Conf. on Intelligent Autonomous Systems*, 547–556.

Chen, J., and Liu, J. 2011. Combining ConceptNet and WordNet for Word Sense Disambiguation. In *International Joint Conference on Natural Language Processing*, 686–694.

Chu, V.; Fitzgerald, T.; and Thomaz, A. L. 2016. Learning object affordances by leveraging the combination of human-guidance and self-exploration. In *2016 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 221–228.

Fung, R., and Chang, K.-C. 2013. Weighing and Integrating Evidence for Stochastic Simulation in Bayesian Networks.

Gabrilovich, E., and Markovitch, S. 2007. Computing Semantic Relatedness using Wikipedia-based Explicit Semantic Analysis. In Veloso, M. M., ed., *International Joint Conference on Artificial Intelligence*, 1606–1611.

Garrison, H., and Chernova, S. 2016. Situated structure learning of a bayesian logic network for commonsense reasoning. *CoRR* abs/1607.00428.

Hermans, T.; Rehg, J.; and Bobick, A. 2011. Affordance prediction via learned object attributes. In *International Conference on Robotics and Automation: Workshop on Semantic Perception, Mapping, and Exploration*.

Jain, D.; Kirchlechner, B.; and Beetz, M. 2007. Extending markov logic to model probability distributions in relational domains. In *KI 2007: Advances in Artificial Intelligence*. Springer. 129–143.

Jain, D.; Waldherr, S.; and Beetz, M. 2009. Bayesian Logic Networks. Technical report, Technische Universität München, München.

Lenat, D. B. 1995. Cyc: A large-scale investment in knowledge infrastructure. *Communications of the ACM* 38(11):33–38.

Lin, T.; Maire, M.; Belongie, S. J.; Bourdev, L. D.; Girshick, R. B.; Hays, J.; Perona, P.; Ramanan, D.; Dollár, P.; and Zitnick, C. L. 2014. Microsoft COCO: common objects in context. *CoRR* abs/1405.0312.

Matuszek, C.; Cabral, J.; Witbrock, M. J.; and DeOliveira, J. 2006. An introduction to the syntax and content of cyc. In *AAAI Spring Symposium: Formalizing and Compiling Background Knowledge and Its Applications to Knowledge Representation and Question Answering*, 44–49. Citeseer.

Miller, G. A. 1995. Wordnet: A lexical database for english. *Commun. ACM* 38(11):39–41.

Redmon, J.; Divvala, S.; Girshick, R.; and Farhadi, A. 2016. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 779–788.

Richardson, M., and Domingos, P. 2006a. Markov logic networks. *Machine Learning* 62(1-2):107–136.

Richardson, M., and Domingos, P. 2006b. Markov logic networks. *Machine learning* 62(1-2):107–136.

Rother, C.; Kolmogorov, V.; and Blake, A. 2004. ”grabcut”: Interactive foreground extraction using iterated graph cuts. *ACM Trans. Graph.* 23(3):309–314.

Saxena, A.; Jain, A.; Sener, O.; Jami, A.; Misra, D. K.; and Koppula, H. S. 2014. Robobrain: Large-scale knowledge engine for robots. *arXiv preprint arXiv:1412.0691*.

Sinapov, J.; Schenck, C.; Staley, K.; Sukhoy, V.; and Stoytchev, A. 2014. Grounding semantic categories in behavioral interactions: Experiments with 100 objects. *Robotics and Autonomous Systems* 62(5):632 – 645. Special Issue Semantic Perception, Mapping and Exploration.

Speer, R., and Havasi, C. 2012. Representing General Relational Knowledge in ConceptNet 5. In *Proceedings of the Eight International Conference on Language Resources and Evaluation*.

Stoica, E., and Hearst, M. A. 2004. Nearly-Automated Metadata Hierarchy Creation. In *North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 117–120.

Sun, Y.; Bo, L.; and Fox, D. 2013. Attribute based object identification. In *2013 IEEE International Conference on Robotics and Automation*, 2096–2103.

Tenorth, M., and Beetz, M. 2009. Knowrobknowledge processing for autonomous personal robots. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, 4261–4266. IEEE.

Tsatsaronis, G.; Varlamis, I.; and Vazirgiannis, M. 2008. Word Sense Disambiguation with Semantic Networks. In Sojka, P.; Horák, A.; Kopeček, I.; and Pala, K., eds., *Text, Speech, and Dialogue*, 219–226. Springer.

Waibel, M.; Beetz, M.; Civera, J.; d'Andrea, R.; Elfring, J.; Galvez-Lopez, D.; Häussermann, K.; Janssen, R.; Montiel, J.; Perzylo, A.; et al. 2011. Roboearth. *IEEE Robotics & Automation Magazine* 18(2):69–82.

Zhu, Y.; Fathi, A.; and Fei-Fei, L. 2014a. Reasoning About Object Affordances in a Knowledge Base Representation. In *European Conference on Computer Vision*.

Zhu, Y.; Fathi, A.; and Fei-Fei, L. 2014b. Reasoning about object affordances in a knowledge base representation. In *European conference on computer vision*, 408–424. Springer.

# Validation of Hierarchical Plans via Parsing of Attribute Grammars

**Roman Barták, Adrien Maillard**
Charles University
Faculty of Mathematics and Physics
Prague, Czech Republic

**Rafael C. Cardoso**
Pontifícia Universidade Católica do Rio Grande do Sul
Porto Alegre, Brazil

**Abstract**

An important problem of automated planning is validating if a plan complies with the planning domain model. Such validation is straightforward for classical sequential planning but until recently there was no such validation approach for Hierarchical Task Networks (HTN) planning. In this paper we propose a novel technique for validating HTN plans that is based on representing the HTN model as an attribute grammar and using a special parsing algorithm to verify if the plan can be generated by the grammar.

## Introduction

Automated planning deals with the problem of finding a sequences of actions to reach a certain goal (Ghallab, Nau, and Traverso 2004). Actions are specified via preconditions and postconditions (also called effects) describing propositions that must be true in the state before action application (preconditions) and that will become true after action application (postconditions). Hence, action are a formal model of state transitions and a plan – a sequence of actions - describes a valid evolution of the world from a given initial state.

To increase efficiency of planning, Hierarchical Task Networks (HTN) were proposed to describe sets of actions as recipes for solving specific tasks (Erol, Hendler, and Nau 1996). HTN models are based on idea of decomposing compound tasks to subtasks until primitive tasks – actions – are obtained. The decomposition may include extra constraints describing precedence relations between sub-tasks and required properties of states (propositions that must hold before or between certain subtasks). The planning problem is specified as a goal task that needs to be decomposed to a sequence of actions applicable to an initial state, while satisfying all the task decomposition constraints and all the causal constraints between the actions. This sequence needs to be a valid plan in terms of causal constraints between the actions.

An important problem in automated planning is validating plans with respect to a given domain model. Such validation is easy for classical sequential planning, where it can be realised by simulating plan execution (Howey and Long 2003). However, until recently, there was no method to validate HTN plans, that is, to validate if a given plan can indeed be obtained from the goal task by some decomposition steps. There exists a recent validation method based on representing all possible decompositions as a SAT problem (Behnke, Höller, and Biundo 2017), but this method does not assume decomposition constraints (except decomposition preconditions that are compiled away to a dummy action). In this paper we suggest a more general approach that covers HTN models completely including all decomposition and causal constraints.

It has already been noted that derivation trees of Context-Free (CF) grammars resemble the structure of Hierarchical Task Networks (HTN). This has been used in (Erol, Hendler, and Nau 1996) to show the expressiveness of planning formalisms. Then, there have been some attempts to represent HTNs as CF grammars or equivalent formalisms (Nederhof, Shieber, and Satta 2003) but as demonstrated in (Höller et al. 2014), the languages defined by HTN planning problems (with partial-order, preconditions and effects) lie somewhere between CF and context-sensitive (CS) languages. In (Geib 2016), the author presents an approach with a similar intention with the help of *Combinatory Categorial Grammars* (CCGs), which are part of a category lying between CF and CS grammars, the *mildly context-sensitive grammars*. The author proposes a single model for both plan recognition and planning and he also proposes a planning algorithm based on CCGs. However, it appears that this modelling process is counter-intuitive as it requires a *lexicalization* (the hierarchical structure is contained in the terminal symbols) while the decomposition approach is more natural in planning. Also, it is not yet sure if this formalism and its planning technique can produce the full range of HTN plans. Recently, a model of HTNs based on attribute grammars has been proposed (Barták and Maillard 2017). The underlying grammar describes proper task decompositions, while a so called *timeline* constraint over the task attributes describes valid orders of actions based on causal relations. It is the only model that handles all HTN constraints including interleaving of actions. Though string shuffling used in plan recognition (Maraist 2017) allows some for of task interleaving, it is not clear how it maintains the causal constraints.

In this paper, we will use attribute grammars to validate HTN plans. We will describe how HTN domain model is represented as an attribute grammar, and for this grammar we will present a parsing technique that does plan validation. Note that due to interleaving of actions and presence of extra constraints, the parsing technique needs to be more general than classical parsing for CF grammars.

## Background on Planning

In this paper we work with classical STRIPS planning that deals with sequences of actions transferring the world from a given initial state to a state satisfying certain goal condition. World states are modelled as sets of propositions that are true in those states and actions are changing validity of certain propositions.

### Classical Planning

Formally, let $P$ be a set of all propositions modelling properties of world states. Then a state $S \subseteq P$ is a set of propositions that are true in that state (every other proposition is false). Later, we will use the notation $S^+ = S$ to describe explicitly the valid propositions in the state $S$ and $S^- = P \setminus S$ to describe explicitly the propositions that are not valid in the state $S$.

Each action $a$ is described by four sets of propositions $(B_a^+, B_a^-, A_a^+, A_a^-)$, where $B_a^+, B_a^-, A_a^+, A_a^- \subseteq P, B_a^+ \cap B_a^- = \emptyset, A_a^+ \cap A_a^- = \emptyset$. Sets $B_a^+$ and $B_a^-$ describe positive and negative preconditions of action $a$, that is, propositions that must be true and false right before the action $a$. Action $a$ is applicable to state $S$ iff $B_a^+ \subseteq S \wedge B_a^- \cap S = \emptyset$. Sets $A_a^+$and $A_a^-$ describe positive and negative effects of action $a$, that is, propositions that will become true and false in the state right after executing the action $a$. If an action $a$ is applicable to state $S$ then the state right after the action $a$ will be

$$\gamma(S, a) = (S \setminus A_a^-) \cup A_a^+. \tag{1}$$

If an action $a$ is not applicable to state $S$ then $\gamma(S, a)$ is undefined.

The classical planning problem, also called a STRIPS problem, consists of a set of actions $A$, a set of propositions $S_0$ called an initial state, and disjoint sets of goal propositions $G^+$ and $G^-$ describing the propositions required to be true and false in the goal state. A solution to the planning problem is a sequence of actions $a_1, a_2, \dots, a_n$ such that $S = \gamma(...\gamma(\gamma(S_0, a_1), a_2), ..., a_n)$ and $G^+ \subseteq S \wedge G^- \cap S = \emptyset$. This sequence of actions is called a *plan*.

### Hierarchical Task Networks as Attribute Grammars

To simplify the planning process, several extensions of the basic STRIPS model were proposed to include some control knowledge. Hierarchical Task Networks (Erol, Hendler, and Nau 1996) were proposed as a planning domain modeling framework that includes control knowledge in the form of recipes how to solve specific tasks. The recipe is represented as a task network, which is a set of sub-tasks to solve a given task together with the set of constraints between the sub-tasks. The constraints can be of the following types:

- $t_1 \prec t_2$: a precedence constraint meaning that in every plan the last action obtained from task $t_1$ is before the first action obtained from task $t_2$,
- *before*$(U, l)$: a precondition constraint meaning that in every plan the literal $l$ holds in the state right before the first action obtained from tasks $U$,
- *after*$(U, l)$: a postcondition constraint meaning that in every plan the literal $l$ will hold in the state right after the last action obtained from tasks $U$,
- *between*$(U, V, l)$: a prevailing condition meaning that in every plan the literal $l$ holds in all the states between the last action obtained from tasks $U$ and the first action obtained from tasks $V$.

In HTN, a compound task is solved by decomposing it to a task network - the connection between the task and the task network is called a (decomposition) *method*. The method can naturally be described as a rewriting rule of an attribute grammar. Attribute grammars (Knuth 1968) use the same type of rewriting rules as context-free grammars, but the grammar symbols may by annotated by attributes connected by constraints. This makes attribute grammars stronger than CF grammars in the sense of recognising a large class of languages.

Let $T(\overrightarrow{X})$ be a compound task with parameters $\overrightarrow{X}$ and $(\{T_1(\overrightarrow{X_1}), ..., T_k(\overrightarrow{X_k})\}, C)$ be a task network, where $C$ are its constraints. We can encode the decomposition method as an attribute grammar rule:

$$T(\overrightarrow{X}) \rightarrow T_1(\overrightarrow{X_1}), ..., T_k(\overrightarrow{X_k}) \;\; [C] \tag{2}$$

The planning problem in HTN is specified by an initial state (the set of propositions that hold at the beginning) and by an initial task representing the goal. The compound tasks need to be decomposed via decomposition methods until a set of primitive tasks – actions – is obtained. Moreover, these actions need to be linearly ordered to satisfy all the constraints obtained during decompositions and the obtained plan – a linear sequence of actions – must be applicable to the initial state in the same sense as in classical planning.

If we do planning by application of grammar rewriting rules, we get a linear sequence of actions (a terminal word in terms of formal grammars), but this sequence does not necessarily form a valid plan as the actions from different tasks may interleave to satisfy the ordering and causal constraints (see Figure 1). So the actions obtained by applying the grammar rules need to be re-ordered to get a valid plan. The attribute grammars model the valid action orderings via a global timeline constraint (Barták and Maillard 2017).

To give a particular example of the decomposition rule, let us assume a task to transfer a container $c$ from one location $l1$ to another location $l2$ by a robot $r$. To solve this task, we need to load the container first, then move it to its destination location, and unload it there. The following rule describes this decomposition method[1]:

$$\begin{aligned} \text{Transfer1}(c, l1, l2, r) \rightarrow\, & \text{Load-rob}(c, r, l1). \\ & \text{Move-rob}(r, l1, l2). \\ & \text{Unload-rob}(c, r, l2)[C] \end{aligned} \tag{3}$$

[1]There are several ways to model the task. For example, the *before* and *after* constraints can be omitted as they will be part of the primitive tasks.

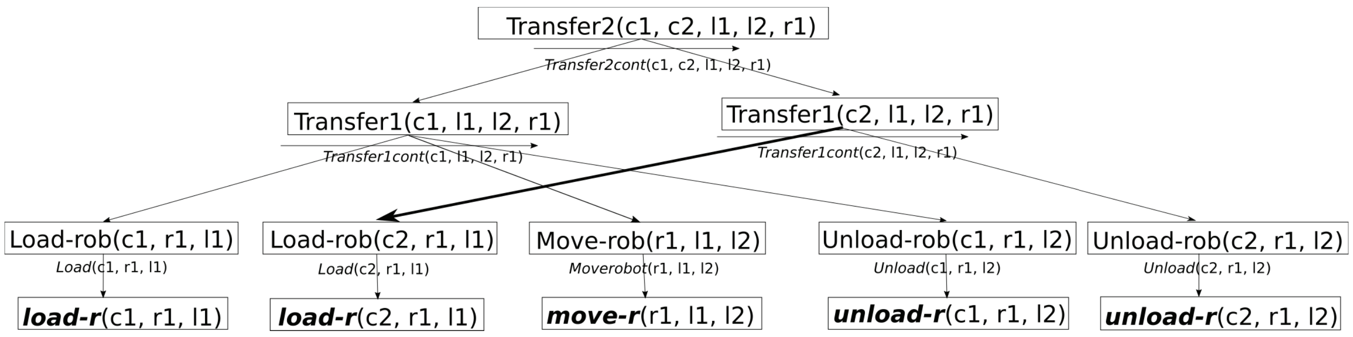


Figure 1: A task decomposition tree showing interleaving of actions obtained from decompositions of different tasks - denoted by the bold arc.

where

$$\begin{aligned} C = \{ & \text{Load-rob} \prec \text{Move-rob},\ \text{Move-rob} \prec \text{Unload-rob}, \\ & \textit{before}(\{\text{Load-rob}\}, at(r, l1)), \\ & \textit{before}(\{\text{Load-rob}\}, at(c, l1)), \\ & \textit{between}(\{\text{Load-rob}\}, \{\text{Move-rob}\}, at(r, l1)), \\ & \textit{between}(\{\text{Move-rob}\}, \{\text{Unload-rob}\}, at(r, l2)), \\ & \textit{between}(\{\text{Load-rob}\}, \{\text{Unload-rob}\}, in(c, r)), \\ & \textit{after}(\{\text{Unload-rob}\}, at(c, l2)\} \end{aligned}$$

The decomposition constraints specify the following restrictions:

- the robot and the container must be at the same location $l1$ before loading,
- the robot does not change its location between loading and the start of moving,
- the container stays in the robot between loading and unloading,
- the robot stays at the destination location $l2$ between the end of moving and the start of unloading,
- the container will be at the destination location $l2$ after unloading.

An alternative decomposition method omits the Move-rob task as it assumes that this task is introduced by decomposition of another compound task. See the task for $c2$ in Figure 1. Still, we need to ensure that the robot is at the right location before unloading, which is done by the constraint *before*$(\{\text{Unload-rob}\}, at(r, l2))$. The alternative decomposition rule looks as follows:

$$\begin{aligned} \text{Transfer1}(c, l1, l2, r) \rightarrow & \text{Load-rob}(c, r, l1). \\ & \text{Unload-rob}(c, r, l2) \\ & [C] \end{aligned} \tag{4}$$

where

$$\begin{aligned} C = \{ & \text{Load-rob} \prec \text{Unload-rob}, \\ & \textit{before}(\{\text{Load-rob}\}, at(r, l1)), \\ & \textit{before}(\{\text{Load-rob}\}, at(c, l1)), \\ & \textit{before}(\{\text{Unload-rob}\}, at(r, l2)), \\ & \textit{between}(\{\text{Load-rob}\}, \{\text{Unload-rob}\}, in(c, r)), \\ & \textit{after}(\{\text{Unload-rob}\}, at(c, l2)\} \end{aligned}$$

The top task for transferring two containers using the same robot and between the same locations can be described using the following decomposition method:

$$\begin{aligned} \text{Transfer2}(c1, c2, l1, l2, r) \rightarrow & \text{Transfer1}(c1, l1, l2, r). \\ & \text{Transfer1}(c2, l1, l2, r) \\ & [] \end{aligned} \tag{5}$$

Notice that having the $before$ and $after$ constraints allows us to describe action preconditions and postconditions as decomposition constraints rather than having them specified separately. This is done by having a compound task for each action, for example Load-rob corresponds to the primitive action load-r. This is the corresponding decomposition method:

$$\text{Load-rob}(c, r, l) \rightarrow \text{load-r}(c, r, l). \qquad [C] \tag{6}$$

where

$$\begin{aligned} C = \{ & \textit{before}(\{\text{load-r}\}, at(r, l)), \\ & \textit{before}(\{\text{load-r}\}, at(c, l)), \\ & \textit{after}(\{\text{load-r}\}, in(c, r)\} \\ & \textit{after}(\{\text{load-r}\}, \neg at(c, l)\} \end{aligned}$$

## HTN Validation Algorithm

The plan validation problem is a problem reverse to the planning problem. We have a plan as the input and the problem is to validate if that plan can be obtained by decomposition from the goal task. In terms of grammars, it means using the grammar rules in an analytical way to do parsing.

Recall that the order of actions in the plan does not necessarily correspond to the order of actions obtained by application of grammar rules. Hence, during parsing, we ignore the order of tasks on the right side of grammar rules and we model the action (task) order explicitly by using indexes assigned to tasks. Each task will be annotated by two indexes describing the order numbers of the first and the last actions obtained from task decomposition. For example, the task $\text{Load-rob}_{1,1}(c1, r1, l1)$ from Figure 1, that gives the action load-r$(c1, r1, l1)$, is annotated by indexes 1,1.

Let us now demonstrate a single parsing step. Assume that we already parsed the tasks $\text{Load-rob}_{1,1}(c1, r1, l1)$, Move-rob$_{3,3}(r1, l1, l2)$, and Unload-rob$_{4,4}(c1, r1, l2)$ and we continue in parsing using the grammar rule (3). The tasks on the right side of the rule already exist and we can verify the ordering constraints $1 \prec 3$ and $3 \prec 4$ by comparing the respective indexes. The result of the parsing step will be a new parsed task Transfer1$_{1,4}(c1, l1, l2, r1)$, where the indexes are taken as minimal and maximal indexes of its subtasks.

We still need to verify the other constraints in the rule. This will be done by maintaining a timeline for each task. The *timeline* is a sequence of slots describing validity of literals in time steps corresponding to the task. For every time step, the slot will describe the literals that hold in the state before the action at that time (a Pre part) and literals that must hold in the state right after the action (a Post part). For example, the task Load-rob$_{1,1}(c1, r1, l1)$ will use a single slot $(\{at(r1, l1), at(c1, l1)\}, \{in(c1, r1), \neg at(c1, l1)\})_1$, where the index represents time and the literals are basically preconditions and postconditions of action load-r$(c1, r1, l1)$ that were encoded as *before* and *after* constraints (see the rule (6)).

During the parsing step, we first *merge* the timelines for the subtasks with possible insertion of empty slots for times not covered by the sub-tasks (slot 2 in our example). Empty slot does not contain any action, but its Pre part may contain literals obtained by propagation (see below). Two slots with the same index can only be merged if (at least) one of them is empty. This way we ensure that each action is generated exactly once. For example, when merging timelines for tasks Transfer1$_{1,4}(c1, l1, l2, r1)$ and Transfer1$_{2,5}(c2, l1, l2, r1)$ we are merging non-empty slots 1,3,4 for the first task with non-empty slots 2, 5 of the second task. If the slots cannot be merged as they both already contain an action, then processing of the derivation rule is stopped and the algorithm continues with the next rule.

After merging the timelines for subtasks we add literals based on the rule *constraints* - for *before* and *between* constraints, the literals are added to the Pre parts of respective slots; for the *after* constraints, the literals are added to the Post parts.

After that, we *propagate* the literals between the slots. This propagation goes from left to right, where the literals from the postcondition part are added to the precondition part of the next slot and, if the slot is not empty (contains some action), the literals in preconditions, that are not deleted by the action, are added to the precondition part of the next slot. This basically follows the state transition formula as specified in (1). The right-to-left propagation adds literals in preconditions to preconditions of the previous slot provided that the slot is not empty and the literal is not added by the action in it. The goal of propagation is to keep information about states up-to-date (notice that propagation changes only the Pre parts of the slots that describe the states).

Finally, we verify that the slots are consistent, which consists of checking that no slot contains a literal and its negation in any of its parts. Table 1 demonstrates this process – it shows how literals are added to the slots in each step (slot merging, constraint addition, propagation).

The validation algorithm first transfers each action to a primitive task with the index corresponding to the order of the action and with the timeline containing a single slot with that action and empty Pre and Post parts. Recall, that preconditions and postconditions of actions will be added later during parsing using the rules of type (6). The literals of the initial state are added to the Pre part of the first slot (for simplicity, we ignored them in the previous example of a parsing step). Then the algorithm takes any grammar rule such that the tasks from its right side are already known and it does the above described parsing step. This may introduce a new parsed task. This process is repeated while some new task is introduced or until a goal task is introduced whose indexes span the whole plan. If the goal task is found then the plan is sound, otherwise, the plan is not sound. Note that the algorithm always finishes as there is only a finite number of compound tasks that can introduced during parsing. We will now describe the validation algorithm formally.

## Data structures

First we will describe the data structures that are used later in the algorithm. Basically, we will introduce slots, timelines, and the parsed tasks :

We define the type `slot` as a tuple $(\text{Pre}^+, \text{Pre}^-, a, \text{Post}^+, \text{Post}^-)$ where

- $\text{Pre}^+$ is a set of atoms (positive propositions in the state)
- $\text{Pre}^-$ is a set of atoms (negative propositions in the state)
- $a \in A \cup \{\textit{empty}\}$ is an action name (or an empty slot)
- $\text{Post}^+$ is a set of atoms (positive postconditions of $a$)
- $\text{Post}^-$ is a set of atoms (negative postconditions of $a$)

To simplify verification of slot/timeline soundness we use separate sets for positive and negative propositions. Note also that the sets $\text{Pre}^+, \text{Pre}^-$ are not only related to action $a$ but they will describe the state right before the action. More precisely, these sets describe the propostions that must hold in the state, but until all slots are non-empty, the state may be described only partially (see Table 1).

Then, we define the type `subplan` that represents a parsed task $T$ as a tuple $(T, b, e, timeline)$ with

- $T$ being a task name,
- $b$ and $e$ ($b \leq e$) being two integers equal to the indexes in the original plan of the first and last actions in the subplan generated from $T$; this pair shows how much the subplan generated from $T$ *spans over* the verified plan,
- $timeline$ being an ordered sequence of $(e - b + 1)$ elements of the `slot` type; we have $timeline = \{s_b, ..., s_e\} \subseteq \textbf{slots}$.

## The algorithm formally

The validation algorithm is shown in Algorithm 1. At the beginning, actions in the plan are put individually in the set **subplans** (line 2). They are all subplans of size 1. The initial state is added to the Pre parts of the slot of the first action. Then, at each iteration the algorithm fires rules in the grammar where all subtasks are elements of **subplans**. When

Table 1: The process of building a timeline during parsing the compound task $\text{Transfer1}_{1,4}(c1, l1, l2, r1)$.

| | 1: load-r$(c1, r1, l1)$ | | 2: *empty* | | 3: move-r$(r1, l1, l2)$ | | 4: unload-r$(c1, r1, l2)$ | |
|---|---|---|---|---|---|---|---|---|
| | $Pre_1$ | $Post_1$ | $Pre_2$ | $Post_2$ | $Pre_3$ | $Post_3$ | $Pre_4$ | $Post_4$ |
| merge | $at(r1, l1)$ $at(c1, l1)$ | $in(c1, r1)$ $\neg at(c1, l1)$ | | | $at(r1, l1)$ | $\neg at(r1, l1)$ $at(r1, l2)$ | $in(c1, r1)$ $at(r1, l2)$ | $\neg in(c1, r1)$ $at(c1, l2)$ |
| constrain | | | $at(r1, l1)$ $in(c1, r1)$ | | $in(c1, r1)$ | | | |
| propagate | | | $\neg at(c1, l1)$ | | | | $\neg at(r1, l1)$ | |

such a rule is found, the precedence constraints are checked (line 7). Then the timelines of subtasks are merged (line 8) and before, after, and between constraints from the grammar rule are applied to this merged timeline (lines 9, 10, and 11). Preconditions and postconditions are then propagated from left to right and from right to left (line 12). Finally, the resulting timeline is verified (13). If no inconsistency is detected, then the new parsed task is added to the set **subplans** so it can be further used for building a higher-level task. Inconsistency means that some atom is both in the positive and in the negative parts of the state.

The positive exit condition (cf. Algorithm 2) is met when there is a *Goal* task in **subplans** that contains all the elements of the verified plan $\mathbf{P}$.

If, it is not possible to find a rule that applies to the current elements of **subplans** and produces a *new* subplan, then it means that the plan $\mathbf{P}$ is not valid with regard to the grammar. In other words, the set **subplans** has not grown during the execution of the for-loop (lines 6 to 18). At this point, the algorithm returns false (line 20).

We also include all the sub-procedures for merging the timelines and for applying the constraints. To simplify notations in the procedures for constraint application (Algorithms 5-7), we use the following notation – if $l$ is a positive literal $p$ then $l^+ = \{p\}$ and $l^- = \{\}$; if $l$ is a negative literal $\neg p$ then $l^+ = \{\}$ and $l^- = \{p\}$.

## Soundness

We shall now show that the algorithm correctly recognises plans that can be derived from a given *Goal* task and an initial state.

First, one should realise that the algorithm always finishes. All sub-procedures clearly finish as they consist of *for* loops and *if-then-else* conditions only. During each iteration of the main *while* loop, some new task may be added to the set of **subplans**. The input plan is finite and we have only a finite number of constants so the number of tasks that can be derived is obviously finite. Hence the *while* loop must finish sometime, either when no new task is added (line 20) or when the *Goal* task is derived (line 5).

Assume that the algorithm finished successfully (with the answer **true**). It means that it found the *Goal* task that spans over the full plan (test in Algorithm 2). By reconstructing how this task was added to the set **subplans**, we get the derivation tree (such as the one in Figure 1). We indeed get a tree as during merging of timelines, two slots can only be merged if at least one of them is empty. Hence each task in the tree has exactly one parent. If the same task appears two (or more) times in the tree then its slots would eventually merge with themselves, which is not possible (see Algorithm 4). All the constraints used in this derivation (decomposition) are satisfied as the algorithm verified the precedence constraints and added the literals from the before, after, and between constraints to the timeline, which is consistent.

Notice that the $Post$ parts of the slots in the timeline contain only the propositions from the after constraints so they model the effects of actions. The $Pre$ parts (in particular the $\text{Pre}^+$ sets) model the states between the actions and we shall show that the sequence of states is correct with respect to the plan. First, each state is sound as it does not contain an atom and its negation ($\text{Pre}^+ \cap \text{Pre}^- = \emptyset$). Next, two subsequent states $\text{Pre}_i^+$ and $\text{Pre}_{i+1}^+$ model a correct state transition thanks to the propagation:

$$\text{Pre}_{i+1}^+ = (\text{Pre}_i^+ \setminus \text{Post}_i^-) \cup \text{Post}_i^+$$

$$\text{Pre}_{i+1}^- = (\text{Pre}_i^- \setminus \text{Post}_i^+) \cup \text{Post}_i^-$$

This realises the state transition formula (1). We will show it for the positive part of the state (the proof is identical for the negative part). Assume slots $i$ and $i+1$ with some action filled in the slot $i$ (the action must appear there eventually as the final timeline has all slots non-empty). Thanks to left-to-right propagation, it must hold $\text{Post}_i^+ \subseteq \text{Pre}_{i+1}^+$ (line 5 of Algorithm 8) and $\text{Pre}_i^+ \setminus \text{Post}_i^- \subseteq \text{Pre}_{i+1}^+$ (line 8 of Algorithm 8). Thanks to right-to-left propagation, it must hold $\text{Pre}_{i+1}^+ \setminus \text{Post}_i^+ \subseteq \text{Pre}_i^+$ (line 14 of Algorithm 8). It means that if a proposition $p \in \text{Pre}_{i+1}^+$ is not added by the action ($p \notin \text{Post}_i^+$) then $p$ must already be part of the previous state ($p \in \text{Pre}_i^+$). Together, we get:

$$\text{Pre}_{i+1}^+ = (\text{Pre}_i^+ \setminus \text{Post}_i^-) \cup \text{Post}_i^+ \tag{7}$$

Notice that the algorithm works even when no initial state is provided. Then the final sets $\text{Pre}_1^+$ and $\text{Pre}_1^-$ specify the propositions that must and must not be valid at the beginning to have a valid plan. If the initial state is provided then it is propagated through the slots.

In summary, the set of actions in the plan is generated by the grammar and forms a valid plan.

If the algorithm finishes with the answer **false** then no derivation exists as no other task can be parsed. Being the plan correct, the derivation tree would be reconstructed by the algorithm as the algorithm finds all the tasks that decompose to any subset of the plan.

```
Data: a plan $\mathbf{P} = (a_1, ..., a_n)$, initial state InitState, a goal
      task Goal, an attribute grammar
      $G = (\Sigma, N, \mathcal{P}, S, A, C)$
Result: a boolean equal to true if the plan can be derived
        from the hierarchical structure, false otherwise
1  Function VerifyPlan
       /* Initialization of the set of
          subplans                                  */
2      $\mathbf{subplans} \leftarrow \{(a_i, i, i, \{(\emptyset, \emptyset, a_i, \emptyset, \emptyset)_i\}) | a_i \in \mathbf{P}\}$ ;
3      $\mathrm{Pre}_1^+ \leftarrow InitState^+$;
4      $\mathrm{Pre}_1^- \leftarrow InitState^-$;
5      while $\neg$PlanIsValid($\mathbf{subplans}, \mathbf{P}, Goal$) do
6          for each rule $R$ in $\mathcal{P}$ of the form
             $T_0 \to T_1, ..., T_k\ [\prec, pre, post, btw]$ such that
             $subtasks = \{(T_i, b_i, e_i, tl_i) | i \in 1..k\} \subseteq$
             $\mathbf{subplans}$ do
7              verify $\prec$ from rule $R$ else break;
8              $timeline \leftarrow$ MergePlans($subtasks$);
9              ApplyPre($timeline, pre$);
10             ApplyPost($timeline, post$);
11             ApplyBetween($timeline, btw$);
12             Propagate($timeline$);
13             if $\exists(\mathrm{Pre}^+, \mathrm{Pre}^-, a, \mathrm{Post}^+, \mathrm{Post}^-) \in$
                 $timeline, \mathrm{Pre}^+ \cap \mathrm{Pre}^- \neq$
                 $\emptyset \vee \mathrm{Post}^+ \cap \mathrm{Post}^- \neq \emptyset$ then
14                 break
15             end
16             $b = \min_{(T_i, b_i, e_i, tl_i) \in subtasks} b_i$,;
17             $e = \max_{(T_i, b_i, e_i, tl_i) \in subtasks} e_i$;
18             $\mathbf{subplans} \leftarrow$
                 $\mathbf{subplans} \cup \{(T_0, b, e, timeline)\}$;
19         end
20         if size of $\mathbf{subplans}$ has not increased since the
             last iteration then
21             return false
22         end
23     end
24     return true
25 end
```

**Algorithm 1:** Verification procedure

We showed that the algorithm always finishes. If it returns **true** then the plan can be derived from the *Goal* task. If it returns **false** then the plan cannot be derived from the *Goal* task. Hence the algorithm validates the plans with respect to the domain model.

```
Data: the set of subplans: $\mathbf{subplans}$, the plan to be
      validated $\mathbf{P}$, the goal task $Goal$
Result: true or false
1  Function PlanIsValid
2      return $(\exists(Goal, 1, |\mathbf{P}|, timeline) \in$
         $\mathbf{subplans}, s.t. \bigcup_{(\_,\_,a_i,\_,\_) \in timeline}\{a_i\} = \mathbf{P})$
3  end
```

**Algorithm 2:** The end condition of the valid plan

```
Data: a set of subplans : $subplans$
Result: a set of slots $newtimeline$, the aggregation of the
        slots of every subplan
1  Function MergePlans($subplans$)
2      $lb = \min_{(T_i, b_i, e_i, timeline_i) \in subplans} b_i$;
3      $ub = \max_{(T_i, b_i, e_i, timeline_i) \in subplans} e_i$;
4      $newtimeline \leftarrow \{(\emptyset, \emptyset, empty, \emptyset, \emptyset)_i | i \in lb..ub\}$;
5      for $(T, b, e, timeline) \in subplans$ do
6          for $s_k \in timeline, s_k^{'} \in newtimeline$ do
7              $s_k^{'} \leftarrow$ MergeSlots($s_k, s_k^{'}$)
8          end
9      end
10     return $newtimeline$
11 end
```

**Algorithm 3:** Merge timelines

## Initial Experiments

In this section we report some initial experiments comparing the performance of the implementation of our algorithm against the PANDA verifier, described in (Behnke, Höller, and Biundo 2017). The PANDA verifier validates a plan by translating it into a SAT formula. This translation requires a bound, the maximum height of the decomposition that any candidate for a solution plan can have.

In these experiments we use the Transport domain, initially introduced in the International Planning Competition (IPC) of 2008, but without action costs. In this domain, each vehicle can transport packages between different locations based on road connections. Our implementation is able to parse directly from SHOP2 planner's (Nau et al. 2003) input files, arguably one of the most used HTN planner. At the moment, we only support basic HTN syntax from SHOP2, but we are gradually adding support for many SHOP2 commands and tags. PANDA verifier uses its own input, which is a PDDL-like representation of HTN.

Our Transport domain description in SHOP2 syntax contains three primitive tasks and three non-primitive tasks. The description used in PANDA verifier has four primitive tasks and six non-primitive tasks. The extra primitive task is a *noop* action, which in our description is encoded directly as a non-primitive task. The extra non-primitive tasks from PANDA's description are dummy methods that represent primitive tasks.

We ran 5 different problem instances and collected the total CPU times. These times include any parsing done by both approaches, and was calculated from the start to the end of each validation. To run these experiments we used a virtual machine (Oracle VM VirtualBox Version 5.1.22) running an Ubuntu 16.04 LTS, with 4 GB of memory and an Intel Core i7-4700MQ processor with 4 cores and 8 threads. Our implementation requires Ruby (we used version 2.3.1), while the PANDA verifier requires Java (we used OpenJDK 1.8) and the MiniSat solver (we used version 2.2.1).

Table 2 shows the initial results comparing our attribute grammar approach with PANDA verifier using the transport domain (with no action cost). The first problem instance ($p1$) has a solution plan with 8 actions, and an initial state with 15 ground atoms. Each subsequent problem instance has the following number of actions and number of ground atoms: 12 and 29; 16 and 45; 19 and 60; 22 and 80. Odd problems ($p1$, $p3$, and $p5$) had valid solutions, while even problems ($p2$, and $p4$) had not.

```
Data: two slots
    $s_1 = (\mathrm{Pre}_1^+, \mathrm{Pre}_1^-, a_1, \mathrm{Post}_1^+, \mathrm{Post}_1^-)$, $s_2 = (\mathrm{Pre}_2^+, \mathrm{Pre}_2^-, a_2, \mathrm{Post}_2^+, \mathrm{Post}_2^-)$
Result: merged slots
1  Function MergeSlots($s_1, s_2$)
2      if $a_1 = empty$ or $a_2 = empty$ then
3          $\mathrm{Pre}^+ = \mathrm{Pre}_1^+ \cup \mathrm{Pre}_2^+$;
4          $\mathrm{Pre}^- = \mathrm{Pre}_1^- \cup \mathrm{Pre}_2^-$;
5          $\mathrm{Post}^+ = \mathrm{Post}_1^+ \cup \mathrm{Post}_2^-$;
6          $\mathrm{Post}^- = \mathrm{Post}_1^- \cup \mathrm{Post}_2^-$;
7          $a = a_1(if\ a_2 = empty)\ or\ a_2(if\ a_1 = empty)$;
8          return $(\mathrm{Pre}^+, \mathrm{Pre}^-, a, \mathrm{Post}^+, \mathrm{Post}^-)$
9      end
10     break
11 end
```

**Algorithm 4:** Merge slots

```
Data: a set of slot : $slots$, a set of $before$ constraints
Result: an updated set of slots
1 Function ApplyPre($slots, pre$)
2     for $before(U, l) \in pre$ do
3         $id = \min\{b_i | T_i \in U\}$;
4         $\mathrm{Pre}_{id}^+ \leftarrow \mathrm{Pre}_{id}^+ \cup l^+$;
5         $\mathrm{Pre}_{id}^- \leftarrow \mathrm{Pre}_{id}^- \cup l^-$
6     end
7 end
```

**Algorithm 5:** Apply before constraints

```
Data: a set of slot : $slots$, a set of $after$ constraints
Result: an updated set of slots
1 Function ApplyPost($slots, post$)
2     for $after(U, l) \in post$ do
3         $id = \max\{e_i | T_i \in U\}$;
4         $\mathrm{Post}_{id}^+ \leftarrow \mathrm{Post}_{id}^+ \cup l^+$;
5         $\mathrm{Post}_{id}^- \leftarrow \mathrm{Post}_{id}^- \cup l^-$
6     end
7 end
```

**Algorithm 6:** Apply after constraints

```
Data: a set of slot : $slots$, a set of $between$ constraints
Result: an updated set of slots
1  Function ApplyBetween($slots, between$)
2      for $between(U, V, l) \in between$ do
3          $s = \max\{e_i | T_i \in U\} + 1$;
4          $e = \min\{b_i | T_i \in V\}$;
5          for $id = s$ to $e$ do
6              $\mathrm{Pre}_{id}^+ \leftarrow \mathrm{Pre}_{id}^+ \cup l^+$;
7              $\mathrm{Pre}_{id}^- \leftarrow \mathrm{Pre}_{id}^- \cup l^-$
8          end
9      end
10 end
```

**Algorithm 7:** Apply between constraints

For these initial experiments, our approach appear to scale linearly when the solution is valid, but takes a bit more time if it is not valid, as shown in Figure 2. PANDA verifier had an exception on $p2$, because it does not seem to allow invalid transitions, but instead of ignoring that decomposition path, it crashes with an exception. And in $p5$, Panda returned that the plan was not valid, which was incorrect.

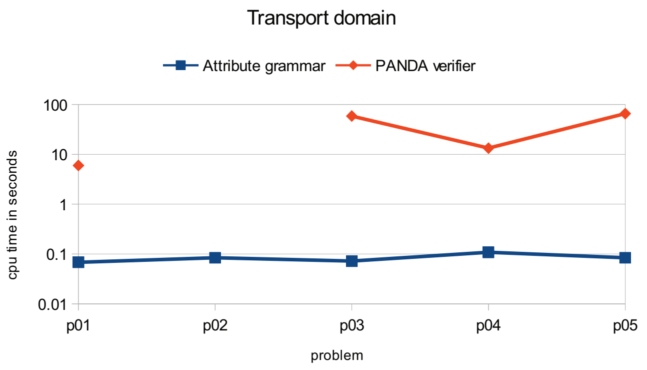


Figure 2: Transport domain results.

## Conclusions

In this paper we proposed an algorithm for validating HTN plans by using parsing of an attribute grammar describing the HTN domain model. The algorithm mimics classical parsing of context-free grammars customised to attribute grammars with the timeline constraint.

The algorithm starts with the plan and applies the decomposition rules in a reverse order to group actions into tasks. The decomposition constraints are verified by keeping information about propositions that must be true at states before and after actions. The algorithm stops when it finds a task that covers the complete plan. Then the plan is valid. The other way of stopping the algorithm is when no other compound task can be constructed. In such a case the plan does not correspond to any task. Note, that the plan might still be a correct sequence of actions but it cannot be obtained by decomposition of any task.

The major innovation of the proposed technique is that it is the first approach that covers HTN models fully including interleaving of actions and various decomposition constraints. In particular, the proposed algorithm is more general than an existing SAT-based approach (Behnke, Höller, and Biundo 2017) in covering precedence, before, between, and after constraints. The SAT-based approach only covers specific before constraints (the constraint is applied to the set of all tasks on the right side of the rule) that must be encoded as dummy actions. These dummy actions must be part of the plan to be validated so for the original plan to be validated one must find proper places, where to insert these dummy actions, which is not discussed in (Behnke, Höller, and Biundo 2017).

Furthermore, our initial experiments indicate that converting HTN models to attribute grammars may provide better performance results for validating plans, rather than converting to SAT. More experiments with other domains are needed to ascertain in which types of domain each approach performs better.

As other planning models such as procedural domain con-

```
Data: a set of slots $slots$
Result: an updated set of slots
1  Function PROPAGATE($slots$)
2      $lb = \min_{(\mathrm{Pre}_j^+, \mathrm{Pre}_j^-, a_j, \mathrm{Post}_j^+, \mathrm{Post}_j^-) \in slots} j$;
3      $ub = \max_{(\mathrm{Pre}_j^+, \mathrm{Pre}_j^-, a_j, \mathrm{Post}_j^+, \mathrm{Post}_j^-) \in slots} j - 1$;
       /* Propagation to the right */
4      for $i = lb$ to $ub$ do
5          $\mathrm{Pre}_{i+1}^+ \leftarrow \mathrm{Pre}_{i+1}^+ \cup \mathrm{Post}_i^+$;
6          $\mathrm{Pre}_{i+1}^- \leftarrow \mathrm{Pre}_{i+1}^- \cup \mathrm{Post}_i^-$;
7          if $a_i \neq$ empty then
8              $\mathrm{Pre}_{i+1}^+ \leftarrow \mathrm{Pre}_{i+1}^+ \cup (\mathrm{Pre}_i^+ \setminus \mathrm{Post}_i^-)$;
9              $\mathrm{Pre}_{i+1}^- \leftarrow \mathrm{Pre}_{i+1}^- \cup (\mathrm{Pre}_i^- \setminus \mathrm{Post}_i^+)$
10         end
11     end
       /* Propagation to the left */
12     for $i = ub$ downto $lb$ do
13         if $a_i \neq$ empty then
14             $\mathrm{Pre}_i^+ \leftarrow \mathrm{Pre}_i^+ \cup (\mathrm{Pre}_{i+1}^+ \setminus \mathrm{Post}_i^+)$;
15             $\mathrm{Pre}_i^- \leftarrow \mathrm{Pre}_i^- \cup (\mathrm{Pre}_{i+1}^- \setminus \mathrm{Post}_i^-)$
16         end
17     end
18 end
```

**Algorithm 8:** Propagate

Table 2: Initial results of experiments comparing CPU run time, in seconds.

| **transport domain** | **p01** | **p02** | **p03** | **p04** | **p05** |
|---|---|---|---|---|---|
| | CPU time | CPU time | CPU time | CPU time | CPU time |
| *Attribute grammar* | 0.068 | 0.084 | 0.072 | 0.108 | 0.084 |
| *PANDA verifier* | 5.968 | - | 58.52 | 13.32 | 65.56 wrong |

trol knowledge (Baier, Fritz, and McIlraith 2007) can be translated to attribute grammars (Barták and Maillard 2017) the proposed algorithm can verify plans with respect to these models too.

Our current implementation of the algorithm uses a straightforward approach to find rules used for parsing. The more efficient implementation of the algorithm may exploit principles of the Rete algorithm (Forgy 1982) used for production rule systems.

## Acknowledgments

Research is supported by the Czech Science Foundation under the project P103-15-19877S.

## References

Baier, J. A.; Fritz, C.; and McIlraith, S. A. 2007. Exploiting Procedural Domain Control Knowledge in State-of-the-Art Planners. In Boddy, M. S.; Fox, M.; and Thiébaux, S., eds., *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling, ICAPS 2007, Providence, Rhode Island, USA, September 22-26, 2007*, 26–33. AAAI.

Barták, R., and Maillard, A. 2017. Attribute grammars with set attributes and global constraints as a unifying framework for planning domain models. In *Proc. of the ICAPS Workshop on Knowledge Engineering for Planning and Scheduling, KEPS 1017*, 45–53.

Behnke, G.; Höller, D.; and Biundo, S. 2017. This is a solution! (... but is it though?) verifying solutions of hierarchical planning problems. In *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling, ICAPS 2017)*, 20–28.

Erol, K.; Hendler, J. A.; and Nau, D. S. 1996. Complexity Results for HTN Planning. *Ann. Math. Artif. Intell.* 18(1):69–93.

Forgy, C. L. 1982. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence* 19(1):17 – 37.

Geib, C. 2016. Lexicalized reasoning about actions. *Advances in Cognitive Systems* 4:187–206.

Ghallab, M.; Nau, D. S.; and Traverso, P. 2004. *Automated planning - theory and practice*. Elsevier.

Howey, R., and Long, D. 2003. VAL's Progress: The Automatic Validation Tool for PDDL2.1 used in the International Planning Competition. In *Proceedings of ICAPS'03 Workshop on the Competition: Impact, Organization, Evaluation, Benchmarks*.

Höller, D.; Behnke, G.; Bercher, P.; and Biundo, S. 2014. Language Classification of Hierarchical Planning Problems. In Schaub, T.; Friedrich, G.; and O'Sullivan, B., eds., *ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014)*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, 447–452. IOS Press.

Knuth, D. E. 1968. Semantics of Context-Free Languages. *Mathematical Systems Theory* 2(2):127–145.

Maraist, J. 2017. String shuffling over a gap between parsing and plan recognition. In *The AAAI-17 Workshop on Plan, Activity, and Intent Recognition WS-17-13*, 835–842.

Nau, D.; Ilghami, O.; Kuter, U.; Murdock, J. W.; Wu, D.; and Yaman, F. 2003. Shop2: An htn planning system. *Journal of Artificial Intelligence Research* 20:379–404.

Nederhof, M.-J.; Shieber, S.; and Satta, G. 2003. Partially ordered multiset context-free grammars and ID/LP parsing. *Proceedings of the Eighth International Workshop on Parsing Technologies* 171–182.

# Perspectives on the Validation and Verification of Machine Learning Systems in the Context of Highly Automated Vehicles

**Werner Damm**
C. v. Ossietzky University
26111 Oldenburg, Germany

**Martin Fränzle**
C. v. Ossietzky University
26111 Oldenburg, Germany

**Sebastian Gerwinn**
OFFIS e. V.
Escherweg 2, 26121 Oldenburg

**Paul Kröger**
C. v. Ossietzky University
26111 Oldenburg, Germany

**Abstract**

Algorithms incorporating learned functionality play an increasingly important role for highly automated vehicles. Their impressive performance within environmental perception and other tasks central to automated driving comes at the price of a hitherto unsolved functional verification problem within safety analysis. We propose to combine statistical guarantee statements about the generalisation ability of learning algorithms with the functional architecture as well as constraints about the dynamics and ontology of the physical world, yielding an integrated formulation of the safety verification problem of functional architectures comprising artificial intelligence components. Its formulation as a probabilistic constraint system enables calculation of low risk manoeuvres. We illustrate the proposed scheme on a simple automotive scenario featuring unreliable environmental perception.

Modern AI and especially machine learning (ML) components are believed to be a key enabler for bringing highly automated driving functions at SAE levels 4 to 5 (SAE and others 2014) onto the market. Before such systems can be released, obtaining a rigorous guarantee of their safety is essential: systematic faults within the design (including the training phase of ML based algorithms) could have dramatic effects on the overall safety of the mass-marketed system implementations and hence also for their societal acceptance. A key challenge for this verification is the inherent uncertainty involved in object identification. To illustrate the impact of such uncertainties, consider the following artifical example of a misperception (see Fig. 1).

At time $t_0$, the EGO vehicle (E) has detected another vehicle $v_1$ on the left lane using information from a camera and RADAR sensors. At a later time instant $t_1$, the vehicle $v_1$ has closed the gap to EGO and consequently is detected still. Additionally, another vehicle $v_2$ has been detected at very short distance in front of EGO, while another detector has recognized the presence of a bridge in front. In this situation, EGO is confronted with the decision to either perform an overtaking manoeuvre – thereby risking a collision with $v_1$, or to perform an emergency brake to mitigate a potential collision with vehicle $v_2$. A third option would be to perform an evasive manoeuvre to the right, thereby risking a collision with a bridge pillar. Note that at $t_0$, the space in

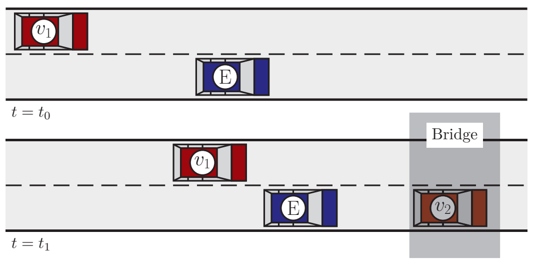


Figure 1: Example scenario. Perception of the environment is considered at two distinct time instants $t_0$ and $t_1$.

front of the EGO vehicle has been perceived as free. In this scenario, we assume that the time gap $t_1 - t_0$ is insufficient for a vehicle $v_2$ to be outside warning range at $t_0$ and to get to the position (and speed) perceived for $v_2$ at $t_1$, given the physical constraints on vehicle dynamics. Thus, the results of the different detectors evidently are contradictory.

To choose an acceptable manoeuvre, a careful assessment of the risks on a vehicle level is necessary – for example by quantifying possible outcomes of a decision using injury risk scales, like AIS or ISS (MacKenzie, Shapiro, and Eastham 1985). Individual ML components, however, are traditionally evaluated using component level loss functions (Cesa-Bianchi, Conconi, and Gentile 2004). Using the common 0-1 loss ($l_{1\text{-}0}$), the resulting risk at the component level can be interpreted as bound on the probability of correctly classifying a random input (distributed according to a fixed but unknown distribution):

$$1 - \mathbb{E}[l_{1\text{-}0}] = P(\text{correctly classified}) \in [\underline{p}(\delta), \overline{p}(\delta)] \quad (1)$$

where the right hand side denotes the confidence interval as obtained from the available bounds, i.e. via cross-validation or generalisation bounds such as within the Probably-Almost-Correct (PAC) framework. These bounds in turn depend on the confidence level $\delta$. Under the assumption that any new data (different from the training data) would be generated according to the same probability distribution which also generated the training data, a generalisation statement can be formulated and proven which provides the desired bound on the true risk.

In order to use such information to assess the risk on vehicle level, we propose a layered approach integrating the individual ML components into a constraint system which includes prior knowledge about physical properties and the functional architecture. The resulting architecture thereby combines features from probabilistic graphical models (Koller and Friedman 2009) capturing probabilistic relationships with features from non-deterministic constraint systems. We consequently employ the same definition of risk as used in reliability and utility theory (expected loss), yet permit underspecification of the probability distribution determining the expected values of interest. Among the possible instants of the underspecified distribution, we aim at calculating worst-case expectations. This permits to compute *robust* low-risk manoeuvres at runtime, whereby individual performance assessment in terms of the empirical risk at component level can be combined with the obtained constraint system to bound the overall risk at vehicle level.

In the following, we will illustrate the proposed approach on the above example, thereby illustrating its potential.

## The Probabilistic Constraint System

In the example of Fig. 1, we are interested in the following analysis questions: Can we compute a robust low-risk manoeuvre for EGO at $t_1$, which keeps risk adequately bounded despite potentially uncertain information? Given such a robust manoeuvre, can we quantify the worst-case residual risk associated with such controller?

To answer such questions, we first construct a constraint system reflecting assumed knowledge as well as imperfect information about the underlying situation. To this end, we try to build a probabilistic system similar to a dynamic Bayesian network (Murphy and Russell 2002). In practice, we sometimes have to admit unknown dependencies not expressible in standard Bayesian networks. For such dependencies, we possess no explicit probability distribution, but can only model constraints. We illustrate such a constraint system in Fig. 2, where the functional architecture is reflected on the left side whereas information about the real world is depicted on the right side. In the following, we refer to each signal or measurement (nodes within the figure) as variables, which can be interpreted as (possibly Dirac distributed) random variables.

We assume that EGO's sensor system provides a glare detector, a bridge detector, and a vehicle detector tracking multiple vehicles. The result of each detector is an observed variable within a Bayesian network (left side of Fig. 2). As the environment and hence also the observation thereof evolves over time, each variable is also annotated with a time index $t_0$, $t_1$ (represented as shaded duplicates of the nodes). We assume the functional architecture to be given. Hence, the Bayesian Network on the left side can be constructed with known dependencies (illustrated as thin arrows). These can contain safety mechanisms like the "Fused Vehicle Detection", which employs detection of glare to improve raw object detection by situationally reducing the importance of camera-based detection. As these are only percepts of objects, corresponding real-world counterparts are modeled on the right side. Within the dynamic Bayesian network, these counterparts act as latent variables of which dependencies and probability distributions are unknown to us. Labeled test data, however, provide values for these variables on an individual data-point basis. Physical dynamical constraints, if available, furthermore restrict their possible evolution over time. Both types of information yield an overall constraint system confining possible instantiations of the unknown distributions and thus permitting to assess worst-case (across possible instantiations) residual risk of the resulting system.

### Probabilistic constraints

Using access to ground truth data from manual labeling, probabilistic constraints can be derived in terms of component based performance (Eq. 1) using standard test-scores. Within our example, the performance of vehicle detection could specify a constraint on the conditional probability

$$P\left(\widehat{v}_i \mid \text{Glare } \wedge v_i \wedge \text{Bridge}\right) \in \hat{p} \pm \epsilon(\delta) \ , \qquad (2)$$

where $\hat{p}$ denotes the empirical performance, $\epsilon(\delta)$ denotes the accuracy of such an estimate depending on the confidence level $\delta$, and $v_i$ denotes vehicle $v_i$'s actual presence whereas $\widehat{v}_i$ represents that $v_i$ was detected. Analogously, fluctuations of sensor readings can be described as probability distributions conditioned on environmental states. Although some (in-)dependence connections might be known, the explicit probability distribution might be unknown. Therefore, instead of fully specifying a dynamic belief network over all discrete and continuous variables, we only collect an incomplete set of constraints of the form of Eq. (2). This necessitates an optimisation over the possible instantiations of such underspecified distributions when calculating a safe bound on the residual risk.

### Dynamic constraints

In addition to such probabilistic constraints originating from individual component tests, prior knowledge about the dynamics can be incorporated (blue box 'dynamic constraints' in Fig. 2). The detected positions of vehicles $v_1$ and $v_2$ can for example be constrained via kinematic constraints of the vehicles. Such constraints can be represented as follows, where $\ell_i(t)$ denotes the position of vehicle $i$ at time $t$ and $\overline{v}, \overline{a}$ are intervals containing minimal and maximal values for velocity and acceleration:

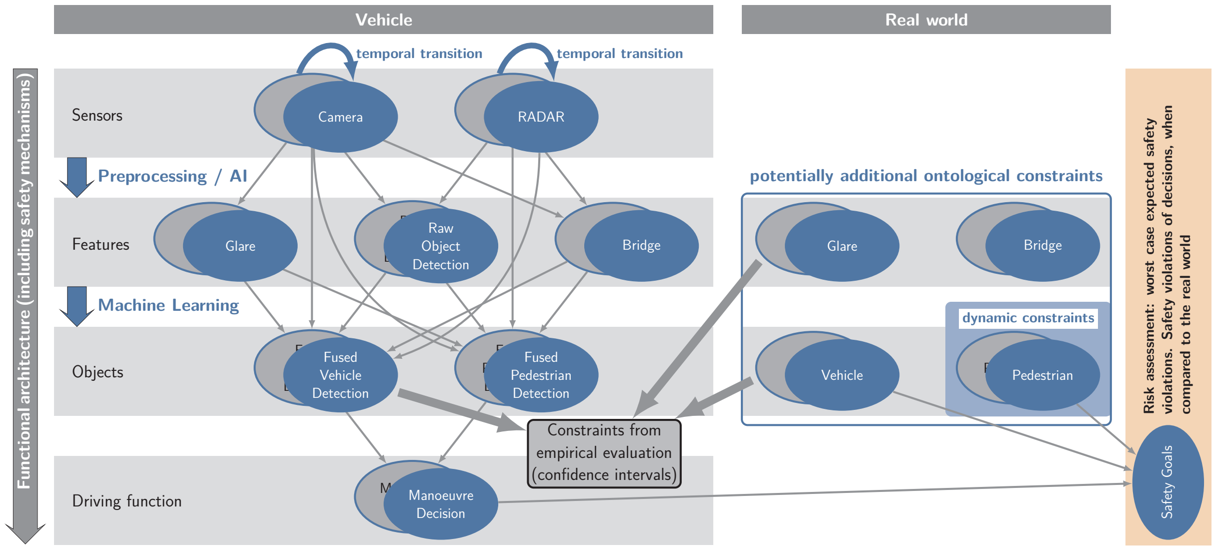
$$\ell_i(t + \Delta t) \in \left(\ell_i(t) + (\Delta t \overline{v} + \frac{1}{2}\overline{a}(\Delta t)^2)\right) \qquad (3)$$

Additional ontological constraints can reflect prior knowledge about the allowed relationship of detected objects.

As we have thus formalised a system involving variables on vehicle level $\phi$ as well as corresponding variables in the real world $\psi$, we can now relate systemic, real-world loss (e.g., in terms of available injury risk scales) to vehicle-level variables. As the vehicle variables include decision and actuator variables, such a loss function $l(\phi, \psi)$ evaluates the real-world severity of detecting, deciding, and acting. Note that both types of variables are collections of variables and in particular include references to different temporal instances.

### Risk assessment

As mentioned earlier, we are interested in the overall risk of the designed function $R$ as well as a situational risk $R^s$ from



Vehicle
Real world
temporal transition
temporal transition
Functional architecture (including safety mechanisms)
Sensors
Camera
RADAR
Preprocessing / AI
Features
Glare
Raw Object Detection
Bridge
Machine Learning
Objects
Fused Vehicle Detection
Fused Pedestrian Detection
Constraints from empirical evaluation (confidence intervals)
Driving function
Manoeuvre Decision
potentially additional ontological constraints
Glare
Bridge
dynamic constraints
Vehicle
Pedestrian
Risk assessment: worst case expected safety violations. Safety violations of decisions, when compared to the real world
Safety Goals

Figure 2: Structure of the probabilistic constraint system generated from the functional architecture and the constraints obtained via empirical evaluation as well as ontological and dynamic constraints. See text for more details.

which we can derive a robust low-risk manoeuvre in a given situation. Mathematically, these quantities can be described as the following expectations:

$$R = \mathbb{E}_{(\phi,\psi)}[l(\phi,\psi)],\ R^s = \mathbb{E}_{(\psi|\phi)}[l^s(\phi,\psi)] \qquad (4)$$

Note that for the situational risk, we use the conditional distribution conditioned on observations obtained in the particular situation and a potentially different loss-function $l^s$ (compared to the overall risk). More specifically, within the overall risk for the designed function, we might, e.g., want to use a binary loss function assigning $l(\phi,\psi) = 1$ if the situation was handled successfully and $l(\phi,\psi) = 0$ else. For the situational risk, we might want to use a quantitative assessment of the outcome. In contrast to the common setting of dynamic Bayesian Networks, the joint distribution $p_{\phi,\psi}$, however, is not completely given. Instead, only constraints over such a distribution are known due to equations like (2). More precisely, constraints as in (2) can be written as projections of the joint distribution using Bayes Rule:

$$P\left(\widehat{v}_i \mid \text{Glare } \wedge v_i \wedge \text{Bridge}\right) \qquad (5)$$
$$= \frac{P\left(\widehat{v}_i \wedge \text{Glare } \wedge v_i \wedge \text{Bridge}\right)}{P\left(\text{Glare } \wedge v_i \wedge \text{Bridge}\right)},$$

where each of the constraint variables either is a variable of the vehicle domain or of the real world (see Fig. 2). As the expression above omits some of the variables defined in those domains, the corresponding expressions have to be obtained by marginalising $p_{\phi,\psi}$. The question whether the (overall or situational) residual risk meets a desired bound $\vartheta$ can be formulated as a noisy optimisation problem

$$\max_{p_{\phi,\psi}\in\mathcal{P}} \mathbb{E}_{(\phi,\psi)}[l(\phi,\psi)] \overset{?}{\leq} \vartheta,\ \max_{p_{\phi,\psi}\in\mathcal{P}} \mathbb{E}_{(\psi|\phi)}[l^s(\phi,\psi)] \overset{?}{\leq} \vartheta\,, \qquad (6)$$

where the different constraints restrict the possible distributions, in the above formulation denoted by the set $\mathcal{P}$. If all variables are discrete, constraints on the distribution can directly be encoded into constraints on the distribution-values for different valuations of the vehicle or real-world variables. For continuous variables, the distribution has to be parametrised accordingly. Both types of constraints, however, can be incorporated into possibly non-linear functions $g_i$ acting on the parametrised version of the distribution and the variables $\phi,\psi$. For the empirical constraint of Eq. (2,5), such functions can be formalised as follows:

$$C_i(P,\phi,\psi)\ \text{def.:}\ g_i(P,\phi,\psi) \leq c_i \qquad (7)$$
$$\underbrace{\frac{\int p(\phi,\psi)d((\phi\cup\psi)\setminus\{\widehat{v}_i, v_i,\ \text{Glare, Bridge}\})}{\int p(\phi,\psi)d((\phi\cup\psi)\setminus\{v_i,\ \text{Glare, Bridge}\})}}_{:=g_0(P,\phi,\psi)} \leq \underbrace{\hat{p}+\epsilon(\delta)}_{:=c_0}$$

Using specification techniques of stochastic satisfiability modulo theory (Fränzle, Hermanns, and Teige 2008), the problem (6) can alternatively be formulated as:

$$\exists_{P:\bigwedge_i C_i(P,\phi,\psi)} \text{Я}_{\phi,\psi\sim P} : l(\phi,\psi) \overset{?}{\leq} \vartheta \qquad (8)$$

Here, we collected all constraints over the distribution as well as over the variables within the conjunction $\bigwedge_i C_i$. Exploiting importance sampling for Eq. 8 (Fränzle et al. 2015), such problem can be made amenable for analysis using available tools (Fränzle, Gao, and Gerwinn 2017). To address scalability issues, one can also resort to statistical model checking (Ellen, Gerwinn, and Fränzle 2014).

## Verification and situational analysis

Calculating the maximal risk as formalised in the previous section provides quantitative evidence to an overall safety verification process on vehicle level. Depending on the number of constraints with confidence statements, one can calculate an overall confidence level on the risk as well. Each confidence-based constraint holds with a certain confidence. If these can be regarded as independent, the overall confidence level is merely the product of the individual confidence levels. In case one is not willing to assume independence between the confidence-based constraints, the overall confidence level can be incorporated in a way similar to probabilistic constraints like (2). Note that such constraints also include constraints like c-approximate-independence as used in (Shalev-Shwartz, Shammah, and Shashua 2017), however we allow for even more pessimistic bounds whenever less information about the dependence is available.

The calculation of the maximal risk can also be performed in a particular situation. Instead of marginalising variables for the expected loss in (4), we can fix the valuation of vehicular variables to the observed values. The maximal risk then enables one to identify the most critical real-world situations and to choose a minimal risk manoeuvre. For our example, this facilitates inferring whether it is indeed more likely to falsely detect $v_2$ at time $t_1$ than having it not detected at time $t_0$. As due to the dynamic constraint, either $v_2$ has been missed at time $t_0$ and correctly classified at $t_1$ or the other way around, this restricts the joint distribution to assign zero probability to the other possibilities. Together with the empirical evidence constraints (e.g., marginal probabilities observing glare or the probability of bridges occurring), we can therefore calculate which of the two remaining possibilities are more likely. As such, it can be interpreted as the worst case interpretation of a Bayesian filter for dynamical systems which can be applied at each point in time. However, as worst-case configurations have to be identified, scalability of such an approach remains to be demonstrated in practice, but is outside of the scope of this short-paper.

## Discussion

We presented a framework designed for computing (a) the current risk under given observations and (b) the overall risk under the given constraints and marginal probabilities arising from empirical evaluations of different machine learning components involved within the functional architecture.

Within our setting, such quantities are different from inference tasks typically considered within Dynamic Bayesian Networks. The central issue is that probability distributions need not completely be known, but can be underspecified, as illustrated by the occurrence of glare or bridges provide constraints on the marginal. In fact, earlier approaches in combining constraints with Bayesian Belief Networks were frequently restricted to representing constraints as pseudo-observations (Crowley, Boerlage, and Poole 2007) or to interpreting the standard inference scheme as constraint propagation (Pearl 1985). But both can also be combined to render the inference machinery more suited for such kind of constrained network (Gogate and Dechter 2012).

Automatically learning the structure of Bayesian Networks has also been explored (Berg, Järvisalo, and Malone 2014). In such an approach, constraints about the parameters (or structure) of the underlying graph can be considered. As it fits the network parameters such that the network best explains a given dataset, that approach does not immediately fit into our robust safety verification setting.

In our work, unknown or underspecified relations between variables of the network are understood as spanning and constraining a set of possible distributions. From a frequentist point of view compatible with quantitative safety, we would like to compute worst and best case scenarios under all possible assignments across the viable probability distributions rather than missing information about the dependency of different variables. This paper explains the pragmatics and the underlying mathematical constructions; the development of scalable tools automating such reasoning as well as their benchmarking remain issues of future work.

## Acknowledgements

This research was partially funded by Deutsche Forschungsgemeinschaft within the Research Training Group 1765 “SCARE: System Correctness under Adverse Conditions”.

## References

Berg, J.; Järvisalo, M.; and Malone, B. 2014. Learning optimal bounded treewidth bayesian networks via maximum satisfiability. In *Artificial Intelligence and Statistics*, 86–95.

Cesa-Bianchi, N.; Conconi, A.; and Gentile, C. 2004. On the generalization ability of on-line learning algorithms. *IEEE Transactions on Information Theory* 50(9):2050–2057.

Crowley, M.; Boerlage, B.; and Poole, D. 2007. Adding local constraints to Bayesian networks. *Advances in AI* 344–355.

Ellen, C.; Gerwinn, S.; and Fränzle, M. 2014. Statistical model checking for stochastic hybrid systems involving nondeterminism over continuous domains. *International Journal on Software Tools for Technology Transfer*. Published online: 03 August 2014.

Fränzle, M.; Gerwinn, S.; Kröger, P.; Abate, A.; and Katoen, J.-P. 2015. Multi-objective parameter synthesis in probabilistic hybrid systems. In *International Conference on Formal Modeling and Analysis of Timed Systems*, 93–107. Springer.

Fränzle, M.; Gao, Y.; and Gerwinn, S. 2017. Constraint-solving techniques for the analysis of stochastic hybrid systems. In *Provably Correct Systems*. Springer. 9–38.

Fränzle, M.; Hermanns, H.; and Teige, T. 2008. Stochastic satisfiability modulo theory: A novel technique for the analysis of probabilistic hybrid systems. In *International Workshop on Hybrid Systems: Computation and Control*, 172–186. Springer.

Gogate, V., and Dechter, R. 2012. Approximate inference algorithms for hybrid Bayesian networks with discrete constraints. *arXiv:1207.1385*.

Koller, D., and Friedman, N. 2009. *Probabilistic Graphical Models: Principles and Techniques*. MIT press.

MacKenzie, E. J.; Shapiro, S.; and Eastham, J. N. 1985. The abbreviated injury scale and injury severity score: Levels of inter- and intrarater reliability. *Medical care* 823–835.

Murphy, K. P., and Russell, S. 2002. Dynamic bayesian networks: Representation, inference and learning.

Pearl, J. 1985. A constraint propagation approach to probabilistic reasoning. In *Proceedings of the First Conference on Uncertainty in Artificial Intelligence*.

SAE, O., et al. 2014. Taxonomy and definitions for terms telated to on-road motor vehicle automated driving systems. *SAE Standard J3016* 01–16.

Shalev-Shwartz, S.; Shammah, S.; and Shashua, A. 2017. On a formal model of safe and scalable self-driving cars. *arXiv preprint arXiv:1708.06374*.

# Safe Goal-Directed Autonomy and the Need for Sound Abstractions

**Siddharth Srivastava**
School of Computing, Informatics and Decision Systems Engineering
Arizona State University
Tempe, AZ 85282
siddharths@asu.edu

## Abstract

The field of sequential decision making (SDM) captures a range of mathematical frameworks geared towards the synthesis of goal-directed behaviors for autonomous systems. Abstract benchmark problems such as the *blocks-world domain* have facilitated immense progress in solution algorithms for SDM. there is some evidence that a direct application of SDM algorithms in real-world situations can produce unsafe behaviors. This is particularly apparent in task and motion planning in robotics. We believe that the reliability of today's SDM algorithms is limited because SDM models, such as the blocks-world domain, are *unsound* abstractions (those that yield false inferences) of real world situations.

This position paper presents the case for a focused research effort towards the study of sound abstractions of models for SDM and algorithms for efficiently computing safe goal-directed behavior using such abstractions.

## Introduction

The increasing maturity of AI techniques presents us with a unique opportunity to develop physical and electronic AI agents that could autonomously assist humans. Such agents would need to be able to accept high-level commands, and reason about what to do over extended periods of time spanning multiple decision epochs. The field of sequential decision making (SDM) captures such problems. In order to solve them, an AI agent needs to the assess different courses of action available to it: which course of actions would accomplish the assigned task? would it be safe to execute? which course of actions would be beneficial? Evaluating a possible courses of action in this way requires some knowledge about the environment and the possible impacts of the agent's actions in it—in other words, *a model*.

In the absence of a model, such evaluations would need to be done through trial and error. It is difficult to conceive of situations where deploying robots would have a high value and where such trials and their associated errors would be acceptable. In situations that involve proximal human-robot collaboration, or situations that are too dangerous for humans, errors are usually associated with forbidding penalties. Just as a bomb-disposal robot that learns on the fly would be an ephemeral investment, a household assistant that attempts to learn through trial an error, which medication is required when a person goes into insulin shock, would be of dubious ethical, social and economic value. It is well known in the AI community that PAC-learning guarantees alone are not sufficient for ensuring safe behavior in such situations; recent analyses have highlighted their limitations in the face of the anticipated roles of AI systems (Russell, Dewey, and Tegmark 2015; Brynjolfsson and Mitchell 2017).

The focus of this position paper is on the *mechanisms for creating domain models that are efficient but sound abstractions of real-world problems, and the algorithmic advances required for using such models for safe behavior synthesis*. Models can be in the form of closed-form mathematical specifications, (such as Markov Decision Process with transition probability specifications) or in the form of black-box simulators or generative models that can sample possible action outcomes (as typically used in reinforcement learning). Models of either form can be derived from existing knowledge, or learned through past experience in the field. Indeed, some of the most popular demonstrations of AI systems rely upon *perfect models* (Silver et al. 2017; Mnih et al. 2015) in the form of game simulators for efficiently obtaining millions of labeled behavioral experiences.

Regardless of the form or the nature of acquisition of models, higher fidelity models feature larger branching factors and larger time horizons and therefore result in SDM problems of higher computational complexity (regardless of the solution approach taken, be it dynamic programming, search, learning from trials and past experience, or a combination thereof). Hierarchical abstractions are used to alleviate this problem by creating input models that are abstractions of the true problem (Sacerdoti 1974; Knoblock 1990; Parr and Russell 1998; Dietterich 2000; Marthi, Russell, and Wolfe 2007).

Hierarchical abstractions include state abstractions (models that maintain fewer environment properties than the real situation) as well as temporal abstractions (models featuring high-level actions that span multiple primitive operations of the underlying actuators).

In recent work (Srivastava, Russell, and Pinto 2016) we showed that simple forms of abstractions can result in models that are not consistent with the underlying problem scenario as well as models that are not Markovian, or not solvable! As a result many real-world problems have never truly been addressed by SDM solution techniques that treat their input models as perfect abstractions.

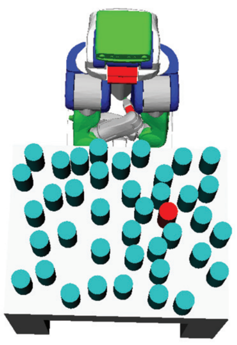


Figure 1: A realistic blocks-world problem. Pickups can be made only from the sides. Although there is no stacking and the preconditions of the pickup action are satisfied, there is no feasible motion plan for picking up most of the objects on the table.

For instance, consider the blocks-world domain, which is among the most easily recognizable, perhaps even infamous, benchmarks for sequential decision making. Given initial and desired configurations of blocks on a surface, the problem is to compute a behavior that would transform the initial configuration to the desired configuration. The set of available actions typically consists of maneuvers such as *pickup* and *place*. Stochastic action effects and noisy sensors for this domain can be easily expressed in most modeling languages for SDM (Boutilier, Reiter, and Price 2001; Younes and Littman 2004; Sanner 2010; Srivastava, Cheng, and Russell 2014). Although SDM *models* for the blocks-world domain are considered to be too "well studied" to be interesting for research, they are poor abstractions of the underlying SDM *problems* of rearranging objects while avoiding collisions (see Fig. 1 for a simplified yet realistic problem). Consequently the underlying problems remain unsolved and feature significant research challenges.

Indeed, while the true space of blocks-world problems captures all pick-and-place problems, ongoing research in robotics shows that SDM solvers that perform well on the standard blocks-world model produce poor *unsafe* solutions even in simplified real-world situations that feature robots with perfect sensing and actuation, and block arrangements without stacking (Cambon, Alami, and Gravot 2009; Kaelbling and Lozano-Pérez 2011; Plaku and Hager 2010; Kaelbling and Lozano-Pérez 2013; Srivastava et al. 2014). These solutions could violate arbitrarily many of the constraints that were abstracted in an unsound fashion, and result in unsafe behaviors that include unintended collisions. This situation is representative of several problems where goal-directed autonomy is desired; we believe that this potential for unsafe behavior effectively prohibits the safe deployment of general purpose AI agents.

## Problem with the Current Situation

**Conventional modeling paradigm** As noted above, it is well appreciated that abstraction is a useful mathematical tool for solving real-world SDM problems. The conventional wisdom along these lines is to use an abstract domain model with an SDM solver to compute the "high-level strategy" for solving a problem (e.g., one that determines the order of unstacking and stacking block-tower configurations), and then use a low-level planner (e.g., a motion planner) or controller to implement each of the actions in the strategy.

**Underlying assumptions and their limitations** This wisdom is based on the assumption that the effect of applying an action in the real world will be consistent with the modeled effect in the abstract model. This in turn is based on the assumption that the result of applying a desired abstraction function on the real situation will be a Markovian model.

On the other hand, constructing a Markovian model requires the inclusion of several properties of the environment as state variables or predicates; abstraction requires *removing, or coarsening properties* in the model. It should therefore be natural to expect the abstraction of an accurate domain model to possibly result in a *non-Markovian domain model*. Recent work shows that this intuition is in fact true (Srivastava, Russell, and Pinto 2016): simple abstractions can result in models that are not Markovian; furthermore, it is often not possible to express the resulting models accurately in existing modeling languages for SDM.

This raises a few questions: all the SDM models we use are Markovian (and naturally, are expressed in the modeling languages that we have been using). Few, if any, of these are accurate, non-abstracted depictions of the real world situation that they represent. Have we been lucky enough to always get Markovian abstractions? Do the domain designers intuitively construct perfect abstract transition systems that retain just the right properties to make the resulting abstracted model tractable as well as Markovian?

To answer these questions, we turn once again to the blocks world and its abstraction expressed as the blocks-world domain. Among other details, this domain states that if a block has nothing on top of it, the robot's gripper should be able to pick it up. In a real situation (e.g. Fig. 1), this is *not true* because there may be no collision-free path for the gripper to pick up the block. The vocabulary used in the blocks-world domain is not sufficient to accurately express this property (Cambon, Alami, and Gravot 2009; Hertle et al. 2012; Kaelbling and Lozano-Pérez 2011). *As a result the standard blocks-world domain is not a sound model of the real blocks world because it implies action consequences that are not true* [1]. Policies computed using such models are unsafe, and can be dangerous. Although our example refers to situations where geometric constraints were abstracted out, such errors can arise with all forms of abstraction. One would not appreciate a robot using such principles in most applications that could benefit from a safe and productive robot, including mining, firefighting, bomb disposals, household help, etc.

[1]It is sound for environments where the gripper is either infinitesimally thin (so that it can slip between adjacent towers), or is an electromagnet suspended from the ceiling. Either way, the ceiling should be arbitrarily high and the table should be broad enough to lay any number of blocks on it. Such situations are unusual if not impossible.

In fact, the sound abstraction of the blocks world using the vocabulary of the blocks-world domain is a non-Markovian transition system: the effect of reaching for a block in this transition system depends on the occurrence of preceding *place* actions. If the target block was initially reachable, and no other *place* actions placed a block on the same table, the block will remain accessible. Otherwise, it may not be. *Therefore, the standard blocks-world model is not only inconsistent with the underlying problem, its vocabulary is insufficient to make the abstract transition system Markovian!* Forcing such a non-Markovian abstract transition system into domain languages that can only express Markovian models results in a model that is inconsistent with the modeled problem. Our research indicates that the situation can be resolved if the modeling languages are extended to annotate parts of the model as imprecise due to abstraction, and algorithms utilize this information to extract more information from higher fidelity models when needed.

**Non-solutions** The preceding discussion may *seem to indicate* that a stochastic formulation (such as an MDP) would help resolve these issues. However, this is not true. First, it would require enumerating and solving for the complete set of *possible* outcomes for an action in an abstract state space. This is infeasible. E.g., in the blocks-world model's vocabulary, every time a robot (not a ceiling mounted gripper) tries to move its hand, all possible subsets of movable objects in the room would need to be considered as potentially being knocked over. Second, such models would not be *complete*: they would disallow solutions that are feasible under a more accurate representation.

The problems highlighted above are orthogonal to efforts aimed at increasing the level of detail expressible in our input modeling languages (e.g., (Hertle et al. 2012; Fox and Long 2002)). Even if we could model SDM problems at the level of detail of sub-atomic particle interactions, this is unlikely to yield more efficient solution techniques. It is equally unlikely that modeling an entire household at the level circuit diagrams of every appliance would "help" a household robot efficiently compute useful behavior. Natural computational consequences of increasing branching factors and time horizons make it clear that a uniformly detailed model at the highest possible fidelity will not yield the most efficient SDM system, regardless of the solution approach. Thus, SDM solvers will continue to rely upon hierarchical abstractions for efficiency in modeling and in solution computation.

## Paths Ahead

We believe that the limitations in correctly expressing abstract SDM models of real-world situations (and consequently, of efficiently solving such problems) have limited the applicability of SDM techniques in the real world. As a community we have made numerous advances under the assumption that inputs will be perfect abstractions that yield exactly the true consequences. Our position is that these advances are necessary, but not sufficient towards deployable autonomous systems. We also need to expand the scope of SDM technology towards principled approaches for designing and computing abstract SDM models that may be imprecise, but not incorrect. New representations for such abstract SDM models (generative models or *simulators*, as well as analytical) would require and facilitate corresponding algorithms that produce truly executable solutions.

Some prior research efforts are highly relevant to this problem. Work on algorithms for planning with models that may be incomplete addresses situations when unknown perturbations may have been applied to accurate domain models that are expressible in the modeling language (Nguyen and Kambhampati 2014). Angelic semantics for high-level actions increase the scope of representation languages to specify upper and lower bounds on reachable states in situations with temporal abstraction rather than state abstraction. The resulting algorithms are able to effectively utilize such bounds in pruning irrelevant high-level actions (Marthi, Russell, and Wolfe 2007). Related research in motion planning highlights the value of state abstractions of control-theoretic models, which are constructed using subsets of the full set of variables required to describe a system (Styler and Simmons 2017). We have been developing representations for efficiently expressing imprecise but sound abstract models resulting from state and temporal abstraction for arbitrary SDM problems. Our solution algorithms utilize sound and imprecise abstract models, but dynamically improve them by deriving abstracted, context-sensitive information from more accurate models. This information is abstracted and incorporated in the abstract models (Srivastava et al. 2014; Srivastava, Russell, and Pinto 2016), allowing SDM algorithms to compute agent behaviors with strong guarantees of safety and correctness. Some of our main results can be summarized as follows:

1. Under certain conditions, abstraction can indeed result in Markovian models. These conditions appear to be rare.
2. In many cases, abstraction results in domain models that includes forms of model-imprecision that could have been resolved during computation had they been expressed. However, current modeling languages do not support constructs that distinguish model imprecision arising due to abstraction from non-determinism or stochasticity that is a feature of the environment.
3. If model imprecision caused due to abstraction is recorded in the abstract model (e.g., by noting that the effect of a *place* action is imprecise, along with the abstraction function that caused the imprecision), the situation can be resolved. It is possible to dynamically tune the abstraction to include more information from accurate models using different solvers for models at different levels of abstraction. Used in this fashion, SDM solvers can effectively produce executable behavior. Dynamically tuning an imprecise (but not incorrect) model during search allowed us to produce a competitive task and motion planner that uses existing SDM solvers.

These initial results indicate that new methods for computing and utilizing abstract models that are sound even when they are imprecise allow us to leverage SDM technology towards solving entire new classes of problems that are abstractions of real-world situations.

## Acknowledgments

I thank Rao Kambhampati for helpful discussions and comments on this paper.

## References

Boutilier, C.; Reiter, R.; and Price, B. 2001. Symbolic dynamic programming for first-order mdps. In *Proc. IJCAI*, volume 1, 690–700.

Brynjolfsson, E., and Mitchell, T. 2017. What can machine learning do? workforce implications. *Science* 358(6370):1530–1534.

Cambon, S.; Alami, R.; and Gravot, F. 2009. A hybrid approach to intricate motion, manipulation and task planning. *IJRR* 28:104–126.

Dietterich, T. G. 2000. Hierarchical reinforcement learning with the maxq value function decomposition. *J. Artif. Intell. Res.(JAIR)* 13:227–303.

Fox, M., and Long, D. 2002. PDDL+: Modeling continuous time dependent effects. In *Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space*.

Hertle, A.; Dornhege, C.; Keller, T.; and Nebel, B. 2012. Planning with semantic attachments: An object-oriented view. In *Proc. ECAI*.

Kaelbling, L. P., and Lozano-Pérez, T. 2011. Hierarchical task and motion planning in the now. In *Proc. ICRA*.

Kaelbling, L. P., and Lozano-Pérez, T. 2013. Integrated task and motion planning in belief space. *The International Journal of Robotics Research* 32(9-10):1194–1227.

Knoblock, C. A. 1990. Learning abstraction hierarchies for problem solving. In *Proc. AAAI*.

Marthi, B.; Russell, S. J.; and Wolfe, J. 2007. Angelic semantics for high-level actions. In *Proc. ICAPS*.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529–533.

Nguyen, T. A., and Kambhampati, S. 2014. A heuristic approach to planning with incomplete STRIPS action models. In *ICAPS*.

Parr, R., and Russell, S. J. 1998. Reinforcement learning with hierarchies of machines. In *Proc. NIPS*.

Plaku, E., and Hager, G. D. 2010. Sampling-based motion and symbolic action planning with geometric and differential constraints. In *Proc. ICRA*.

Russell, S.; Dewey, D.; and Tegmark, M. 2015. Research priorities for robust and beneficial artificial intelligence. *AI Magazine* 36(4):105–114.

Sacerdoti, E. D. 1974. Planning in a hierarchy of abstraction spaces. *Artificial intelligence* 5(2):115–135.

Sanner, S. 2010. Relational dynamic influence diagram language (rddl): Language description. http://users.cecs.anu.edu.au/~ssanner/IPPC_2011/RDDL.pdf.

Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; Chen, Y.; Lillicrap, T.; Hui, F.; Sifre, L.; van den Driessche, G.; Graepel, T.; and Hassabis, D. 2017. Mastering the game of go without human knowledge. *Nature* 550(7676):354–359.

Srivastava, S.; Fang, E.; Riano, L.; Chitnis, R.; Russell, S.; and Abbeel, P. 2014. A modular approach to task and motion planning with an extensible planner-independent interface layer. In *Proc. ICRA*.

Srivastava, S.; Cheng, X.; and Russell, S. 2014. First-order open-universe POMDPs: Formulation and algorithms. In *Proc. UAI*.

Srivastava, S.; Russell, S.; and Pinto, A. 2016. Metaphysics of planning domain descriptions. In *Proc. AAAI*.

Styler, B. K., and Simmons, R. 2017. Plan-time multi-model switching for motion planning. In *Proc. ICAPS*.

Younes, H. L., and Littman, M. L. 2004. PPDDL 1.0: An extension to pddl for expressing planning domains with probabilistic effects. *Technical Report CMU-CS-04-162*.

# Learning Generalized Reactive Policies Using Deep Neural Networks

**Edward Groshev,**[†] **Aviv Tamar,**[†] **Maxwell Goldstein,**[‡]
**Siddharth Srivastava,**[*§] **Pieter Abbeel**[†]
[†] Department of Computer Science, University of California, Berkeley CA 94720
[‡] Department of Computer Science, Princeton University, Princeton, NJ 08544
[§] School of Computing, Informatics and Decision Systems Engineering, Arizona State University, Tempe, AZ 85281

**Abstract**

We present a new approach to learning for planning, where knowledge acquired while solving a given set of planning problems is used to plan faster in related, but new problem instances. We show that a deep neural network can be used to learn and represent a *generalized reactive policy* (GRP) that maps a problem instance and a state to an action, and that the learned GRPs efficiently solve large classes of challenging problem instances. In contrast to prior efforts in this direction, our approach significantly reduces the dependence of learning on handcrafted domain knowledge or feature selection. Instead, the GRP is trained from scratch using a set of successful execution traces. We show that our approach can also be used to automatically learn a heuristic function that can be used in directed search algorithms. We evaluate our approach using an extensive suite of experiments on two challenging planning problem domains and show that our approach facilitates learning complex decision making policies and powerful heuristic functions with minimal human input. Video results available at goo.gl/Hpy4e3.

## Introduction

In order to help with day to day chores such as organizing a cabinet or arranging a dinner table, robots need to be able plan: to reason about the best course of action that could lead to a given objective. Unfortunately, planning is well known to be a challenging computational problem: plan-existence for deterministic, fully observable environments is PSPACE-complete when expressed using rudimentary propositional representations (Bylander 1994). Such results have inspired multiple approaches for reusing knowledge acquired while planning across multiple problem instances (in the form of triangle tables (Fikes, Hart, and Nilsson 1972), learning control knowledge for planning (Yoon, Fern, and Givan 2008), and constructing generalized plans that solve multiple problem instances (Srivastava, Immerman, and Zilberstein 2011; Hu and De Giacomo 2011) with the goal of faster plan computation on a new problem instance.

In this work, we present an approach that unifies the principles of imitation learning (IL) and generalized planning for learning a *generalized reactive policy* (GRP) that predicts the action to be taken, given an observation of the planning problem instance and the current state. The GRP is represented as a deep neural network (DNN). We use an off-the-shelf planner to plan on a set of training problems, and train the DNN to learn a GRP that imitates and generalizes the behavior generated by the planner. We then evaluate the learned GRP on a set of unseen test problems from the same domain. We show that the learned GRP successfully generalizes to unseen problem instances including those with larger state spaces than were available in the training set. This allows our approach to be used in end-to-end systems that learn representations as well as executable behavior purely from observations of successful executions in similar problems.

We also show that our approach can generate representation-independent heuristic functions for a given domain, to be used in arbitrary directed search algorithms such as A$^*$ (Hart, Nilsson, and Raphael 1968). Our approach can be used in this fashion when stronger guarantees of completeness and classical notions of “explainability” are desired. Furthermore, in a process that we call “leapfrogging", such heuristic functions can be used in tandem with directed search algorithms to generate training data for much larger problem instances, which in turn can be used for training more general GRPs. This process can be repeated, leading to GRPs that solve larger and more difficult problem instances with iteration.

While recent work on DNNs has illustrated their utility as function representations in situations where the input data can be expressed in an image-based representation, we show that DNNs can also be effective for learning and representing GRPs in a broader class of problems where the input is expressed using a graph data structure. For the purpose of this paper, we restrict our attention to deterministic, fully observable planning problems. We evaluate our approach on two planning domains that feature different forms of input representations. The first domain is Sokoban (see Figure 1). This domain represents problems where the execution of a plan can be accurately expressed as a sequence of images. This category captures a number of problems of interest in household robotics including setting the dinner table. This problem has been described as the most challenging problem

[*]Part of the work was done while this author was at United Technologies Research Center

in the literature on learning for planning (Fern, Khardon, and Tadepalli 2011).

Our second test domain is the traveling salesperson problem (TSP), which represents a category of problems where execution is *not* efficiently describable through a sequence of images. This problem is challenging for classical planners as valid solutions need to satisfy a plan-wide property (namely a Hamiltonian cycle, which does not revisit any nodes). Our experiments with the TSP show that using graph convolutions (Dai et al. 2017) DNNs can be used effectively as function representations for GRPs in problems where the grounded planning domain is expressed as a graph data structure.

Our experiments reveal that several architectural components are required to learn GRPs in the form of DNNs: (1) A *deep* network. (2) Structuring the network to receive as input pairs of current state and goal observations. This allows us to 'bootstrap' the data, by training with *all pairs* of states in a demonstration trajectory. (3) Predicting plan length as an auxiliary training signal can improve IL performance. In addition, the plan length can be effectively exploited as a heuristic by standard planners.

We believe that these observations are general, and will hold for many domains. For the particular case of Sokoban, using these insights, we were able to demonstrate a 97% success rate in one object domains, and an 87% success rate in two object domains. In Figure 1 we show an example test domain, and a non-trivial solution produced by our learned DNN.

## Related Work

The interface of planning and learning (Fern, Khardon, and Tadepalli 2011) has been investigated extensively in the past. The works of Khadron (Khardon 1999), Martin and Geffner (Martín and Geffner 2004), and Yoon et al. (Yoon, Fern, and Givan 2002) learn policies represented as decision lists on the logical problem representation, which needs to be hand specified. On the other hand, the literature on generalized planning (Srivastava, Immerman, and Zilberstein 2011; Hu and De Giacomo 2011) has focused on computing iterative generalized plans that solve broad classes of problem instances, with strong formal guarantees of correctness. While all of these strive to reuse knowledge made available during planning, the selection of a good *representation* for expressing the data as well as the learned functions or generalized plans is handcrafted. Feature sets and domain descriptions in these approaches are specified by experts using formal languages such as PDDL (Fox and Long 2003). Similarly, approaches such as case-based planning (Spalzzi 2001), approaches for extracting macro actions (Fikes, Hart, and Nilsson 1972; Scala, Torasso, and others 2015) and for explanation based plan generalization (Shavlik 1989; Kambhampati and Kedar 1994) rely on curated vocabularies and domain knowledge for representing the appropriate concepts necessary for efficient generalization of observations and the instantiation of learned knowledge. Our approach requires as input only a set of successful plans and their executions—our neural network architecture is able to learn a reactive policy that predicts the best action to execute based on the current state of the environment without any additional representational expressions. The current state is expressed either as an image (Sokoban) or as an instance of the graph data structure (TSP).

Neural networks have previously been used for learning heuristic functions (Ernandes and Gori 2004). Recently, deep convolutional neural networks (DNNs) have been used to automatically extract expressive features from data, leading to state-of-the-art learning results in image classification (Krizhevsky, Sutskever, and Hinton 2012), natural language processing (Sutskever, Vinyals, and Le 2014), and control (Mnih et al. 2015), among other domains. The phenomenal success of DNNs for across various disciplines motivates us to investigate whether DNNs can learn useful representations in the learning for planning setting as well. Indeed, one of the contributions of our work is a general convolutional DNN architecture that is suitable for learning to plan.

Imitation learning has been previously used with DNNs to learn policies for tasks that involve short horizon reasoning such as path following and obstacle avoidance (Pomerleau 1989; Ross, Gordon, and Bagnell 2011; Tamar et al. 2016; Pfeiffer et al. 2016), focused robot skills (Mülling et al. 2013; Nair et al. 2017), and recently block stacking (Duan et al. 2017). From a planning perspective, the Sokoban domain considered here is considerably more challenging than block stacking or navigation between obstacles. In (Tamar et al. 2016), a value iteration planning computation was embedded within the network structure, and demonstrated successful learning on 2D gridworld navigation. Due to the curse of dimensionality, it is not clear how to extend that work to planning domains with much larger state spaces, such as the Sokoban domain considered here. In that work the state space was a 2D grid world with local connectivity, making value iteration tractable. However, for Sokoban, the state must include the position of both the agent and the objects, making it much larger than a 2D grid world. While one can construct such a state space, running value iteration on it would be too slow. Another alternative is to try to embed the Sokoban problem in some 2D grid world and run VI on it. This method performs significantly worse than our proposed method. Concurrently with our work, Weber et al. (Weber et al. 2017) proposed a DNN architecture that combines model based planning with model free components for reinforcement learning, and demonstrated results on the Sokoban domain. In comparison, our IL approach requires significantly less training instances of the planning problem (over 3 orders of magnitude) to achieve similar performance in Sokoban.

The 'one-shot' techniques in (Duan et al. 2017), however, are complimentary to this work. The impressive Alpha-Go-Zero (Silver et al. 2017) program learned a DNN policy for Go using reinforcement learning and self play. Key to its success is the natural curriculum in self play, which allows reinforcement learning to gradually explore more complicated strategies. A similar self-play strategy was essential for Tesauro's earlier Backgammon agent (Tesauro 1995). For the goal-directed planning problems we consider here, it is not clear how to develop such a curriculum strategy, although our leapfrogging idea takes a step in that direction. Extending

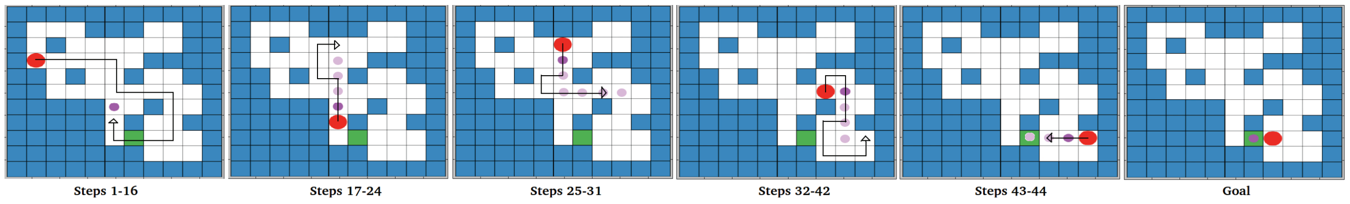


Figure 1: The Sokoban domain (best viewed in color). In Sokoban the agent (red dot) needs to push around movable objects (purple dots) between unmovable obstacles (blue squares) to a goal position (green square). In this figure we show a challenging Sokoban instance with one object. From left to right, we plot several steps in the shortest plan for this task: arrows represent the agent's path, and light purple dots show the resulting object movement. This 44 step trajectory was produced by our learned DNN policy. Note that it demonstrates reasoning about dead ends that may happen many steps after the initial state.

our work to reinforcement learning is a direction for future research.

Our approach thus offers two major advantages over prior efforts: (1) in situations where successful plan executions can be observed, e.g. by observing humans solving problems, our approach significantly reduces the effort required in designing domain representations; (2) in situations where guarantees of success are required, and domain representations are available, our approach provides an avenue for automatically generating a representation-independent heuristic function, which can be used with arbitrary guided search algorithms.

## Formal Framework

We assume the reader is familiar with the formalization of deterministic, fully observable planning domains and planning problems in languages such as PDDL (Fox and Long 2003; Helmert 2009) and present the most relevant concepts here. A planning problem domain can be defined as a tuple $K = \langle \mathcal{R}, \mathcal{A} \rangle$, where $\mathcal{R}$ is a set of binary *relations*; and $\mathcal{A}$ is a set of *parameterized actions*. Each action in $\mathcal{A}$ is defined by a set of preconditions categorizing the states on which it can be applied, and the set of instantiated relations that will changed to true or false as a result of executing that action. A planning problem instance associated with a planning domain can be defined as $\Pi = \langle \mathcal{E}, s_0, G \rangle$, where $\mathcal{E}$ is a set of entities, $s_0$ is an initial state and $G$ is a set of goal conditions. Relations in $\mathcal{R}$ instantiated with entities from $\mathcal{E}$ define the set of *grounded fluents*, $\mathcal{F}$. Similarly, actions in $\mathcal{A}$ instantiated with appropriately entities in $\mathcal{E}$ define the set of *grounded actions*, denoted as $\mathcal{A}[\mathcal{E}]$. The initial state, $s_0$, for a given planning problem is a complete truth valuation of fluents in $\mathcal{F}$; the goal condition, $G$, is a truth valuation of a subset of the grounded fluents in $\mathcal{F}$.

As an example, the discrete move action could be represented as follows:

$$\mathit{Move}(\text{loc1}, \text{loc2}) : \begin{cases} \mathit{pre} : \mathit{RobotAt}(loc1), \\ \mathit{eff} : \neg \mathit{RobotAt}(\text{loc1}), \mathit{RobotAt}(\text{loc2}). \end{cases}$$

We introduce several additional notations to the planning problem, to make the connection with imitation learning clearer. Given a planning domain and a planning problem instance, we denote by $S = 2^{\mathcal{F}}$ the state space of the planning problem. A state $s \in S$ corresponds to the values of each fluent in $\mathcal{F}$. The task in planning is to find a sequence of grounded actions, $a_0, \ldots, a_n$ – the so called *plan* – such that $a_n(\ldots(a_0(s_0))\ldots) \models G$.

In Sokoban, the domain represents the legal movement actions and the notion of movement on a bounded grid, a problem instance represents the exact grid layout (denoting which cell-entities are blocked), the starting locations of the objects and the agent, and the goal locations of the objects.

We denote by $o(\Pi, s)$ the *observation* for a problem instance $\Pi$ when the state is $s$. For example, $o$ can be an image of the current game state (Figure 1) for Sokoban. We let $\tau = \{s_0, o_0, a_0, s_1, \ldots, s_g, o_g\}$ denote the state-observation-action trajectory implied by the plan. The plan length is the number of states in $\tau$.

Our objective is to learn a generalized behavior representation that efficiently solves multiple problem instances for a domain. More precisely, given a domain $K$, and a problem instance $\Pi$, let $\mathcal{O}_{K,\Pi}$ be the set of possible observations of states from $\Pi$. Given a planning problem domain $K = \langle \mathcal{R}, \mathcal{A} \rangle$ we define a *generalized reactive policy (GRP)* as a function mapping observations of problem instances and states to actions: $GRP_K : \cup_\Pi \{\mathcal{O}_{K,\Pi}\} \rightarrow \cup_\Pi \{\mathcal{A}[\mathcal{E}_\Pi]\}$, where $\mathcal{E}_\Pi$ is the set of entities defined by the problem $\Pi$ and the unions range over all possible problem instances associated with $K$. Further, $GRP_K$ is constrained so that the observations from every problem instance are mapped to the grounded actions for that problem instance ($\forall \Pi \quad GRP_K(\mathcal{O}_{K,\Pi}) \subseteq \mathcal{A}[\mathcal{E}_\Pi]$). This effectively generalizes the concept of a policy to functions that can map states from multiple problem instances of a domain to action spaces that are legal within those instances.

**Imitation Learning** In imitation learning (IL), demonstrations of an expert solving a problem are given in the form of observation-action trajectories $D_{\text{imitation}} = \{o_0, a_0, o_1, \ldots, o_T, a_T\}$. The goal is to find a policy – a mapping from observation to actions $a = \mu(o)$, which imitates the expert. A straightforward IL approach is *behavioral cloning* (Pomerleau 1989), in which supervised learning is used to learn $\mu$ from the data.

## Learning Generalized Reactive Policies

We assume we are given a set $D_{\text{train}}$ of $N_{\text{train}}$ problem instances $\{\Pi_1, \ldots, \Pi_{N_{\text{train}}}\}$, which will be used for learning a GRP, and a set $D_{\text{test}}$ of $N_{\text{test}}$ problem instances that will be used for evaluating the learned model. We also assume that the training and test problem instances are similar in some sense, so that relevant knowledge can be extracted from the training set to improve performance on the test set. Concretely, both training and test instances come from the same distribution.

Our approach consists of two stages: a data generation stage and a policy training stage.

**Data generation** We generate a random set of problem instances $D_{\text{train}}$. For each $\Pi \in D_{\text{train}}$, we run an off-the-shelf planner to generate a plan and corresponding trajectory $\tau$, and then add the observations and actions in $\tau$ to $D_{\text{imitation}}$. In our experiments we used the Fast-Forward (FF) planner (Jörg Hoffman 2001), though any other PDDL planner can be used instead.

**Policy training** Given the generated data $D_{\text{imitation}}$, we use IL to learn a GRP $\mu$. The learned policy $\mu$ maps an observation to action, and therefore can be readily deployed to any test problem in $D_{\text{test}}$.

One may wonder why such a naive approach would even learn to produce the complex decision making ability that is required to solve unseen instances in $D_{\text{test}}$. Indeed, as we show in our experiments, naive behavioral cloning with standard shallow neural networks fails on this task. One of the contributions of this work is the investigation of DNN representations that make this simple approach succeed.

## Data Bootstrapping

In the IL literature (e.g., (Pomerleau 1989; Ross, Gordon, and Bagnell 2011)), the policy is typically structured as a mapping from the observation of a state to an action. However, GRPs need to consider the problem instance while generating an action to be executed since different problem instances may have different goals. Although this seems to require more data, we present an approach for "data bootstrapping" that mitigates the data requirements.

Recall that our training data $D_{\text{imitation}}$ consists of $N_{\text{train}}$ trajectories composed of observation-action pairs. This means that the number of training samples for a policy mapping state-observations to actions is equal to the number of observation-action pairs in the training data. However, since GRPs use the goal condition in their inputs (captured by a problem instance),*any pair* of observations from successive states $(o(\Pi, s_i), o(\Pi, s_j))$ and the intermediate trajectory in an execution in $D_{\text{train}}$ can be used as a sample for training the policy by setting $s_j$ as a goal condition for the intermediate trajectory. Our reasoning for this data bootstrapping technique is based on the following fact:

**Proposition 1.** *For a planning problem $\Pi$ with initial state $s_0$ and goal state $s_g$, let $\tau_{opt} = \{s_0, s_1, \ldots, s_g\}$ denote the shortest plan from $s_0$ to $s_g$. Let $\mu_{opt}(s)$ denote an optimal policy for $\Pi$ in the sense that executing it from $s_0$ generates the shortest path $\tau_{opt}$ to $s_g$. Then, $\mu_{opt}$ is also optimal for a problem $\Pi$ with the initial and goal states replaced with any two states $s_i, s_j \in \tau_{opt}$ such that $i < j$.*

Proposition 1 underlies classical planning methods such as triangle tables (Fikes, Hart, and Nilsson 1972). Here, we exploit it to design our DNN to take as input *both* the *current observation* and a *goal observation*. For a given trajectory of length $T$, the bootstrap can potentially increase the number of training samples from $T$ to $(T-1)^2/2$. In practice, for each trajectory $\tau \in D_{\text{imitation}}$, we uniformly sample $n_{\text{bootstrap}}$ pairs of observations from $\tau$. In each pair, the first observation is treated as the current observation, while the last observation is treated as the goal observation[1]. This results in $n_{\text{bootstrap}}+T$ training samples for each trajectory $\tau$, which are added to a bootstrap training set $D_{\text{bootstrap}}$ to be used instead of $D_{\text{imitation}}$ for training the policy. [2]

## Network Structure

We propose a general structure for a convolutional network that can learn a GRP.

Our network is depicted in Figure 2. The current state and goal state observations are passed through several layers of convolution which are shared between the action prediction network and the plan length prediction network. There are also skip connections from the input layer to to every convolution layer.

The shared representation is motivated by the fact that both the actions and the overall plan length are integral parts of a plan. Having knowledge of the actions makes it easy to determine plan length and vice versa, knowledge about the plan length can act as a template for determining the actions. The skip connections are motivated by the fact that several planning algorithms can be seen as applying a repeated computation, based on the planning domain, to a latent variable. For example, greedy search expands the current node based on the possible next states, which are encoded in the domain; value iteration is a repeated modification of the value given the reward and state transitions, which are also encoded in the domain. Since the network receives no other knowledge about the domain, other than what's present in the observation, we hypothesize that feeding the observation to every conv-net layer can facilitate the learning of similar planning computations. We note that in value iteration networks (Tamar et al. 2016), similar skip connections were used in an explicit neural network implementation of value iteration.

For planning in graph domains, we propose to use graph convolutions, similar to the work of (Dai et al. 2017). The graph convolution can be seen as a generalization of an image convolution, where an image is simply a grid graph. Each node in the graph is represented by a feature vector, and linear operations are performed between a node and its neighbors, followed by a nonlinear activation. A detailed description is provided in the supplementary material. For the TSP problem with $n$ nodes, we map a partial Hamiltonian path $P$ of the graph to a feature representation as follows. For each node, the features are represented as a 3-dimensional binary vector.

[1]In our experiments, we used the FF planner, which does not necessarily produce shortest plans. However, Proposition 1 can be extended to satisficing plans.

[2]Note that for the Sokoban domain, goal observations in the test set (i.e., real goals) do not contain the robot position, while the goal observations in the bootstrap training set include the robot position. However, this inconsistency had no effect in practice, which we verified by explicitly removing the robot from the observation.

The first element is 1 if the node has been visited in $P$, the second element is 1 if it is the current location of the agent, and the third element is 1 if the node is the terminal node. For a Hamiltonian cycle the terminal node is the start node. The state is then represented as a collection of feature vectors, one for each node. In the TSP every Hamiltonian cycle is of length $n$, so predicting the plan length in this case is trivial, as we encode the number of visited cities in the feature matrix. Therefore, we omit the plan-length prediction part of the network.

### Generalization to Different Problem Sizes

A primary challenge in learning for planning is finding representations that can generalize across different problem sizes. For example, we expect that a good policy for Sokoban should work well on the instances it was trained on, $9 \times 9$ domains for example, as well as on larger instances, such as $12 \times 12$ domains. A convolution-based architecture naturally addresses this challenge.

However, while the convolution layers can be applied to any image/graph size, the number of inputs to the fully connected layer is strictly tied to the problem size. This means that the network architecture described above is fixed to a particular grid dimension. To remove this dependency, we employ a trick used in fully convolutional networks (Long, Shelhamer, and Darrell 2015), and keep only a $k \times k$ window of the last convolution layer, centered around the current agent position. This modification makes our DNN applicable to any grid dimension. Note that since the window is applied *after* the convolution layers, the receptive field can be much larger than $k \times k$. In particular, a value of $k = 1$ worked well in our experiments. For the graph architectures, a similar trick is applied, where the decision at a particular node is a function of the convolution result of its neighbors, and the same convolution weights are used across different graph sizes.

## Experiments

Here we report our experiments on learning for planning with DNNs. Our focus is on the following questions:

1. What makes a good DNN architecture for learning a GRP?
2. Can a useful planning heuristic be extracted from the GRP?

The first question aims to show that recent developments in the representation learning community, such as deep convolutional architectures, can be beneficial for planning. The second question has immediate practical value – a good heuristic can decrease planning costs. However, it also investigates a deeper premise. If a useful heuristic can indeed be extracted from the GRP, it means that the GRP has learned some underlying structure in the problem. In the domains we consider, such structure is hard to encode manually, suggesting that the data-driven DNN approach can be promising.

To investigate these questions, we selected two test domains representative of very different classes of planning problems. We used the *Sokoban* domain to represent problems where plan execution can be captured as a set of images, and the goal takes the form of achieving a state property (objects at their target locations). We used the *traveling salesperson problem* as an exemplar for problems where plan execution is not easy to capture as a set of images and the goal features a temporal property.

**Sokoban** For Sokoban, we consider two difficulty levels: moving a single object as described in Figure 1, and a harder task of moving two objects. We generated training data using a Sokoban random level generator[3].

For imitation learning, we represent the policy with the DNNs as described in Network Structure section and optimize using Adam (Kingma and Ba 2014) (step size 0.001). When training with data bootstrapping, we selected $n_{\text{bootstrap}} = T$ for generating $D_{\text{bootstrap}}$. Unless stated otherwise, the training set used in all Sokoban experiments was comprised of 45k observation-action trajectories from 9k different obstacle configurations.

To evaluate policy performance on the Sokoban domain we use execution success rate. Starting from the initial state, we execute the learned policy deterministically and track whether or not the goal state is reached. We evaluate performance both on test domains of the same size the GRPs were trained on, $9 \times 9$ grids, and also on larger problems. We explicitly verified that *none of the test domains appeared in the training set*.

Videos of executions of our learned GRPs for Sokoban are available at goo.gl/Hpy4e3.

**TSP** For TSP, we consider two different graph distributions. The first is the space of complete graphs with edge weights sampled uniformly in $[0, 1]$. The second, which we term *chord graphs*, is generated by first creating an $n$-node graph in the form of a cycle, and then adding $2n$ undirected chords between randomly chosen pairs of nodes, with a uniformly sampled weight in $[0, 1]$. The resulting graphs are guaranteed to contain Hamiltonian cycles. However, in contrast to the complete graphs, finding such a Hamiltonian cycle is not trivial. Our results for the chord graphs are similar to the complete graphs, and for space constraints, we present them in the supplementary material. Training data was generated using the TSP solver in Google Optimization Tools[4].

As before, we train the DNN using Adam. We found it sufficient to use only 1k observation-action trajectories for our TSP domain. The metric used is average relative cost[5], defined as the ratio between the cycle cost of the learned policy and the Google solver, averaged over all initial nodes in each test domain. We also compare the DNN policy against a greedy policy which always picks the lowest-cost edge

[3]The Sokoban data-set from the learning for planning competition contains only 60 training domains, which is not enough to train a DNN. Our generator works as follows: we assume the room dimensions are a multiple of 3 and partition the grid into 3x3 blocks. Each block is filled with a randomly selected and randomly rotated pattern from a predefined set of 17 different patterns. To make sure the generated levels are not too easy and not impossible, we discard the ones containing open areas greater than 3x4 and discard the ones with disconnected floor tiles. For more details we refer the reader to Taylor et al. (Taylor and Parberry 2011).

[4]https://developers.google.com/optimization

[5]For the complete graphs, all policies always succeeded in finding a Hamiltonian cycle. For the chord graphs, we report success rates in the supplementary material.

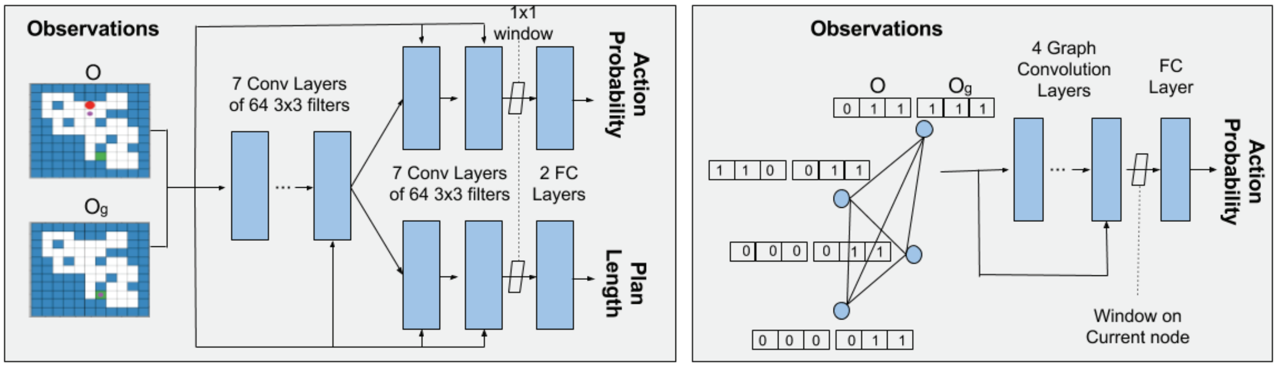


Figure 2: Network architecture. The architecture on the left is used for Sokoban, while the one on the right is used for the TSP. A pair of current and goal observations are passed in to a shared conv-net. This shared representation is input to an action prediction conv-net and a plan length prediction conv-net. Skip connections from the input observations to all conv-layers are added. For the TSP network, we omitted the plan length prediction, as the features directly encode the number of nodes visited, making the prediction trivial. All activation functions are ReLU's and the final one is a SoftMax. In both architectures, after the last convolution layer, we apply a $k \times k$ window around the agents location to ensure a constant size feature vector is passed to the fully connected layers. This effectively decouples the architecture from the problem size and allows the receptive field to be greater than the $k \times k$ window.

leading to an unvisited node.

As in the Sokoban domain, we evaluate performance on test domains with graphs of the same size as the training set, 4 node graphs, and on larger graphs with up-to 11 nodes.

## Evaluation of Learned GRPs

Here we evaluate performance of the learned GRPs on previously unseen test problems. Our results suggest that the GRP can learn a well-performing planning-like policy for challenging problems. In the Sokoban domain, on $9 \times 9$ grids, the learned GRP in the best performing architecture (14 layers, with bootstrapping and a shared representation) can solve one-object Sokoban with 97% success rate, and two-object Sokoban with 87% success rate. Figure 1 shows a trajectory that the policy predicted in a challenging one-object domain from the test set. Two-object trajectories are difficult to illustrate using images; we provide a video demonstration at goo.gl/Hpy4e3. We observed that the GRP effectively learned to select actions that avoid dead ends far in the future, as Figure 1 demonstrates. The most common failure mode is due to cycles in the policy, and is a consequence of using a deterministic policy. Due to space constraints, further analysis of failure modes is given in the supplementary material. The learned GRP can thus be used to solve new planning problem instances with a high chance of success. In domains where simulators are available, a planner can be used as a fallback if the policy fails in simulation.

Figure **??** shows the performance of the GRP policy on complete graphs of sizes $4-11$, when trained on graphs of the same size (respectively). For both the GRP and the greedy policy, the cost increases approximately linearly with the graph size. For the greedy policy, the rate of cost increase is roughly twice the rate for the GRP, showing that the GRP learned to perform some type of lookahead planning.

## Investigation of Network Structure

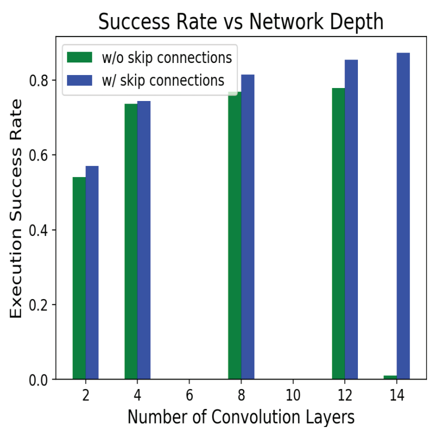
We performed ablation experiments to tease out the important ingredients for a successful GRP. Our results suggest that deeper networks improve performance.

In Figure 3a we plot execution success rate on two-object Sokoban, for different network depths, and with or without skip connections. The results show that deeper networks perform better, with skip connections resulting in a consistent advantage. In the supplementary material we show that a deep network significantly outperformed a shallow network with the same number of parameters, further establishing this claim. The improved results for the deeper networks suggest that for learning GRP's – **the deeper the network the better**. We note a related observation in the context of a DNN representation of the value iteration planning algorithm in (Tamar et al. 2016). However, in our experiments the performance levels off after 14 layers. We attribute this to the general difficulty of training deep DNNs due to gradient propagation, as evident in the failure of training the 14 layer architecture without skip connections, Figure 3a.

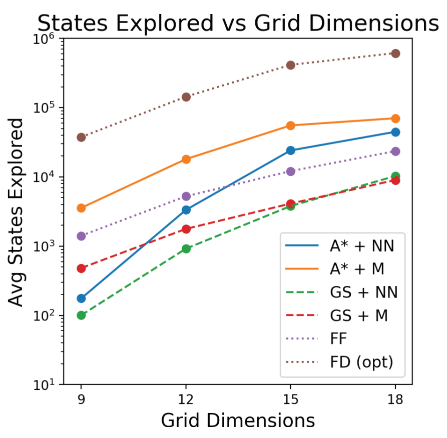
We also investigated the benefit of having a shared representation for both action and plan length prediction, compared to predicting each with a separate network. The ablation results are presented in Table 1. Interestingly, the plan length prediction improves the accuracy of the action prediction.

## GRP as a Heuristic Generator

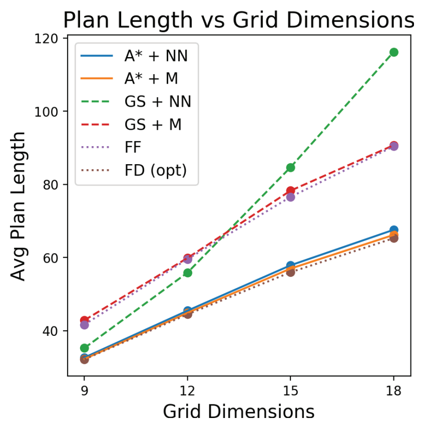
We now show that the learned GRPs can be used to extract *representation independent heuristics* for use with arbitrary guided search algorithms. To our knowledge, there are no other approaches for computing such heuristics without using hand-curated domain vocabularies or features for learning and/or expressing them. However, to evaluate the quality of our learned heuristics, we compared them with a few well-known heuristics that are either handcrafted or com-



(a)

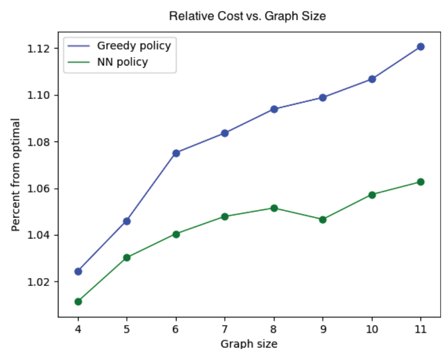


(b)

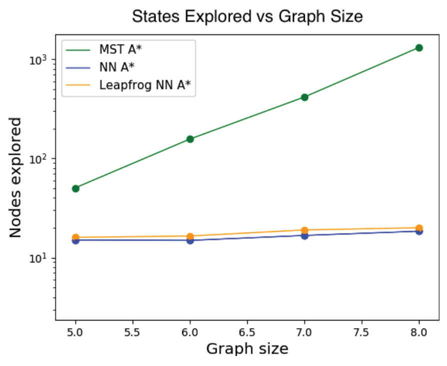


(c)

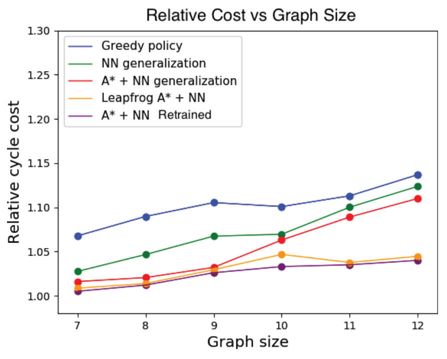
Figure 3: Sokoban results. (a) Investigating DNN depth and skip connections. We plot the success rate for deterministic execution in two-object Sokoban. Deeper networks show improved success rates and skip connections improve performance consistently. We were unable to successfully train a 14 layer deep network without skip connections. (b,c) Performance of learned heuristic. The GRP was trained only on 9x9 instances, and evaluated (as a heuristic, see text for more details) on larger instances. (b) shows number of states explored (i.e., planning speed) and (c) shows plan length (i.e., planning quality). A* with the learned heuristic produced nearly optimal plans with an order of magnitude reduction in the number of states explored.



(a)



(b)



(c)

Figure 4: TSP results. (a) Performance (average relative cost; see text for details) for GRPs trained and tested on problems of sizes $4 - 11$, respectively. We compare the GRP with a greedy policy. (b,c) Performance of learned heuristic. The GRP was trained on 4-node graphs, and evaluated (as a heuristic, see text for more details) on larger instances. (b) shows number of states explored (i.e., planning speed). We compare with the minimum spanning tree heuristic, which is admissible for TSP. (c) shows average relative cost (i.e., planning quality) compared to plans from the Google solver. Note that up to a graph of size $9$, the performance of A$^*$ with GRP heuristic (labeled A$^*$+NN generalization) was within $5\%$ of optimal, while requiring orders of magnitude less computation than the MST heuristic. We also present results for the leapfrogging algorithm (see text for details), and additionally compare to a baseline of retraining the GRP with optimal data for each graph size. Note that the leapfrogging results are very close to the results obtained with retraining, although optimal data was only given for the smallest graph size. This shows that the GRP heuristic can be used for generating reliable training data for domains of larger size than trained on.

puted using handcrafted representations. We found that the representation-independent GRP heuristic was competitive, and remains effective on larger problems than the GRP was trained on. For the Sokoban domain, the plan-length prediction can be directly used as a heuristic function. This approach can be used for state-property based goals in problems where execution can be captured using images. For the TSP domain, we used a heuristic that is inversely proportional to the probability of selecting the next node to visit, as the number of steps required to create a complete cycle is not discriminative. Full details are given in the supplementary material.

We investigated using the GRP as a heuristic in greedy search and A$^*$ search (Hart, Nilsson, and Raphael 1968). We use two performance measures: the number of states explored during search and the length of the computed plan. The first measure corresponds to planning speed since evaluating less nodes translates to faster planning. The second measure rep-

|  | w/ bootstrap | w/o bootstrap |  |
|---|---|---|---|
| Predict plan length | 2.211 | 2.481 | $\ell_1$ norm |
| Predict plan length | **2.205** | 2.319 | $\ell_1$ norm |
| & actions | **0.844** | 0.818 | Succ Rate |
| Predict actions | 0.814 | 0.814 | Succ Rate |

Table 1: Benefits of bootstrapping and having a shared representation. To evaluate accuracy of the plan length prediction, we measure the average $\ell_1$ loss (absolute difference). To evaluate action prediction we measure the success rate on execution. Best performance was obtained with using bootstrapping and the shared representation. For this experiment the training set contained 25k observation-action trajectories.

resents plan quality.

**Sokoban** We compare performance in Sokoban to the Manhattan heuristic[6] in Figure 3b. In the same figure we evaluate generalization of the learned heuristic to larger, never before seen, instances as well as the performance of two state-of-the-art planners: Fast Forward (FF, (Jörg Hoffman 2001)) and Fast Downward (FD, (Helmert 2006))[7]. The GRP was trained on $9 \times 9$ domains, and evaluated on new problem instances of similar size or larger. During training, we chose the window size $k = 1$ to influence learning a problem-instance-size-invariant policy. As seen in Figure 3b the learned GRP heuristic *significantly outperforms the Manhattan heuristic* in both greedy search and A* search, on the 9x9 problems. As the size of the test problems increases, the learned heuristic shines when used in conjunction with A*, consistently expanding fewer nodes than the Manhattan heuristic. Note that even though the GRP heuristic is not guaranteed to be admissible, when used with A*, the plan quality is very close to optimal, while exploring an order of magnitude less nodes than the conventional alternatives.

**TSP** We trained the GRP on 6-node complete graphs and evaluated the GRP, used either directly as a policy or as a heuristic within A$^*$, on graphs of larger size. Figure 4(b-c) shows generalization performance of the GRP, both in terms of planning speed (number of nodes explored) and in terms of plan quality (average relative cost). We compare both to a greedy policy, and to A$^*$ with the minimum spanning tree (MST) heuristic. Note that the GRP heuristic is significantly more efficient than MST, while not losing much in terms of plan quality, especially when compared to the greedy policy.

## Leap-Frogging Algorithm

The effective generalization of the GRP heuristic to larger problem sizes motivates a novel algorithmic idea for learning to plan on iteratively increasing problem sizes, which we term *leap-frogging*. The idea is that, we can use a 'general and optimal' planner, such as FD, to generate data for a small domain, of size $d$. We then train a GRP using this data, and use the resulting GRP heuristic in A$^*$ to *quickly* solve planning problems from a larger domain $d' > d$. These solutions can then be used as new data for training another GRP on the domain size $d'$. Thus, we can iteratively apply this procedure to solve problems of larger and larger sizes, while only requiring the slow 'general' planner to be applied in the smallest domain size.

In Figure 4c we demonstrate this idea in the TSP domain. We used the solver to generate training data for a graph with 4 nodes. We then evaluate the GRP heuristic trained using leapfrogging on larger domains, and compare with a GRP heuristic that was only trained on the 4-node graph. Note that we significantly improve upon the standard GRP heuristic, while using the same initial optimal data obtained from the slow Google solver. We also compare with a GRP heuristic that was re-trained with optimal data for each graph size. Interestingly, this heuristic performed only slightly better than the GRP trained using leap-frogging, showing that the generalization of the GRP heuristic is effective enough to produce reliable new training data.

## Conclusion

We presented a new approach in learning for planning, based on imitation learning from execution traces of a planner. We used deep convolutional neural networks for learning a generalized policy, and proposed several network designs that improve learning performance in this setting, and are capable of generalization across problem sizes. In addition, we showed that our networks can be used to extract a heuristic for off-the-shelf planners, which led to significant improvements over standard heuristics that do not leverage learning.

Our results on the challenging Sokoban domain suggest that DNNs have the capability to extract powerful features from observations, and the potential to learn the type of 'visual thinking' that makes some planning problems easy for humans but very hard for automatic planners. The leapfrogging results, suggest a new approach for planning – when facing a large and difficult problem, first solve simpler instances of the problem and learn a DNN heuristic that aids search algorithms in solving larger instances. This heuristic can be used to generate data for training a new DNN heuristic for larger instances, and so on. Our preliminary results suggest this approach to be promising.

There is still much to explore in employing deep networks for planning. While representations for images based on deep conv-nets have become standard, representations for other modalities such as graphs and logical expressions are an active research area (Dai et al. 2017; Kansky et al. 2017). We believe that the results presented here will motivate future research in representation learning for planning.

[6]The Manhattan heuristic is only admissible in one-object Sokoban. We tried Euclidean distance and Hamiltonian distance. However, Manhattan distance had the best trade-off between performance and computation time.

[7]FD uses an anytime algorithm, so we constrained the planning time to be no more than 5 minutes per instance. For the problem instances we evaluated, FD always found the optimal solution.

## References

Bylander, T. 1994. The computational complexity of propositional strips planning. *Artificial Intelligence* 69(1-2):165–204.

Dai, H.; Khalil, E. B.; Zhang, Y.; Dilkina, B.; and Song, L. 2017. Learning combinatorial optimization algorithms over graphs. *arXiv preprint arXiv:1704.01665*.

Duan, Y.; Andrychowicz, M.; Stadie, B.; Ho, J.; Schneider, J.; Sutskever, I.; Abbeel, P.; and Zaremba, W. 2017. One-shot imitation learning. *arXiv preprint arXiv:1703.07326*.

Ernandes, M., and Gori, M. 2004. Likely-admissible and sub-symbolic heuristics. In *Proceedings of the 16th European Conference on Artificial Intelligence*, 613–617. IOS Press.

Fern, A.; Khardon, R.; and Tadepalli, P. 2011. The first learning track of the international planning competition. *Machine Learning* 84(1):81–107.

Fikes, R. E.; Hart, P. E.; and Nilsson, N. J. 1972. Learning and executing generalized robot plans. *Artificial Intelligence* 3:251 – 288.

Fox, M., and Long, D. 2003. PDDL2. 1: An extension to PDDL for expressing temporal planning domains. *J. Artif. Intell. Res.(JAIR)* 20:61–124.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics* 4(2):100–107.

Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence (JAIR)* 26:191–246.

Helmert, M. 2009. Concise finite-domain representations for pddl planning tasks. *Artificial Intelligence* 173(5):503 – 535.

Hu, Y., and De Giacomo, G. 2011. Generalized planning: Synthesizing plans that work for multiple environments. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*.

Jörg Hoffman. 2001. FF: The fast-forward planning system. *AI Magazine* 22:57–62.

Kambhampati, S., and Kedar, S. 1994. A unified framework for explanation-based generalization of partially ordered and partially instantiated plans. *Artificial Intelligence* 67(1):29–70.

Kansky, K.; Silver, T.; Mély, D. A.; Eldawy, M.; Lázaro-Gredilla, M.; Lou, X.; Dorfman, N.; Sidor, S.; Phoenix, S.; and George, D. 2017. Schema networks: Zero-shot transfer with a generative causal model of intuitive physics. *arXiv preprint arXiv:1706.04317*.

Khardon, R. 1999. Learning action strategies for planning domains. *Artificial Intelligence* 113(1):125 – 148.

Kingma, D., and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 1097–1105.

Long, J.; Shelhamer, E.; and Darrell, T. 2015. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 3431–3440.

Martín, M., and Geffner, H. 2004. Learning generalized policies from planning examples using concept languages. *Applied Intelligence* 20(1):9–19.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529–533.

Mülling, K.; Kober, J.; Kroemer, O.; and Peters, J. 2013. Learning to select and generalize striking movements in robot table tennis. *The International Journal of Robotics Research* 32(3):263–279.

Nair, A.; Chen, D.; Agrawal, P.; Isola, P.; Abbeel, P.; Malik, J.; and Levine, S. 2017. Combining self-supervised learning and imitation for vision-based rope manipulation. *arXiv preprint arXiv:1703.02018*.

Pfeiffer, M.; Schaeuble, M.; Nieto, J.; Siegwart, R.; and Cadena, C. 2016. From perception to decision: A data-driven approach to end-to-end motion planning for autonomous ground robots. *arXiv preprint arXiv:1609.07910*.

Pomerleau, D. A. 1989. Alvinn: An autonomous land vehicle in a neural network. In *Advances in Neural Information Processing Systems*, 305–313.

Ross, S.; Gordon, G. J.; and Bagnell, D. 2011. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*.

Scala, E.; Torasso, P.; et al. 2015. Deordering and numeric macro actions for plan repair. In *IJCAI*, 1673–1681.

Shavlik, J. W. 1989. Acquiring recursive concepts with explanation-based learning. In *IJCAI*, 688–693.

Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; et al. 2017. Mastering the game of go without human knowledge. *Nature* 550(7676):354–359.

Spalzzi, L. 2001. A survey on case-based planning. *Artificial Intelligence Review* 16(1):3–36.

Srivastava, S.; Immerman, N.; and Zilberstein, S. 2011. A new representation and associated algorithms for generalized planning. *Artificial Intelligence* 175(2):615–647.

Sutskever, I.; Vinyals, O.; and Le, Q. V. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, 3104–3112.

Tamar, A.; Levine, S.; Abbeel, P.; WU, Y.; and Thomas, G. 2016. Value iteration networks. In *Advances in Neural Information Processing Systems*, 2146–2154.

Taylor, J., and Parberry, I. 2011. Procedural generation of sokoban levels. In *Proceedings of the International North American Conference on Intelligent Games and Simulation*.

Tesauro, G. 1995. Temporal difference learning and td-gammon. *Communications of the ACM* 38(3):58–68.

Weber, T.; Racanière, S.; Reichert, D. P.; Buesing, L.; Guez, A.; Rezende, D. J.; Badia, A. P.; Vinyals, O.; Heess, N.; Li, Y.; et al. 2017. Imagination-augmented agents for deep reinforcement learning. *arXiv preprint arXiv:1707.06203*.

Yoon, S.; Fern, A.; and Givan, R. 2002. Inductive policy selection for first-order MDPs. In *Proceedings of the Eigh-*

*teenth conference on Uncertainty in artificial intelligence*, 568–576. Morgan Kaufmann Publishers Inc.

Yoon, S.; Fern, A.; and Givan, R. 2008. Learning control knowledge for forward search planning. *Journal of Machine Learning Research* 9(Apr):683–718.

## Appendix

### Graph Convolution Network

Consider a graph $\mathcal{G} = (V, \mathcal{E})$ with adjacency matrix $A$ where $V$ has $N$ nodes and $\mathcal{E}$ is the weighted edge set with weight matrix $W$. Suppose that each node $v \in V$ has a corresponding feature $x_v \in \mathbb{R}^m$ and consider a parametric function $f_\theta : \mathbb{R}^{2m} \to \mathbb{R}^m$ parameterized by $\theta \in \mathbb{R}^f$. Let $\mathcal{N}_i : V \to 2^V$ denote a function mapping a vertex to its $i$th degree neighborhood. The propagation rule is given by the following equation

$$H_v = \sigma\left(\sum_{u \in \mathcal{N}(v)} A_{uv} f_\theta(x_u, x_v)\right) \tag{1}$$

where $\sigma$ is the ReLU function. Consider a graph $\mathcal{G}$ of size $n$, with each vertex having feature vector of size $C$ encoded in the feature matrix $X \in \mathbb{R}^{n \times C}$. In the TSP experiments, we use the propagation rule where the $ij$ entry of the next layer is given by

$$H_{ij} = \sigma\left(\sum_{s \in \mathcal{N}(i)} A_{si}[x_s, x_i, W_{si}]^T \Theta_j + b_j\right) \tag{2}$$

Here, $W$ is the weight matrix of $\mathcal{G}$, $A$ is the adjacency matrix, and $\Theta \in \mathbb{R}^{(2C+1)\times C'}$ is the matrix of weights that we learn and $b \in \mathcal{R}^{C'}$ is a learned bias vector. $\Theta_j$ is the $j$th column of $\Theta$.

In the networks we used for the TSP domain, the initial feature vector is of size $C = 6$. We then applied $4$ convolution layers of size $C = 26$. We then applied a convolution of size $C = 1$, corresponding to a fully connected layer. Thus, $j = 1$ in $H_{ij}$ for all $i$ in the last convolution layer.

The final layer of the network is a softmax over $H_{i1}$, and we select the node $i$ with the highest score that is also connected to the current node.

**Relation to Image Convolution** In the next proposition we show that this graph-based propagation rule can be seen as a generalization of a standard 2-D convolution, when applied to images (grid graphs). Namely, we show that there exists features for a grid graph and parameters $\Theta$ for which the above propagation rule reduces to a standard 2-D convolution.

**Proposition 2.** *When $\mathcal{G}$ is a grid graph, for a particular choice of $f_\theta$ the above propagation rule reduces to the traditional convolutional network. In particular, for a filter of size $n$, choosing $f_\theta$ as a polynomial of degree $2(N-1)$ and $\theta \in \mathbb{R}^{N^2}$ works.*

*Proof.* For each node $v$, consider its representation as $v = (v_x, v_y)$ where $(v_x, v_y)$ are the grid coordinates of the vertex.

| Num Params | Deep-8 | Wide-2 | Wide-1 | |
|---|---|---|---|---|
| 556288 | 0.068 | 0.092 | 0.129 | error rate |
| | 0.83 | 0.62 | 0.38 | succ rate |

Table 2: Comparison of deep vs. shallow networks. The deep network has 8 convolution layers with 64 filter per layer. The shallow networks contain 2 and 1 layers respectively with 256 and 512 filters per layer respectively. Clearly, deeper networks outperform shallow networks while containing an equal number of parameters.

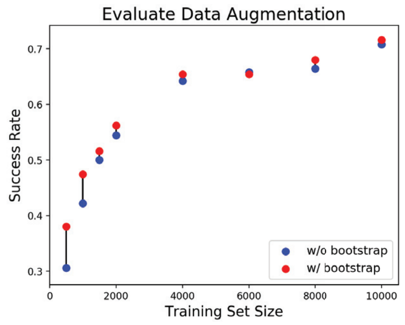


Figure 5: This shows the affect of data bootstrapping on the performance of two-object Sokoban, as a function of the dataset size. Smaller datasets benefit more from data augmentation.

Let $a := \frac{n-1}{2}$. We first transform the coordinates to center them around $v$ by transforming $u \to (u_x - v_x, u_y - v_y)$ so that $u$ lies in the set $[-a, a] \times [-a, a]$.

We wish to design a polynomial $g$ that takes the value $\theta_{i,j}$ at location $(i, j)$. We show that it is possible to do with a degree $2(n-1)$ polynomial by construction. The polynomial $g$ is given by

$$g(x, y) := \sum_{i=-a}^{a} \sum_{j=-a}^{a} \theta_{i,j} \prod_{s=-a, s \neq i}^{a} (s + y) \prod_{t=-a, t \neq j}^{a} (t + x) \tag{3}$$

To see why this is correct, note that for any $(s, t) \in [-a, a] \times [-a, a]$ there is exactly one polynomial inside the summands that does not have either of the terms $(i + u_y)$ or $(j + u_x)$ appearing in its factorization. Indeed, by construction this term is the polynomial corresponding to $\theta_{i,j}$ so that $g(i, j) = C\theta_{i,j}$ for some constant $C$.

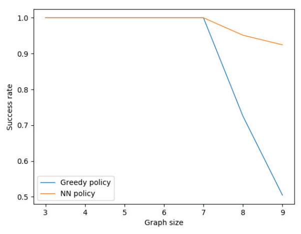
The polynomial inside the summands is of degree $(n-1) + (n-1) = 2(n-1)$, so $g$ is of degree $2(n-1)$. Letting $p_u$ denote th pixel value at node $u$, setting

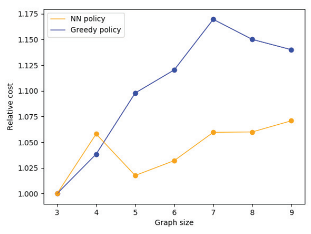
$$f_\theta(x_u, x_v) := p_u g(x_u - x_v) \tag{4}$$

completes the proof. $\square$

### TSP domain heuristic

We can use the graph convolution network as a heuristic inside A-star search. Given a feature encoding of a partial cycle $P$, we can compute the probability $p_i$ of moving to


(a)


(b)

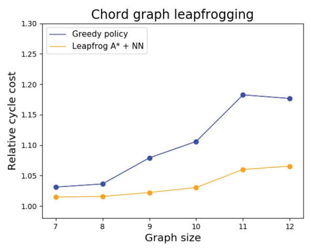
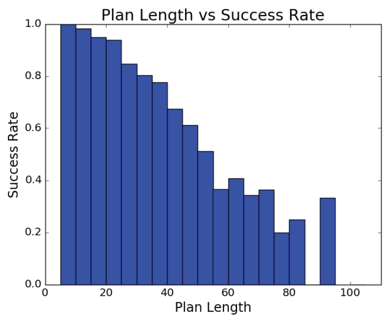
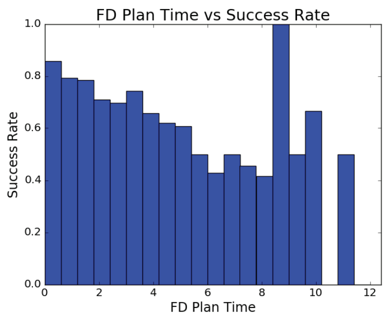
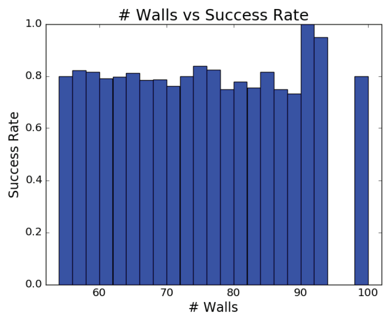
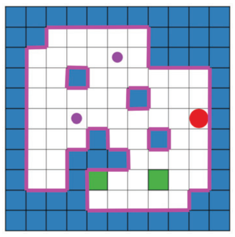

(c)

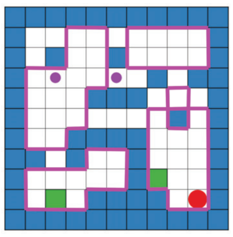
Figure 6: Chord-graph TSP results. (a) Success rate of neural network policy on chord graphs of size $3-9$, respectively. Note that the agent is only allowed to visit each node once, so the agent may visit a node with no un-visited neighbors which is a dead end. We also show the success rate of the greedy policy. (b) Performance of neural network policy on chord graphs of size 3-9. (c) Leapfrogging algorithm results on chord graphs of size 7-12. We compare to a baseline greedy policy

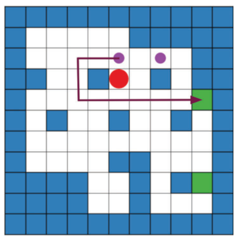

(a)

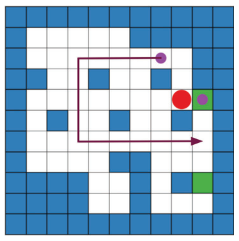

(b)


(c)


(d)


(e)


(f)


(g)

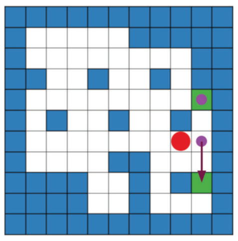

(h)

Figure 7: Analysis of Failure Modes. (a-c): Success rate vs features of the domain. Plan length (a) seems to be the main factor in determining success rate. Longer plans fail more often. While there is some relationship between planning time and success rate (b), planning time is not always an accurate indicator, as explained in (d,e). The number of walls (c) does not affect success rate. (d,e): Domains containing large open rooms results in a high branching factor and thus produce the illusion of difficulty while still having a simple underlying policy. The domain in (d) took FD significantly longer time to solve, 8.6 seconds compared to 1.6 seconds for the domain in (e), although it has a shorter optimal solution, 51 steps compared to 65 steps. This is since the domain in (e) can be broken up into small regions which are all connected by hallways, a configuration that reduces the branching factor and thus the overall planning speed. (f-h): Demonstration of the 2nd failure mode in Section . From the start state, the policy moves the first object using the path shown in (f). It proceeds to move the next object using the path in (g). As the game state approaches (h) it becomes clear that the current domain is no longer solvable. The lower object needs to be pushed down but is blocked by the upper object, which can no longer be moved out of the way. In order to solve this level, the first object must ether be moved to the bottom goal or must be moved after the second object has been placed at the bottom goal. Both solutions require a look-ahead consisting of 20+ steps.

any node $i$. We then use the quantity $(N-v)(1-p_i)/2$ as the heuristic, where $N$ is the total number of nodes and $v$ is the number of visited nodes in the current partial path. Multiplying by $(N-v)/2$ puts the output of the heuristic on the same scale as the current cost of the partial path.

## Deep VS Shallow Networks

Here we present another experiment to further establish the claim that the depth of the network improves performance and not necessarily the number of parameters in the network. In Table 2 we compare deep networks against shallow networks containing the same number of parameters. Note that we evaluate based on two different metrics. The first metric is classification error on the next action, which shows whether or not the action matches what the planner would have done. The second metrics is execution success rate, as defined above.

## Evaluation of Bootstrap Performance

We briefly summarize the evaluation of data bootstrapping in the Sokoban domain. Table 1 shows the success rate and plan length prediction error for architectures with and without the bootstrapping. As can be observed, the bootstrapping resulted in better use of the data, and led to improved results.

While investigating the performance of data bootstrapping with respect to training set size, we observed that a non-uniform sampling performed better on smaller datasets. For each $\tau \in D_{\text{imitation}}$, we sampled an observation $\hat{o}$ from a distribution that is linearly increasing in time, such that observations near the goal have higher probability. The performance of this bootstrapping strategy is shown in Figure 5. As should be expected, performance improvement due to data augmentation is more significant for smaller data sets.

## Analysis of Failure Modes

While investigating the failure modes of the learned GRP in the Sokoban domain, we noticed that there were two primary failure modes. The first failure mode is due to cycles in the policy, and is a consequence of using a deterministic policy. For example, when the agent is between two objects a deterministic policy may oscillate, moving back and fourth between the two. We found that a stochastic policy significantly reduces this type of failure. However, stochastic policies have some non-zero probability of choosing actions that lead to a dead end (e.g., pushing the box directly up against a wall), which can lead to different failures. The second failure mode was the inability of our policy to foresee long term dependencies between the two objects. An example of such a case is shown in Figure 7 (f-h), where deciding which object to move first requires a look-ahead of more than 20 steps. A possible explanation for this failure is that such scenarios are not frequent in the training data. This is less a limitation of our approach and more a limitation of the neural network, more specifically the depth of the neural network.

Additionally, we investigated whether the failure cases can be related to specific features in the task. Specifically, we considered the task plan length (computed using FD), the number of walls in the domain, and the planning time with the FD planner (results are similar with other planners). Intuitively, these features are expected to correlate with the difficulty of the task. In Figure 7 (a-c) we plot the success rate vs. the features described above. As expected, success rate decreases with plan length. Interestingly, however, several domains that required a long time for FD were ‘easy’ for the learned policy, and had a high success rate. Further investigation revealed that these domains had large open areas, which are ‘hard’ for planners to solve due to a large branching factor, but admit a simple policy. An example of one such domain is shown in Figure 7 (d-e). We also note that the number of walls had no visible effect on success rate – it is the configuration of the walls that matters, and not their quantity.

# Teaching Virtual Agents to Perform Complex Spatial-Temporal Activities

**Tuan Do, Nikhil Krishnaswamy, James Pustejovsky**
Department of Computer Science
Brandeis University
Waltham, MA 02453 USA
{tuandn, nkrishna, jamesp}@brandeis.edu

**Abstract**

In this paper, we introduce a framework in which computers learn to enact complex temporal-spatial actions by observing humans, and outline our ongoing experiments in this domain. Our framework processes motion capture data of human subjects performing actions, and uses qualitative spatial reasoning to learn multi-level representations for these actions. Using reinforcement learning, these observed sequences are used to guide a simulated agent to perform novel actions. To evaluate, we render the action being performed in an embodied 3D simulation environment, which allows evaluators to judge whether the system has successfully learned the novel concepts. This approach complements other planning approaches in robotics and demonstrates a method of teaching a robotic or virtual agent to understand predicate-level distinctions in novel concepts.

## Motivation

The community surrounding "learning from (human) observation" (LfO) studies how computational and robotic agents can learn to perform complex tasks by observing humans (Young and Hawes 2015). Work in this area can be traced back to reinforcement learning studies by (Smart and Kaelbling 2002) or (Asada, Uchibe, and Hosoda 1999), which closely resembles the way humans learn. Children, as early as 14 months old, can imitate adults in a variety of tasks, such as *turning on and off a light-box*, and can even interpret the intentions behind actions and consider all constraints involved (Gergely, Bekkering, and Király 2002).

Most robots developed in the previous decades have shipped with pre-installed programs, limited to a set of predefined functionalities. Learning approaches in the robotics community seek to move toward smarter and more adaptable robots, for the following reasons, among others:

- Consumer desire for mobile or household assistant robots that can perform multiple tasks with a flexible apparatus, such as multiple grasping arms (Bogue 2017). Robots with behavioral robustness can learn from a wider range of experiences by interacting with humans in a dynamic environment (Hawes et al. 2017).
- Advances in deep learning have afforded robotic agents a high-level understanding of embedded semantics in multiple modalities, including language, gesture, object recognition, and navigation. This increases the circumstances and modalities available for robotic learning.

Event recognition and classification have achieved recent relevance in human communication with robotic agents (Paul et al. 2017). Meanwhile, lexical computational semantic approaches to events (e.g., Pustejovsky (1995), Pustejovsky and Moszkowicz (2011)) make it clear that event semantics are compositional with their arguments.

We have previously presented an approach toward facilitating human communication with a computational agent, using a rich model of events and their participants (Pustejovsky, Krishnaswamy, and Do 2017). Formally, we have devised a semantic framework using *Multimodal Semantic Simulations (MSS)*, which can be used to encode events as programs in a dynamic logic with an operational semantics. Computationally, we have been looking at event representation through sequential modeling, using data from 3-dimensional video captures, to distinguish between different event classes (Do and Pustejovsky 2017a). In this work, we aim to bridge the gap between these two lines of research by proposing a methodology to learn programmatic event representations from linguistic and visual event representations.

Linguistic event representation in our framework is modeled as a verbal subcategorization in a frame theory, a la Framenet (Baker, Fillmore, and Lowe 1998), with thematic role arguments. However, we also account for *extra-verbal factors* in our event type distinction. For example, we consider *A moves B toward C* and *A moves B around C* to be different event types and we learn each event type as a separate action.

Our visual event representation comprises visual features extracted from tracked objects in captured videos or virtual object positions saved from a simulation environment. Both types of feature represent information visible to humans and observable by a machine in an object state. Using these data points and sequences, machines can observe humans performing actions through processing captured and annotated videos, while humans can observe machines performing actions through watching simulated scenes.

Programmatic event representation can be based on formal event semantics or on features that can direct simulated or robotic agents to perform an action with an object of given properties. From a human perspective, the distinction between learning to recognize and learning to perform an action might be obvious. However from a machine's perspective, these two tasks might require different learning methods. Our work aims to demonstrate that given an appropriate framework, it is feasible to map between them, in a manner similar to the way humans actually learn: by matching actions to observations.

In this paper, (1) we discuss related work in AI that focuses on the learning of action and object models, including our own past studies; (2) we discuss several technologies and machine learning methodologies that provide the foundation for our experiments; (3) we discuss our ongoing experiment to learn actions; (4) we discuss our evaluation scheme and possible extensions to our framework.

## Related Work

Work on action and object representation can generally be divided into two types of approaches: bottom-up approaches and top-down approaches.

Bottom-up approaches include both unsupervised and supervised feature-based learning. Work such as (Duckworth et al. 2016; Alomari et al. 2017) aims for unsupervised co-learning of object and event representations in the same step, and introduced the notion of a learned *concept* as an abstraction of feature spaces. In such a framework, "learnable" concepts are any distinctions meaningful to a human, such as a facial expression, color, object property, or action distinction, and these categories can then be assigned labels based on their commonly-occurring features. Notable supervised learning studies include (Koppula, Gupta, and Saxena 2013), which jointly models the human activities and object *affordances*, or attached behaviors which the object either facilitates by its geometry (which we term Gibsonian) (Gibson, Reed, and Jones 1982), or for which it is intended to be used (which we term "telic") (Pustejovsky 1995). Such a model could be used to distinguish longer activities by means of labeling sub-activities and object affordances: for example, labeling a "meal preparation" and its different subtasks based on understanding the objects involved at each step.

The foundation of our embodied event simulation is the modeling language known as VoxML (Visual Object Concept Modeling Language) (Pustejovsky and Krishnaswamy 2016). We encode verbal programs into a dynamic logic format from which we can conduct programmatic planning of complex events from atomic subevents. This is a top-down approach in which verbs are encoded with their subevent structures into programmatic "voxemes," or visual instantiations of lexemes which can then be visualized and enacted by an agent in a virtual environment. Subevent programs may themselves be linked to other voxemes, allowing for condition satisfaction, as in Figure 1, where "touching" is defined as the $EC$ (externally connected) relation in RCC (Region Connection Calculus (Randell et al. 1992)). This is underspecified and may be further constrained by relative orientations between the two objects involved: $x$ and $y$.

We aim to unify the two broad types of approaches outlined above using a form of *apprenticeship learning*, wherein a learning model observes an expert demonstrating the task that we want it to learn to perform. We propose a model, cf. (Abbeel and Ng 2004), in which reinforcement learning is used as a backbone for planning, while estimating a reward function as measuring the progression of the event-actions to be learned.

$$\begin{bmatrix} \textbf{touching} & \\ \text{LEX} = & \begin{bmatrix} \text{PRED} = \textbf{touching} \end{bmatrix} \\ \text{TYPE} = & \begin{bmatrix} \text{CLASS} = \textbf{config} \\ \text{VALUE} = \textbf{EC} \\ \text{ARGS} = \begin{bmatrix} \text{A}_1 = \textbf{x:3D} \\ \text{A}_2 = \textbf{y:3D} \end{bmatrix} \\ \text{CONSTR} = \textbf{nil} \end{bmatrix} \end{bmatrix}$$

Figure 1: Sample voxeme: [[TOUCHING]]

## Background

### Simulators

**VoxSim** Our simulated environment is built in **VoxSim** (Krishnaswamy and Pustejovsky 2016), a semantically-informed visual event simulator built on top of the Unity game engine (Goldstone 2009). VoxSim contains a 3D agent capable of manipulating objects in the virtual world by creating parent-child relationships between the objects and its joints to simulate grasping. Assuming the simulated agent's skeleton is isomorphic to the joint structure of a physical robot, this then allows us to simulate events in the 3D world that represent real-world events (such as moving the virtual robot around a virtual table that has blocks on it in a configuration that is generated from the positioning of real blocks on a real table). The embodied agent can perform a set of simple actions:

- $ENGAGE$: grasp object near its end-effector.
- $MOVE(x)$: move end-effector (hand) to 3D point $x$, with parent limb motions calculated using inverse kinematics
- $DISENGAGE$: ungrasp current object, and retract the agent to standing position.

The simulation environment is used to demonstrate the agent's understanding of learned behavior, by enacting new behaviors over a set of virtual objects. Scenes generated by VoxSim will be used to evaluate performance of the system, as discussed later.

**Simplified Simulator** For the updating loops in our reinforcement learning algorithm, we want to simulate observational data similar to the real captured data faster than real-time for effective computation. As a real-time, graphics heavy simulator, VoxSim is not feasible for this portion of the task. We are aware of a few other physical simulation environments such as Gazebo[1], but as we do not focus on physical constraints in this study, so we implemented our own simplified simulator in Python.

[1]http://gazebosim.org/

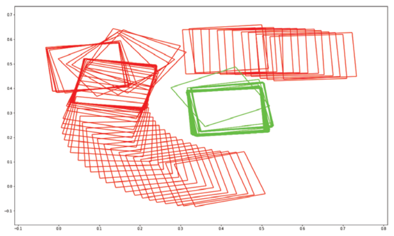


Figure 2: An event "Move A around B" projected into simulator. A is projected as a red square, B as a green square

Our set of learnable actions is limited to ones that can be easily approximated in 2D space. 3D captured data is transformed into simplified simulator space by projecting it onto a 2D plane defined by the surface of the table used for performing the captured interaction. Our 2D simulator has the following features:

- Each object is represented as a polygon (or square), with a $transform$ object that stores its position, rotation, and scale.
- The space is constrained so objects do not overlap.
- Speed can be specified so that object movement can be recorded as a sequence of feature vectors interpolated from frame to frame.

## Qualitative Spatial Reasoning

Qualitative spatial reasoning (QSR), a sub-field of qualitative reasoning, is considered to be formally akin to the way humans understand geometry and space, due to the cognitive advantages of conceptual neighborhood relations and its ability to draw coarse inferences under uncertainty (Freksa 1992). It is also considered a promising framework in robotic planning (Cohn and Renz 2001). QSR allows formalization of many qualitative concepts, such as *near*, *toward*, *in*, *around*, and facilitates learning distinctions between them (Do and Pustejovsky 2017b). QSR has many methods of accounting for relative vs. absolute relations, such as allowing *near* to be thresholded relative to an existing reference point (Renz and Nebel 2007), which reinforces the intuition that predicates such as *near* are inherently relative (Peters 2007). The use of qualitative predicates ensure that scenes which are semantically close have very similar feature descriptions. We use the following QSR types for feature extraction.

- CARDINAL DIRECTION measures relations between two objects as compass directions (north, northeast, etc.)
- MOVING or STATIC measures whether a point is moving or not.
- QUALITATIVE DISTANCE CALCULUS discretizes the distance between two moving points, following (Yang and Webb 2009).
- QUALITATIVE TRAJECTORY CALCULUS is a representation of motions between two objects by considering them as two moving point objects (MPOs).

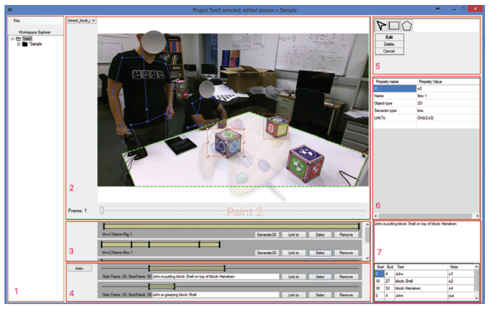


Figure 3: ECAT GUI showing performer interacting with recognized and annotated objects.

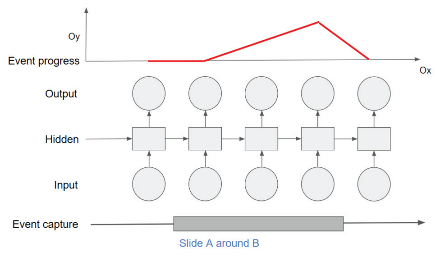


Figure 4: LSTM network producing event progress function

## Event Annotation Framework

We use an event capture and annotation tool developed in our lab, ECAT (Do, Krishnaswamy, and Pustejovsky 2016), which employs Microsoft Kinect® to capture performers interacting with objects in a blocks world environment. Objects are tracked using markers fixed to their sides. They are then projected into three dimensional space using Depth of Field (DoF). Performers are also tracked using the Kinect® API, which provides three dimensional inputs of their joint points (e.g., wrist, palm, shoulder).

## Learning Framework

**Sequential Learning** In this study, we consider a version of Long-short term memory (LSTM) (Hochreiter and Schmidhuber 1997) that processes sequential inputs to a sequence of output signals. LSTM has found utility in a range of problems involving sequential learning, such as speech and gesture recognition. Inputs are the feature vectors taken from action captures or from the simplified simulator and output is a function that corresponds to the progress of an event. In particular, we create a function that takes a sequence $S$ of feature vectors, current frame $i$ and action $e$: $f(S, i, e) = 0 \leq q_i \leq 1$

The training set of sequential captured data is passed through an LSTM network, which is fitted to predict a linear progressing function. At the start or outside of an event span, the network produces 0, whereas at the end, it produces 1.

**Reinforcement Learning** The objective of the embodied agent is to generate a sequence of actions to attain a maximum reward, whereas our reward corresponds to how closely the produced object movement resembles movement of objects in the training data. Visual (tracked) information is used to evaluate performance of the system.

Currently, the action space is continuous. Therefore, planning is carried out by selecting the action at step $k$ ($u_k$) based on the current state of the system ($X_k \in R^n$). A stochastic planning step is parameterized by policy parameters $\theta : u_k \sim \pi_\theta(u_k|x_k)$.

This type of parameterized reinforcement learning policies is best solved by using policy gradients (Gullapalli 1990; Peters and Schaal 2008). Here, we use the REINFORCE algorithm (Williams 1992), for its effectiveness in policy gradient learning.

We consider two versions of REINFORCE, which carry out planning in continuous and discrete search spaces, respectively. For continuous space, we propose using a Gaussian distribution policy $\pi_\theta(u|x) = Gaussian(\mu, \sigma)$. For simplicity, the dimensions of $\mu$ and $\sigma$ are the same as the degrees of freedom in our simplified simulator (2 dimensions for position and 1 dimension for rotation). An artificial neural network (ANN) will be used to produce values $\mu$ and $\sigma$. The set of weights in our ANN is the parameter $\theta$ from the REINFORCE algorithm, learned with gradient descent.

For discrete space, we again use a qualitative reasoning method. Specifically, the searching space for the *transform* of the target location could be separated into two spaces, for $(X, Y)$ coordinates and rotation $r$. The searching space for $(X, Y)$ could be discretized according to cardinal direction and quantized distance.

A searching method employing simple random search with back-up is used as baseline to evaluate performance of the progress learner. We will present some preliminary results from this searching method.

## Experiments

Here we describe our experimental setup and evaluation plans.

### Experiment

We aim to use the learning framework outlined above for teaching an agent to perform a set of actions where it interacts directly with a single object while the other objects stay relatively static and the interaction takes place over a continuous span.

1. An agent moves {object A} **closer to** {object B}
2. An agent moves {object A} **away from** {object B}
3. An agent moves {object A} **past** {object B}
4. An agent moves {object A} **next to** {object B}
5. An agent moves {object A} **around** {object B}

This set of actions differ only in their prepositional adjuncts, which describe different motion trajectories. Thus for this experiment, the learning problem is reduced to one of motion paths.

These actions are, however, generally classified into different event types. Using the treatment from (Pustejovsky 1991), an action such as "moves {object A} **next to** {object B}" is an *achievement*, which means it has a logical culmination or duration. Other actions do not have a defined ending, though for "moves {object A} **closer to** {object B}," this action is ended at the point when "{object A} is **next to** {object B}." From a cognitive point of view, recognition of these action types, except possibly for **move next to**, requires consideration of the trajectory as well as the start and ending points of the objects involved. For example, **closer to** conceptually involves change of distance between the start and the ending position of the moving object relative to the static object, but a complex motion path could lead to misinterpretation of the action. **Closer to**, therefore, strongly indicates a trajectory of the moving object toward the static object.

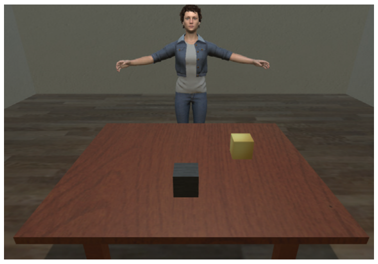


Figure 5: Visualizer implemented in Unity

By grouping the learning of different event types together, we aim to examine the capability of a single learning framework that to learn multiple event types. The reason is rather obvious: we, as humans, can learn all of these actions without prior knowledge of different action types.

For each action type, we are capturing 40 sessions of two different performers. Block positions are randomized at the start. We mark the beginning and end of the captured action and give it a textual description.

We generate frame-by-frame feature vectors by employing the set of aforementioned QSR features: cardinal direction and qualitative distance between objects' positions and frame-to-frame difference; qualitative trajectory for each object and frame-to-frame difference. These features are used only for the sequential model to predict event progress, whereas we use objects' parameters (positions and rotations) across consecutive frames as state of the system $X_k$.

### Evaluation

Human evaluation will be carried out on action demonstrations generated by both the 2D simulator and our lab's 3D visualizer, VoxSim (Figure 5). In VoxSim, we create a testbed scene with blocks on a table, similar to the setup used in video captures. For each randomized configuration of objects (block positions and rotations), we command the virtual agent to perform one of the actions, and the scene is recorded for evaluators to judge its performance.

Our human-driven evaluation method aims to help answer the following questions:

1. Does the virtual agent learn the concept in question? Re-

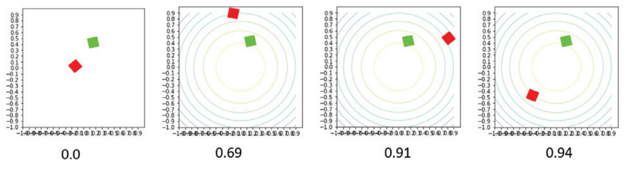


Figure 6: A correct demonstration of "Move red block around green block."

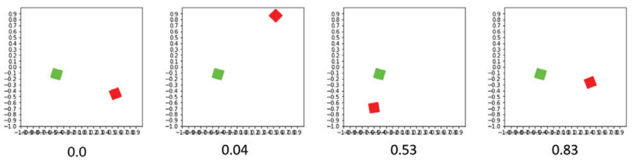


Figure 7: A wrong demonstration of "Move red block around green block." The value beneath each frame is value predicted by the progress learner.

flected by average score given to a demonstration when annotators know the action label.

2. Can the virtual agent make distinctions between learned actions? Reflected by confusion matrix when annotators have to label the action performed in a scene.
3. Will evaluation scores on the 2D simulator significantly differ from those on the 3D visualizer?
4. Can we use the feedback from human evaluation to improve the learned model? Generated demonstrations with feedback scores complement real, captured data, and in some sense are better than learning by demonstration, in that they provide a rigorous way to include negative samples.

Evaluations of this type using VoxSim-generated scenes have already been conducted in (Krishnaswamy 2017; Krishnaswamy and Pustejovsky 2017), using Amazon Mechanical Turk to crowdsource judgments. Human judgments of a scene are given as "acceptable" or "unacceptable" relative to the event's linguistic description.

## Preliminary results

Preliminary runs of the system with brute-force searching show that the progress learner can help to generate correct demonstrations (Fig. 6), but sometimes produces deviations (Fig. 7), probably because of the lack of negative training samples. We hope that incorporating feedback from evaluators will improve the overall performance of the learner.

We also provide a quantitative breakdown of a small-scale human evaluation in Table 1. Two annotators (college students) are asked to give scores from 0 to 10 and are also asked to give comments on any video they graded between 3 and 7 (higher scores are considered better). **Evaluator Disparity** is the average of the absolute values of the differences between scores given by two annotators over the demonstrations of a particular action.

| Action Type | Average Score | Evaluator Disparity |
|---|---|---|
| Slide Closer | 5.4 | 1.57 |
| Slide Away | 6.48 | 2.37 |
| Slide Next To | 5.55 | 1.7 |
| Slide Past | 6.38 | 1.9 |
| Slide Around | 2.75 | 1.03 |

Table 1: Evaluation

Evaluator comments provide some insight into bad demonstrations. Typical comments on *Slide Next To* include "Need to be even closer", while on *Slide Closer To* a typical comment is "The blocks touched." That suggests some confusion between these two actions, which requires a method to help distinguish them. Three reasons are given by evaluators for low scores on *Slide Around* demonstrations: the movement being not smooth, one or more additional steps needed for completion, and many cases where the algorithm does not generate the proper trajectory.

Code, experimental and evaluation results can be found on GitHub[2]. Complete experimental results will be forthcoming at that address.

## Conclusion

Two different lines of research may be extended from this framework. One involves a learning mechanism for more complex actions, such as "make a row from given objects," and one involves learning the "manner of motion" of actions.

Learning complex actions from simpler actions requires an additional semantic framework for objects and actions. For example, to learn "make a row from given objects" given observations of 2-unit and 3-unit rows, the learner needs to be equipped with the concept of *recursion*, the concept of a composite object made from elementary objects (e.g. the size and shape of the composite object), and other abstract concepts, such as object axis and extension of a structure along said axis.

Learning the manner aspect of actions requires a finer-grained treatment of object affordances. For example, for the learner to distinguish "rolling a bottle" and "sliding a bottle," we need to equip it with a reasoning mechanism to determine how an object's pose and position dictate its affordances. VoxML, the underlying platform to the VoxSim system, supports modeling these types of affordance distinctions, so reference to the VoxML semantics of objects and events can provide the reasoner with the mechanism for distinguishing these behavior types, as illustrated by (Krishnaswamy and Pustejovsky 2016).

## Acknowledgements

We would like to thank the reviewers for their helpful comments. This work is supported by a contract with the US Defense Advanced Research Projects Agency (DARPA), Contract W911NF-15-C-0238. Approved for Public Release, Distribution Unlimited. The views expressed are those of

[2]https://github.com/tuandnvn/learn-to-perform/

the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government. We would like to thank Alex Luu and Dr. Pengyu Hong for their discussion and input on this topic. All errors and mistakes are, of course, the responsibilities of the authors.

## References

Abbeel, P., and Ng, A. Y. 2004. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, 1. ACM.

Alomari, M.; Duckworth, P.; Bore, N.; Hawasly, M.; Hogg, D. C.; and Cohn, A. G. 2017. Grounding of human environments and activities for autonomous robots. In *IJCAI-17 Proceedings*.

Asada, M.; Uchibe, E.; and Hosoda, K. 1999. Cooperative behavior acquisition for mobile robots in dynamically changing real worlds via vision-based reinforcement learning and development. *Artificial Intelligence* 110(2):275–292.

Baker, C. F.; Fillmore, C. J.; and Lowe, J. B. 1998. The berkeley framenet project. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics-Volume 1*, 86–90. Association for Computational Linguistics.

Bogue, R. 2017. Domestic robots: Has their time finally come? *Industrial Robot: An International Journal* 44(2):129–136.

Cohn, A. G., and Renz, J. 2001. Qualitative spatial representation and reasoning. 46:1–2.

Do, T., and Pustejovsky, J. 2017a. Fine-grained event learning of human-object interaction with lstm-crf. *Proceedings of the European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*.

Do, T., and Pustejovsky, J. 2017b. Learning event representation: As sparse as possible, but not sparser. *arXiv preprint arXiv:1710.00448*.

Do, T.; Krishnaswamy, N.; and Pustejovsky, J. 2016. Ecat: Event capture annotation tool. *Proceedings of ISA-12: International Workshop on Semantic Annotation*.

Duckworth, P.; Alomari, M.; Gatsoulis, Y.; Hogg, D. C.; and Cohn, A. G. 2016. Unsupervised activity recognition using latent semantic analysis on a mobile robot. In *IOS Press Proceedings*, number 285, 1062–1070.

Freksa, C. 1992. *Using orientation information for qualitative spatial reasoning*. Springer.

Gergely, G.; Bekkering, H.; and Király, I. 2002. Developmental psychology: Rational imitation in preverbal infants. *Nature* 415(6873):755.

Gibson, J. J.; Reed, E. S.; and Jones, R. 1982. *Reasons for realism: Selected essays of James J. Gibson*. Lawrence Erlbaum Associates.

Goldstone, W. 2009. *Unity Game Development Essentials*. Packt Publishing Ltd.

Gullapalli, V. 1990. A stochastic reinforcement learning algorithm for learning real-valued functions. *Neural networks* 3(6):671–692.

Hawes, N.; Burbridge, C.; Jovan, F.; Kunze, L.; Lacerda, B.; Mudrová, L.; Young, J.; Wyatt, J.; Hebesberger, D.; Kortner, T.; et al. 2017. The strands project: Long-term autonomy in everyday environments. *IEEE Robotics & Automation Magazine* 24(3):146–156.

Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780.

Koppula, H. S.; Gupta, R.; and Saxena, A. 2013. Learning human activities and object affordances from rgb-d videos. *The International Journal of Robotics Research* 32(8):951–970.

Krishnaswamy, N., and Pustejovsky, J. 2016. Multimodal semantic simulations of linguistically underspecified motion events. In *Spatial Cognition X: International Conference on Spatial Cognition*. Springer.

Krishnaswamy, N., and Pustejovsky, J. 2017. Do you see what I see? effects of pov on spatial relation specifications. In *Proc. 30th International Workshop on Qualitative Reasoning*.

Krishnaswamy, N. 2017. *Monte-Carlo Simulation Generation Through Operationalization of Spatial Primitives*. Ph.D. Dissertation, Brandeis University.

Paul, R.; Arkin, J.; Roy, N.; and Howard, T. 2017. Grounding abstract spatial concepts for language interaction with robots. In *IJCAI-17 Proceedings*.

Peters, J., and Schaal, S. 2008. Reinforcement learning of motor skills with policy gradients. *Neural networks* 21(4):682–697.

Peters, J. F. 2007. Near sets. Special theory about nearness of objects. *Fundamenta Informaticae* 75(1-4):407–433.

Pustejovsky, J., and Krishnaswamy, N. 2016. VoxML: A visualization modeling language. In Chair), N. C. C.; Choukri, K.; Declerck, T.; Goggi, S.; Grobelnik, M.; Maegaard, B.; Mariani, J.; Mazo, H.; Moreno, A.; Odijk, J.; and Piperidis, S., eds., *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*. Paris, France: European Language Resources Association (ELRA).

Pustejovsky, J., and Moszkowicz, J. 2011. The qualitative spatial dynamics of motion. *The Journal of Spatial Cognition and Computation*.

Pustejovsky, J.; Krishnaswamy, N.; and Do, T. 2017. Object embodiment in a multimodal simulation. *AAAI Spring Symposium: Interactive Multisensory Object Perception for Embodied Agents*.

Pustejovsky, J. 1991. The syntax of event structure. *Cognition* 41(1):47–81.

Pustejovsky, J. 1995. *The Generative Lexicon*. Cambridge, MA: MIT Press.

Randell, D.; Cui, Z.; Cohn, A.; Nebel, B.; Rich, C.; and Swartout, W. 1992. A spatial logic based on regions and connection. In *KR'92. Principles of Knowledge Representa-*

*tion and Reasoning: Proceedings of the Third International Conference*, 165–176. San Mateo: Morgan Kaufmann.

Renz, J., and Nebel, B. 2007. Qualitative spatial reasoning using constraint calculi. In *Handbook of spatial logics*. Springer. 161–215.

Smart, W. D., and Kaelbling, L. P. 2002. Effective reinforcement learning for mobile robots. In *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, volume 4, 3404–3410. IEEE.

Williams, R. J. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8(3-4):229–256.

Yang, Y., and Webb, G. I. 2009. Discretization for naive-bayes learning: managing discretization bias and variance. *Machine learning* 74(1):39–74.

Young, J., and Hawes, N. 2015. Learning by observation using qualitative spatial relations. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, 745–751. International Foundation for Autonomous Agents and Multiagent Systems.

# Learning to Act in Partially Structured Dynamic Environment

**Chen Huang,[1] Lantao Liu,[2] Gaurav Sukhatme[1]**
[1]Department of Computer Science at the University of Southern California, Los Angeles, CA 90089, USA
E-mail: {huan574, gaurav}@usc.edu
[2]Intelligent Systems Engineering Department at Indiana University - Bloomington
Bloomington, IN 47408, USA. E-mail: lantao@iu.edu

## Abstract

We investigate the scenario that a robot needs to reach a designated goal after taking a sequence of appropriate actions in a non-static environment that is partially structured. One application example is to control a marine vehicle to move in the ocean. The ocean environment is dynamic and the ocean waves typically result in strong disturbances that can disturb the vehicle's motion.

Modeling such dynamic environment is non-trivial, and integrating such model in the robotic motion control is particularly difficult. Fortunately, the ocean currents usually form some local patterns (e.g. vortex) and thus the environment is partially structured. The historically observed data can be used to train the robot to learn to interact with the ocean flow disturbances. In this paper we propose a method that applies the deep reinforcement learning framework to learn such partially structured complex disturbances. Our preliminary results show that, by training the robot under artificial and real ocean disturbances, the robot is able to successfully act in complex and spatiotemporal environments.

## Introduction and Related Work

Acting in unstructured environments can be challenging especially when the environment is dynamic and involves continuous states. We study the goal-directed action decision-making problem where a robot's action can be disturbed by environmental disturbances such as the ocean waves or air turbulence.

To be more concrete, consider a scenario where an underwater vehicle navigates across an area of ocean over a period of a few weeks to reach a goal location. Underwater vehicles such as autonomous gliders currently in use can travel long distances but move at speeds comparable to or slower than, typical ocean currents [Wynn et al., Smith et al.]. Moreover, the disturbances caused by ocean eddies oftentimes are complex to be modeled. This is because when we navigate the underwater (or generically aquatic) vehicles, we usually consider long term and long distance missions, and during this process the ocean currents can change significantly, causing spatially and temporally varying disturbances. The ocean currents are not only complex in patterns, but are also strong in tidal forces and can easily perturb the underwater vehicle' motion, causing significantly uncertain action outcomes.

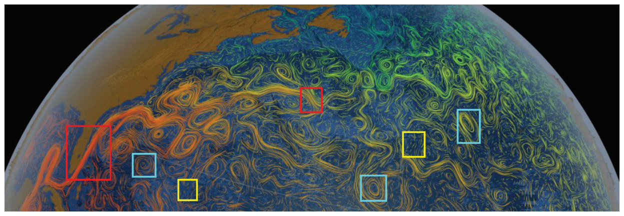


Figure 1: Ocean currents consist of local patterns (source: NASA). Red box: uniform pattern. Blue box: vortex. Yellow box: meandering

In general, such non-static and diverse disturbances are a reflection of the unstructured natural environment, and oftentimes it is very difficult to accurately formulate the complex disturbance dynamics using mathematical models. Fortunately, many disturbances caused by nature are seasonal and can be observed, and the observation data is available for some time horizons. For example, we can get the forecast, nowcast, and hindcast of the weather including the wind (air turbulence) information. Similarly, the ocean currents information can also be obtained, and using such data allows us to train the robot to learn to interact with the ocean currents.

Recently, studies on deep and reinforcement learning have revealed a great potential for addressing complex decision problems such as game playing [Mnih et al., Silver et al., Oroojlooyjadid et al.].

We found that there are certain similarities between our marine robots decision-making and the game playing scenarios if one regards the agent's interacting platform/environment here is the nature instead of a game. However, one general critical challenge that prevents robots from using deep learning is the lack of sufficient training data. Indeed, using robots to collect training data can be extremely costly (e.g., in order to get one set of marine data using on-board sensors, it is not uncommon that a marine vehicle needs to take a few days and traverse hundreds of miles). Also, modeling a vast area of environment can be computationally expensive.

Fortunately, oftentimes the complex-patterned disturbance can be characterized by local patches, where a single patch may possess a particular disturbance pattern (e.g., a vortex/ring pattern) [Oey, Ezer, and Lee], and the total number of the basic patterns are enumerable. Therefore, we are motivated by training the vehicle to learn those local patches/patterns offline so that during the real-time mission, if the disturbance is a mixture of a subset of those learned patterns, the vehicle can take advantage of what it has learned to cope with it easily, thus reducing the computation time for online action prediction and control. We use the iterative linear quadratic regulator [Li and Todorov] to model the vehicle dynamics and control, and use the policy gradient framework [Levine and Koltun] to train the network. We tested our method on simulations with both artificially created dynamic disturbances as well as from a history of ocean current data, and our preliminary results show that the trained robot achieved satisfying performance.

## Technical Approach

We use the deep reinforcement learning framework to model our decision-making problem. Specifically, we use $s$, $a$ to denote the robot's state and action, respectively. The input of the deep network is the disturbance information which is typically a vector field. Our goal is to obtain a stochastic form of policy $\pi_\theta(s,a) = \mathrm{P}(a|s,\theta)$ paramterized by $\theta$ (i.e., weights of the neural network) that maximizes the discounted, cumulative reward $R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$, where $T$ is a horizon term specifying the maximum time steps and $r_t$ is the reward at time $t$ and $\gamma$ is a discounting constant between 0 and 1 that ensures the sum converges. A deep convolutional neural network is used to approximate the optimal action-value function $Q^*(s,a) = \max_\pi \mathrm{E}[R_t|s_t, a_t, \pi]$. More details of the basic model can be found in [Mnih et al.].

### Network Design

Since the ocean currents data over a period is available, we build our neural network with an input that integrates both the ocean (environmental) and the vehicle's states. The environmental state here is a vector field representing the ocean currents (their strengths and directions). Fig. 2 shows the structure of the neural network.

Specifically, the input consists of two components: environment and vehicle states. The environment component has three channels, where the first two channels convey information of the $x$-axis and $y$-axis of the disturbance vector field. Since in the environment we need to define goal states, and there may be obstacles, thus, we use a third channel to capture such information. In greater detail, we assume that each grid of the input map has three forms: it can be occupied by obstacle (we set its value -1), or be free/empty for robot to transit to (with value 0), or be occupied by the robot (with value 1). The other component of the input is a vector that contains vehicle state information, including the vehicle's velocity and its direction towards the goal. Note that we do not include the robot's position in input because we want the robot to be sensitive only to environmental dynamics but not to specific (static) locations.

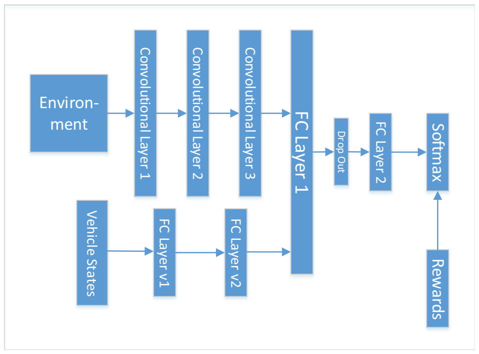


Figure 2: Neural Network Structure

The design of internal hidden layers is depicted in Fig. 2. The front 3 convolutional layers process the environment information, while the vehicle states begin to be combined starting from the first fully connected (FC) layer. The reason of such a design lies in that, the whole net could be regarded as two sub-nets that are not strongly correlated: one sub-net is used to characterize features of disturbances, which is analogous to that of image classification; the other sub-net is a decision component for choosing the best action strategy. In addition, such separation of input can reduce the number of parameters so that the training process can be accelerated.

After each convolutional layer a max-pool is applied. The vehicle states will pass through 2 FC layers, and then are combined with the environmental component output from convolutional layer 3 as the input to a successive FC Layer 1. Between FC Layer 1 and 2 there exists a drop-out layer to avoid overfitting. The Softmax layer is used to normalize outputs for generating a probability distribution that can be used for sampling future actions. Additionally, the *loss funciton* is calculated using this probability distribution as well as the actual rewards.

### Loss Function and Reward

We employ the policy gradient framework for solution convergence. With the stochastic policy $\pi_\theta(s,a)$ and the Q-value $Q_{\pi_\theta}(s,a)$ for the state-action pair, the policy gradient of loss function is $L(\theta)$ can be defined as follows:

$$\nabla_\theta L(\theta) = \mathbb{E}_{\pi_\theta}\big[Q_{\pi_\theta}(s,a)\nabla_\theta \log\pi_\theta(s,a)\big]. \tag{1}$$

To improve the sampling efficiency and accelerate the convergence, we adopt the *importance sampling* strategy using guided samples [Levine and Koltun].

With the objective of reaching the designated goal, our rewarding mechanism is therefore to minimize the cost from start to goal. The main idea is to reinforce with a large positive value for those correct actions that lead to reaching the goal quickly, and punish those undesired actions (e.g., those take long time or even fail to reach the goal) with small or even negative values. Formally, we define the reward $r$ of each trial/episode as:

$$r = \begin{cases} r_s, & \text{succeeded,} \\ -(\alpha r_s + (1-\alpha) r_d), & \text{failed.} \end{cases} \tag{2}$$

where

$$r_s = \frac{1}{\sum_t \pi_\theta(s,a)||p_t - p_G||_2}, \quad (3)$$

$$r_d = 1 - e^{-D_{min}}. \quad (4)$$

where $||p_t - p_G||_2$ denotes the distance from the $t$-th step position to the goal state, and $D_{min} = \min_t ||p_t - p_G||_2$ is the minimum such distance along the whole path. The term $r_s$ in Eq. (3) evaluates the state with respect to the goal state, whereas the term $r_d$ in Eq. (4) summarizes an evaluation over the entire path. Coefficient $\alpha \in [0,1]$ is an empirical value to scale between $r_s$ and $r_d$ so that they contribute about the same to the total reward $r$. In our experiments $\alpha$ is set to 0.9.

## Offline Training and Online Decision-Making

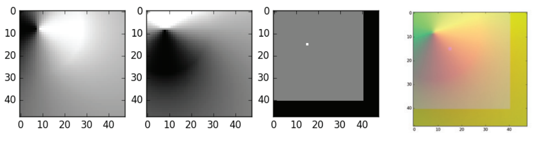
We train the robot by setting different starting and goal positions in the disturbance field, and the *experience replay* [Mnih et al., Riedmiller] mechanism is employed. Specifically, we define an *experience* as a 3-tuple $(s, a, r)$ consisting of state $s$, action $a$, and reward $r$. The idea is to store those experiences obtained in the past into a dataset. Then during the reinforcement learning update process, a mini-batch of experiences is sampled from the dataset each time for training. The process of training is described in Algorithm 1, which can be summarized into four steps.

1. Following incumbent action policies, sample actions and finish a trial path or an episode.
2. Upon completion of each episode, obtain corresponding rewards (a list) according to whether the goal is reached, and assign the rewards to actions taken on that path.
3. Add all these experiences into dataset. If the dataset has exceeded the maximum limit, erase as many as the oldest ones to satisfy the capacity.
4. Sample a mini-batch of experiences from the dataset. This batch should include the most recent path. Then shuffle this batch of data and feed them into the neural network for training. If current round number is less than the max training rounds, go back to step 1.

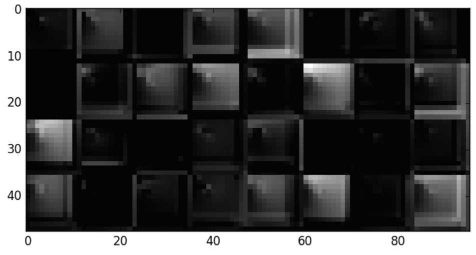
With the offline trained results, the decision-making is straightforward: only one forward propagation of the network with small computational effort is needed. This also allows us to handle continuous motion and unknown states.

**Algorithm 1:** Training

```
$round \leftarrow 0$
while $round < n$ do
    Obtain reward $List\langle s,a \rangle$ of each episode.
    $experiences \leftarrow \varnothing$
    for all $\langle s,a \rangle \in List\langle s,a \rangle$ do
        $r \leftarrow get\_reward(s,a)$
        $experiences \leftarrow experiences \bigcup \langle s,a,r \rangle$
    end for
    $subset \leftarrow experiences$
    pad up $subset$ to batch size with data from dataset
    store $experiences$ into dataset
    shuffle $subset$
    feed $subset$ into neural network
    perform back propagation
    $round \leftarrow round + 1$
end while
```



(a) Input (b) Input(mix)



(c) Convolutional Layer 3

Figure 3: Illustration of disturbance features captured by hidden layer

## Results

We validated the method in the scenario of marine robot goal-driven decision-making, where the ocean disturbances vary both spatially and temporally.The simulation environment was constructed as a two dimensional ocean surface, and the spatiotemporal ocean currents are external disturbances for the robot and are represented as a vector field, with each vector representing the water flow speed captured at a specific moment in a specific location.

The robot used in simulation is a underwater glider with a kinematic motion model with state $z = (x, y, \phi)$ including the vehicle's position and orientation in the world frame, respectively. Since the behavior of the vehicle on the 2D ocean surface is similar to that of the ground mobile robot, thus we opt to use a Dubins car model to simulate its motion. (Similar settings can be found in [Yao, Wang, and Su, Mahmoudian and Woolsey].) The dynamics can be written as:

$$\dot{x} = v\cos\phi, \quad \dot{y} = v\sin\phi, \quad \dot{\phi} = \omega, \quad (5)$$

where control inputs $u = (v, \omega)$ are the vehicle's net speed and turning rate, respectively. The dynamics are obvious nonlinear and in the discrete time case are denoted as $z_{t+1} = f(z_t, u_t)$. Such non-linear control problem can be solved using the iterative Linear Quadratic Regulator (iLQR) [Li and Todorov].

## Network Training

We use Tensorflow [Abadi et al.] to build and train the network described in Fig. 2. In our experiments, the input vec-

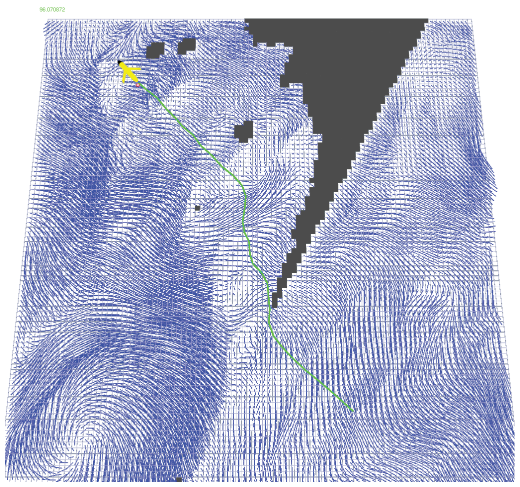


Figure 4: Demonstration of the ocean currents and a path of the robot

tor field map is $48 \times 48$, and the size of dataset for action replay is set to 10000. The learning rate is $1e - 6$, and the batch size we used for each iteration is 500. We also set the length of each episode as 1000 steps.

Fig. 3 shows some features extracted from internal layers of the network. Fig. 3(a) illustrates the feature of a random disturbance vector field. Specifically, the first two channels of Fig. 3(a) are $x$ and $y$ components of the vector field, and the grey-scale color represents the strength of disturbance. The third channel of Fig. 3(a) is a pixel map that contains the goal point (white dot) and obstacle information (black borders).

Other grey grids denote free place. Fig. 3(b) shows a mixed view of the features, with three channels colored in red, green and blue, respectively. The picture depicts a local vortex pattern with the vortex center located near the upper left corner. Fig. 3(c) shows outputs of convolutional layer 3, from which we can observe that the hidden layers extract some local features.

## Evaluations

We implemented two methods: one belongs to the control paradigm and we use the basic iLQR to compute the control inputs; the other one is the deep reinforcement learning (DRL) framework that employs the guided policy mechanism, where the policy is guided by (and combined with) the iLQR solving process [Levine and Koltun].

**Artificial Disturbances** We first investigate the method using artificially generated disturbances. We tested different vector fields including vortex, meandering, uniform, and centripetal patterns.

For different trials, we specify the robot with different start and goal locations, and the *goal reaching rate* is calculated by the times of success divided by total number of simulations.

The results in Table 1 show that within given time limits, both the iLQR and DRL methods lead to a good success rate, and particularly the DRL performs better in complex environments like the vortex field; whereas the iLQR framework has a slightly better performance in relatively mild environments where current speed is low, like the meander disturbance field.

Then, we test the average time costs, as shown in Table 2. The results reveal that the trials using iLQR tend to use less time than those of the DRL method. This can be due to the "idealized" artificial disturbances with simple and accurate patterns, which can be precisely handled by the traditional control methodology.

| Disturbance pattern | Method | Num of trials | Num of success | Success rate |
|---|---|---|---|---|
| Vortex | DRL | 50 | 48 | 0.96 |
| | iLQR | 50 | 46 | 0.92 |
| Meander | DRL | 50 | 49 | 0.98 |
| | iLQR | 50 | 50 | 1.00 |
| Uniform | DRL | 50 | 49 | 0.98 |
| | iLQR | 50 | 48 | 0.96 |
| Centripetal | DRL | 50 | 49 | 0.98 |
| | iLQR | 50 | 48 | 0.96 |

Table 1: Simulation with artificially generated disturbances

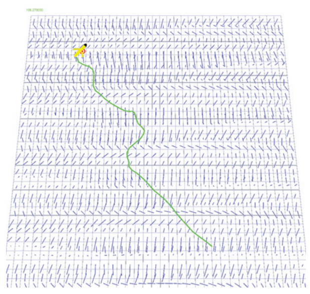
**Ocean Data Disturbances** In this part of evaluation, we use ocean current data obtained from the California Regional Ocean Modeling System (ROMS) [Shchepetkin and McWilliams]. The ocean data along the coast near Los Angeles is released every 6 hours and a window of 30 days of data is maintained and retrievable [Chao].

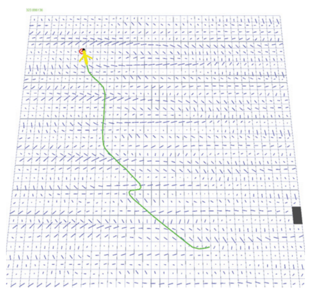
An example of ocean current surface can be visualized in Fig. 4, which also demonstrates a robot's path from executing our training result.

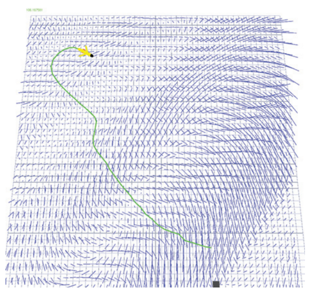
Because the raw ROMS ocean data covers a vast area and practically it requires several days for the robot to travel through the whole space, thus, we randomly cropped local areas to evaluate our training results. Fig. 5 demonstrates a few paths generated in such randomly selected areas.

Similar to the evaluation process for the artificial disturbances, we also looked into those aforementioned performances under the real ocean disturbances. We then evaluate the success rate and time cost, and Table. 3 shows the results (robot speed does not scale to map). Fig. 5 gives a more friendly visualization of those three areas used in our experiments. The results indicate that in most cases the DLR performs better than the basic iLQR strategy.

Fig. 5(c) and 5(d) show scenarios that can be challenging due to strong vortexes. Fig. 5(c) shows that by selecting a good path going around the vortex, the robot successfully reached the goal state. Note, in the area 3 of Fig. 5(d), a very curvy path (e.g., near the goal point) could occur due to some strong vortex in certain local areas. In this example, the ocean current around the goal area has a speed approximately equal to (or even greater than) the robot's maximal speed, but is against the robot's moving direction, so that the robot cannot easily proceed, and both DRL and iLQR eventually failed to reach the goal in this situation. A possible


(a) area1


(b) area2


(c) area3

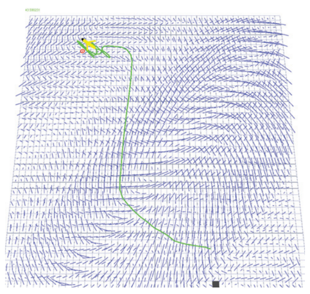

(d) area3

Figure 5: Examples of robot paths under different spatiotemporal disturbance patterns.

| Pattern | Method | Num of trials | Average time cost |
|---|---|---|---|
| Vortex | DRL | 50 | 20.549 |
| | iLQR | 50 | 14.811 |
| Meander | DRL | 50 | 16.926 |
| | iLQR | 50 | 15.367 |
| Uniform | DRL | 50 | 17.667 |
| | iLQR | 50 | 17.803 |
| Centripetal | DRL | 50 | 20.220 |
| | iLQR | 50 | 14.792 |

Table 2: Average time cost under artificial disturbances

| Area | Method | Num of trials | Success rate | Average time cost |
|---|---|---|---|---|
| Area 1 | DRL | 15 | 1.00 | 13.787 |
| | iLQR | 15 | 0.93 | 16.375 |
| Area 2 | DRL | 15 | 1.00 | 14.998 |
| | iLQR | 15 | 1.00 | 15.530 |
| Area 3 | DRL | 15 | 0.60 | 22.875 |
| | iLQR | 15 | 0.80 | 19.546 |

Table 3: Average time cost under ocean disturbances

solution is to manipulate the robot's maximal speed to be larger (this however may be against the reality).

From Table 1 to Table 3, we can conclude that the DRL framework is particularly capable of handling complex and (partially) unstructured environments.

## Conclusions

In this paper we investigate applying the deep reinforcement learning framework for robotic learning and acting in partially-structured environments. We use the scenario of marine vehicle decision-making under spatiotemporal disturbances to demonstrate and validate the framework. We show that the deep network well characterizes local features of varying disturbances. By training the robot under artificial and real ocean disturbances, our simulation results indicate that the robot is able to successfully and efficiently act in complex and partially structured environments.

## References

Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G. S.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Goodfellow, I.; Harp, A.; Irving, G.; Isard, M.; Jia, Y.; Jozefowicz, R.; Kaiser, L.; Kudlur, M.; Levenberg, J.; Mané, D.; Monga, R.; Moore, S.; Murray, D.; Olah, C.; Schuster, M.; Shlens, J.; Steiner, B.; Sutskever, I.; Talwar, K.; Tucker, P.; Vanhoucke, V.; Vasudevan, V.; Viégas, F.; Vinyals, O.; Warden, P.; Wattenberg, M.; Wicke, M.; Yu, Y.; and Zheng, X. 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Chao, Y. 2017. Regional ocean model system. http://www.sccoos.org/data/roms-3km/.

Levine, S., and Koltun, V. 2013. Guided policy search. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, 1–9.

Li, W., and Todorov, E. 2004. Iterative linear quadratic regulator design for nonlinear biological movement systems. In *ICINCO (1)*, 222–229.

Mahmoudian, N., and Woolsey, C. 2008. Underwater glider motion control. In *Decision and Control, 2008. CDC 2008. 47th IEEE Conference on*, 552–557. IEEE.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529–533.

Oey, L.-Y.; Ezer, T.; and Lee, H.-C. 2005. Loop current, rings and related circulation in the gulf of mexico: A review of numerical models and future challenges. *Circulation in the Gulf of Mexico: Observations and models* 31–56.

Oroojlooyjadid, A.; Nazari, M.; Snyder, L. V.; and Takác, M. 2017. A deep q-network for the beer game with partial information. *CoRR* abs/1708.05924.

Riedmiller, M. 2005. Neural fitted q iteration-first experiences with a data efficient neural reinforcement learning method. In *ECML*, volume 3720, 317–328. Springer.

Shchepetkin, A. F., and McWilliams, J. C. 2005. The regional oceanic modeling system (ROMS): a split-explicit, free-surface, topography-following-coordinate oceanic model. *Ocean Modelling* 9(4):347–404.

Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; Chen, Y.; Lillicrap, T.; Hui, F.; Sifre, L.; van den Driessche, G.; Graepel, T.; and Hassabis, D. 2017. Mastering the game of go without human knowledge. *Nature* 550(7676):354–359.

Smith, R. N.; Schwager, M.; Smith, S. L.; Jones, B. H.; Rus, D.; and Sukhatme, G. S. 2011. Persistent ocean monitoring with underwater gliders: Adapting sampling resolution. *Journal of Field Robotics* 28(5):714 – 741.

Wynn, R. B.; Huvenne, V. A.; Bas, T. P. L.; Murton, B. J.; Connelly, D. P.; Bett, B. J.; Ruhl, H. A.; Morris, K. J.; Peakall, J.; Parsons, D. R.; Sumner, E. J.; Darby, S. E.; Dorrell, R. M.; and Hunt, J. E. 2014. Autonomous underwater vehicles (auvs): Their past, present and future contributions to the advancement of marine geoscience. *Marine Geology* 352:451 – 468.

Yao, P.; Wang, H.; and Su, Z. 2015. Uav feasible path planning based on disturbed fluid and trajectory propagation. *Chinese Journal of Aeronautics* 28(4):1163–1177.

# Learning Abstractions by Transferring Abstract Policies to Grounded State Spaces

**Lawson L. S. Wong**
Department of Computer Science, Brown University
Providence, RI 02912, USA
lsw@brown.edu

## Abstract

Learning from demonstration is an effective paradigm to teach specific tasks to robots. However, such demonstrations often have to be performed on the robot, which is both time-consuming and often still requires expert knowledge (e.g., kinesthetically controlling the joints). It is often easier to specify tasks at a high level of abstraction, and let the robot figure out the grounding to the robot/agent space. We consider how to learn such a mapping. In particular, we consider the task of learning to navigate on a mobile robot given only an abstraction of the path and potential landmarks. We cast this as a learning problem between abstract and robot (grounded) state spaces and illustrate how this works in several cases. Through these cases, we see that the "abstract navigation" task touches on many interesting issues related to abstraction, and suggest avenues for further investigation.

## Introduction

To tackle the high-dimensional complexity of the world and long-horizon nature of complex tasks, agents need *abstraction*, the act of compressing both state and time in service of certain goals. Much of artificial intelligence has been devoted to manually endowing agents with abstractions, such as via symbols (state abstraction) (Dietterich 2000; Konidaris, Kaelbling, and Lozano-Pérez 2018) and subtasks/options (temporal abstraction) (Sutton, Precup, and Singh 1999). However, agents that operate in a continual and lifelong setting will eventually encounter conditions unforeseen to the designer, and must come up with its own abstractions. Existing work in learning abstractions, most notably in reinforcement learning, typically require much experience within the domain, and arguably have not achieved widespread success. Indeed, one of the challenging aspects of abstraction is that in the time it takes to induce an abstraction and learn how to use it effectively, the specific ground / non-abstract task could already have been solved.

In contrast, humans use abstractions very effectively. For example, when provided a 2-D map of a new location (e.g., Figure 1), people can typically follow the map to reach a desired destination on the first try, without requiring the numerous episodes of trial and error that reinforcement learners require. This feat is even more remarkable when considering that the real-world looks nothing like the 2-D map: it is 3-D, is perceived from a first-person perspective (instead of bird's-eye for maps), and contains many more objects and other distractors compared to the map itself. Even so, when encountering these completely new percepts and 'states', people can follow where they are on the map and navigate as desired. Humans have mastered the abstraction of 2-D maps: from the current *ground* state in the real world, they are able to find the corresponding *abstract* state as a 2-D point on the map, determine the appropriate next *abstract* action within the abstract world, and then *ground* this action into physical motion. Furthermore, humans have mastered the entire *class* of such 2-D map abstractions; given a *new* instance of the abstraction (e.g, a map of a new place), humans can immediately perform the necessary grounding.

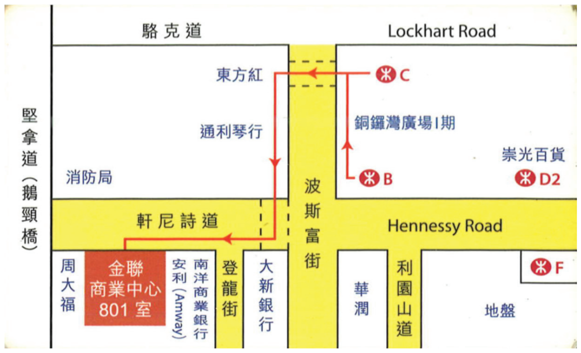


Figure 1: Humans can navigate in new places by using abstract 2-D maps, such as by following the walking directions depicted by the red arrows in the map above, which direct a person to exit a certain subway exit and cross two roads to reach an office (red square in the bottom left). They are able to take *abstract* policy-related knowledge encoded in the map, and *ground* the relevant actions in the real world. If robotic agents can learn to use existing abstractions, not only will they be easier to instruct by humans, they may even be able to produce abstract and interpretable knowledge.

We first formalize the notion of abstraction, then frame the problem of learning how to use existing abstractions as a fully supervised, learning-from-demonstration problem. For the "abstract navigation" task described above, we consider several classes of possible abstractions, some of which are

$$\begin{tikzcd} S \arrow[r, "\pi"] \arrow[d, "f_s"'] & A^+ \\ \widetilde{S} \arrow[r, "\tilde{\pi}"] & \widetilde{A} \arrow[u, "f_a"'] \end{tikzcd}$$

Figure 2: Abstraction diagram.

easy to learn, whereas others remain unsolved. Finally, we discuss directions of ongoing and future investigation.

## Related Work

The general problem setup has ties to transfer learning (Taylor and Stone 2009) and learning from demonstration (Argall et al. 2009). Cobo et al. (Cobo et al. 2014) also explored learning abstractions from demonstrations, using an approach based on feature selection and task decomposition.

The formulation of abstractions in this work is inspired by the pioneering work of Ravindran (Ravindran 2004) and subsequent work by Abel et al. (Abel, Hershkowitz, and Littman 2016). Both lines of work analyze theoretical properties of abstraction in reinforcement learning.

Recently, there has been work on navigation using abstract 2-D maps such as hand-sketched maps (Boniardi et al. 2015; 2016), floor plans (Gao et al. 2017), and mazes (Brunner et al. 2018). However, most of these approaches are specific to 2-D robot/agent navigation.

## Model and Problem Formulation

The agent operates in the grounded state space $S$ and action space $A$. The objective is to determine a plan or policy $\pi : S \to A$ that achieves some given task in the world. The premise of this work is that we are given an abstract solution for the task, such as a route to follow on an abstract 2-D map that reaches a desired goal location. Formally, we are given an abstraction in abstract state and action spaces $\widetilde{S}, \widetilde{A}$, as well as an abstract policy $\tilde{\pi} : \widetilde{S} \to \widetilde{A}$.

The abstract policy is a solution for the (grounded) task if there exist *abstraction functions* $f_s : S \to \widetilde{S}$ and $f_a : \widetilde{A} \to A^+$ that can produce a grounded policy according to the diagram in Figure 2. In particular, the requirement is:

$$\pi = f_a \circ \tilde{\pi} \circ f_s \qquad (1)$$

To find the next ground action(s), we first lift the ground state to the abstract state using $f_s$, apply the given abstract policy $\tilde{\pi}$, then ground the resulting abstract action using $f_a$ to an executable primitive action (or action sequence, if there is temporal abstraction). If $\tilde{\pi}$ is an abstract solution for the task, then repeatedly applying this procedure should result in the agent reaching the goal in its grounded space.

Our goal is to *learn* the abstraction functions $f_s$ and $f_a$, such that when presented with a new instance of the abstraction class (e.g., a 2-D map of a new location), the agent can follow the given abstract solution via Equation 1, i.e., transfer an abstract policy to the agent's grounded state space.

To learn the abstraction functions, we need training data. We consider the simplest setting, where paired trajectories in both ground and abstract spaces are provided. This is a fully-supervised, learning-from-demonstration setting, where the agent is shown grounded solutions to various task instances (e.g., by guiding it through the real world), together with annotated abstract solutions to the same problems (e.g., by drawing the route on the 2-D map).

## Abstract Navigation

We consider several instances of a problem where the task is to follow a specified path, given in an abstract space. The grounded state space in all these cases is the state of the robotic agent, which includes highly relevant state dimensions such as odometry (noisy estimate of location relative to its starting position), moderately relevant features such as detected landmarks, and irrelevant features such as its arms' joint angles (if it has arms) or its battery level.

### Isometric path

In the simplest case, the abstract path is given as a 2-D trajectory that accurately preserves relative lengths and angles, except possibly in a different global coordinate frame and scale. (This would be the case if the path was specified in most popular web mapping services such as Google Maps.) If the abstract path is also annotated at each point with the appropriate ground action, which could also be easily inferred from an isometric 2-D solution trajectory, then $f_a$ can be assumed to be the identity function. The paired trajectories during training give corresponding pairs $(s, \tilde{s})$ of high-dimensional ground states and 2-D abstract states respectively. Learning $f_s$ then becomes a multi-label linear regression problem (mapping $s$ to $\tilde{s}$), since the ground and abstract states are related via an affine transformation (in the case of an isometric abstract path). In simulation, this method alone is highly effective at ignoring irrelevant features in the ground state $s$ and handling zero-mean additive noise.

For abstract paths that are not perfect isometries, we need to learn non-linear regression functions. This is still strictly within the realm of supervised machine learning, for which many approaches exist to learn non-linear $f_s$ functions.

### The issue of orientation

The previous case provided a way to accurately find the abstract $(x, y)$ location on the provided abstract path. However, the first problem one encounters when implementing the strategy on a point robot is orientation: if the robot is not facing in the same direction as the path intended, then following the abstract policy $\tilde{\pi}$ causes the robot to deviate from the path. The main issue is that the abstraction is insufficient to distinguish between the canonical path-following orientation from other states sharing the same abstract $(x, y)$.

There are several potential ways to fix this. The simplest is to expand the abstract space to incorporate orientation $\theta$ as well; however, this requires a more complicated abstract policy to be specified. Alternatively, the burden may be placed on the agent, by formulating each step of the path-following as a subtask (instead of a primitive action), where the subgoal is to return to a canonical orientation. The canonical orientation can be learned during training, or may be required to be the initial heading of the robot.

More generally, incomplete abstractions are likely to be encountered, and it would be useful to detect them and make local corrections, such as by inserting subgoals. This points to one argument for learning both ground and abstract transition models, $T$ and $\widetilde{T}$ respectively: an incomplete abstraction will not generally be able to enforce one-step consistency between $f_s \circ T$ and $\widetilde{T} \circ f_s$. Thus transition models enable error detection in abstraction.

### Landmarks

In typical maps, even in the case that the map is an isometry, there are additional features such as street names, room numbers, and other iconic elements such as architecturally distinct buildings. For example, in Figure 1, various street names (black font) and store names (blue font) are given near their respective locations. As humans, our sense of odometry is likely worse than mobile robots, so we must rely on these highly distinguishable landmark cues for robustness. If the robot is provided with detectors that allow it to detect landmark features, then these detections can simply be incorporated as additional ground state dimensions, and we can proceed to learn $f_s$ from demonstrations via nonlinear regression. In simulation, we considered landmarks in the form of 'color patches' encountered in local regions of the world; if the color is confined to a unique region in the abstract space, these landmarks are highly informative and can correct for otherwise inaccurate geometric mappings.

### Topological path

In the previous case, landmarks provide information that is redundant with the geometric abstract map. Hence it is possible to remove the geometric aspects of the abstraction and simply retain the topological information provided by landmarks. A path in the space of landmarks can now be represented as a deterministic finite automaton; for example, in the case of street names as landmarks, nodes may correspond to streets, and edges with street intersections (with an appropriate output ground action to perform the correct turn, if any). Note that this abstraction only allows specifying a single action to be repeatedly performed between two landmarks; for example, when on a certain street, the agent can only move in one direction on the street, until an intersection is encountered. In this case, uniqueness of landmarks is essential, since they are the only source of information, unless transition models are also provided to enable tracking.

### Richer abstractions

The initial motivation for the abstract mapping task was to follow an abstract 2-D map, such as the one in Figure 1. Ultimately, these maps are typically perceived via vision, and it would be much easier for a robot to use existing maps if it can process them in image form, rather than requiring a manual encoding of the abstract policy $\tilde{\pi}$. Compared to previous cases, using the 2-D map in image form is interesting because it is both featurally richer compared to previous abstractions, while at the same time still much lower-dimensional with respect to the robot. One possibility for using this image-based abstraction is to extract features from it, such as using convolutional neural networks, and to then learn to map ground states to abstract visual features.

The automaton-based abstraction in the previous case is also closely related to using natural language instructions for navigation. For example, "go straight on street A for two blocks until the intersection with street B, then turn left" can be represented as an automaton. We can therefore consider using natural language itself as an abstraction, either by mapping the sequence of instructions to an automaton, or by directly mapping ground states to abstract linguistic features, as in the case for images.

## Discussion

We considered the problem of learning to use existing abstractions in novel environments, in the context of the problem of navigation using abstract 2-D maps. The problem was formulated as a fully-supervised, learning from demonstration problem, and several cases of potential abstraction classes were considered. In the process of analyzing these cases, various aspects and issues of abstraction were encountered, and many problems and solutions still lie ahead.

There remains the issue of learning the action abstraction function $f_a$. This is the problem of temporal abstraction, which has arguably received greater attention in the field thus far. One way to consider an abstract action $\tilde{a}$ is to view it as a subgoal, which instantiates a local planning problem. This was a potential strategy used to overcome the lack of orientation information in the abstract 2-D map.

So far, the problem has only been considered in the fully-supervised setting. Although this provides the strongest signal for learning, it also requires significant effort from the user. One possibility is provide weak supervision through reinforcement learning, in the extreme case only providing reward if the correct path is followed. An intermediate regime would be to still provide demonstrations, but no longer with ground-abstract state correspondences.

Two cases in the previous section touched upon the utility of learning transition models. The benefit so far appears to be increased robustness in determining the correct abstract state. Transition models are also needed for planning; if the solution path is not provided, and only an abstract map is given (which is the case when using a standard map), then planning in the abstract space will be necessary. This is a useful extension to the problem considered so far: find an abstract policy and follow it via the same grounding mechanism, with the assumption that the abstraction is a "faithful" representation of the world with respect to the task.

The proposed approach may also provide useful theoretical analysis of abstractions. Since the problem of learning abstractions has been transformed into one of supervised learning, we may be able to adapt theoretical tools from computational learning theory in this more familiar setting, and characterize the utility of various abstractions. In particular, to use the given abstraction effectively, we had to learn the abstraction function $f_s$ (and eventually $f_a$); the complexity of this learning problem tells us how practical the abstraction is. If it is difficult to learn $f_s$, then it may not be worth the extra learning effort for the potential reduction in representational complexity. An abstraction may only be useful if it is an accurate representation of the world with respect to the task, provides some degree of information compression, *and the abstraction functions are easy to learn*.

## References

Abel, D.; Hershkowitz, D.; and Littman, M. 2016. Near optimal behavior via approximate state abstraction. In *International Conference on Machine Learning*.

Argall, B.; Chernova, S.; Veloso, M.; and Browning, B. 2009. A survey of robot learning from demonstration. *Robotics and Autonomous Systems* 57(5):469–483.

Boniardi, F.; Behzadian, B.; Burgard, W.; and Tipaldi, G. 2015. Robot navigation in hand-drawn sketched maps. In *European Conference on Mobile Robots*.

Boniardi, F.; Valada, A.; Burgard, W.; and Tipaldi, G. 2016. Autonomous indoor robot navigation using a sketch interface for drawing maps and routes. In *IEEE International Conference on Robotics and Automation*.

Brunner, G.; Richter, O.; Wang, Y.; and Wattenhofer, R. 2018. Teaching a Machine to Read Maps with Deep Reinforcement Learning. In *AAAI Conference on Artificial Intelligence*.

Cobo, L.; Subramanian, K.; Isbell, C.; Lanterman, A.; and Thomaz, A. 2014. Abstraction from demonstration for efficient reinforcement learning in high-dimensional domains. *Artificial Intelligence* 216:103–128.

Dietterich, T. 2000. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research* 13:227–303.

Gao, W.; Hsu, D.; Lee, W.; Shen, S.; and Subramanian, K. 2017. Intention-net: Integrating planning and deep learning for goal-directed autonomous navigation. In *Conference on Robot Learning*.

Konidaris, G.; Kaelbling, L.; and Lozano-Pérez, T. 2018. From skills to symbols: Learning symbolic representations for abstract high-level planning. *Journal of Artificial Intelligence Research* 61:215–289.

Ravindran, B. 2004. *An Algebraic Approach to Abstraction in Reinforcement Learning*. Ph.D. Dissertation, University of Massachusetts Amherst.

Sutton, R.; Precup, D.; and Singh, S. 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence* 112(1):181–211.

Taylor, M., and Stone, P. 2009. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research* 10(1):1633–1685.

# Represention, Use, and Acquisition of Affordances in Cognitive Systems

**Pat Langley**
Institute for the Study of Learning and Expertise
2164 Staunton Court, Palo Alto, CA 94306 USA

**Mohan Sridharan,Ben Meadows**
Department of Electrical & Computer Engineering
University of Auckland, Private Bag 92019
Auckland 1142 New Zealand

**Abstract**

We review the psychological notion of *affordances* and examine it anew from a cognitive systems perspective. We distinguish between environmental affordances and their internal representation, choosing to focus on the latter. We consider issues that arise in representing mental affordances, using them to understand and generate plans, and learning them from experience. In each case, we present theoretical claims that, together, form an incipient theory of affordance in cognitive systems. We close by noting related research and proposing directions for future work in this arena.

## 1 Introduction and Background

Intelligent agents, both human and artificial, often operate in the context of an external environment and interact with entities therein. The agent can interact effectively with these objects in some ways but not others. For instance, depending on its manipulators, an agent will be able to grasp, lift, or throw some items but not different ones. Similarly, it can sit or recline on some objects but not others. Gibson (1977) referred to such relationships as *affordances*, a term that has been widely adopted in perceptual psychology, human-computer interaction, and, more recently, AI and robotics.

Gibson viewed affordances as existing in the environment, but others have used the term, rather differently, to refer to internalized models of these relations. For example, Vera and Simon (1993) have proposed that they are encoded as symbol structures which the agent can use to guide its decision making. They mapped affordances onto both the condition sides of production rules and onto perceptual chunks to which they refer. More recently, Sahin et al. (2007) and Zech et al. (2017) have reviewed different formalizations in robotics, focusing on relations between agents and the environment. We will incorporate ideas from each of these earlier efforts in our own analysis.

In this paper we present a high-level theory of affordances that makes commitments about a number of key issues. Like Vera and Simon, we focus on internal representations of affordances that describe an agent's ability for action. However, we move beyond their treatment to make more specific statements about the role of affordances in intelligence, focusing in turn on issues of representation, performance, and learning. We propose theoretical postulates about affordances that we feel are promising, but we do not report implemented agents that incorporate these tenets or experimental evaluations of them, which we reserve for future work.

## 2 Representing Knowledge of Affordances

Because representation constrains both performance and learning, we should address first how an intelligent agent can encode affordances in memory and how they relate to other cognitive structures. We distinguish between grounded short-term elements, say a belief that the agent can lift a particular box, and generic long-term ones, say a predicate and associated rule that specifies the class of situations in which lifting is possible. The typical usage of 'affordance' focuses on the grounded version, but we maintain that such elements are always instances of generic structures, so the primary representational challenges concern encoding the latter.

We hypothesize two distinct forms of knowledge: *concepts* that denote classes of objects or relations among them; and *skills* that specify the conditions in which multi-step activities produce specific outcomes.[1] Skills refer to concepts when describing their conditions and effects, making the latter structures more basic than the former. This leads naturally to our first theoretical postulate:

- *Affordances are concepts that describe the class of situations and the characteristics of agents for which particular activities produce specific effects.*

In other words, they are reified predicates that link the structures of objects and the features of agents that can use those objects to achieve given ends. Affordances take the same form as other concepts, in that they specify a predicate with associated arguments and a set of conditions that describe when they hold. The key difference is that each affordance concept serves as the sole condition on a skill, indicating when the latter produces its associated effects. Conceptual memory also contains other concepts, such as ones that describe situations which result from a skill's application.

Note that we view affordances as three-way relationships among the way an object is used, structural aspects of that

[1] We have borrowed this disctintion from Li, Stacuzzi, and Langley's (2012) ICARUS architecture, but it has roots in psychology.

object, and characteristics of the agent that uses it. A typical hammer has a handle with a head on one end, but it cannot be used to drive a nail or spike unless the agent is strong enough to lift and swing it. This means that a sledge hammer may afford the hammering activity for some agents but not others. Some conditions in an affordance concept will be qualitative, but others will specify numeric relations, such as whether a tool's weight is less than what the agent can lift.

We also postulate that many affordances are matters of degree. Some handles are easier for a given agent to grasp than others, while some ladders are easier for that agent to climb. This suggests that logical definitions of concepts, often assumed in AI, are insufficient. Instead, we propose that:

- *Affordances are graded concepts that match situations to greater or lesser degrees.*

For instance, a hammer may be more or less usable by a person depending on the difference between its weight and what he can lift, among other factors. Probabilistic categories are one way to support graded behavior, but any approach that measures distance from a prototype or central tendency will suffice. Most work in this tradition has assumed attribute-value notations, but one can also define relational concepts that match to different degrees (e.g., Choi 2010).

Finally, treating affordances as reified conceptual predicates suggests another representational characteristic that, we hypothesize, is especially important for describing extended activities that involve multiple steps:

- *Complex affordances are decomposable into elements that denote different aspects of usability.*

For example, a tool has a hammering affordance when an agent can grasp its handle, lift it upward, and propel its flat head against the target. We can view each of these elements as a distinct 'subaffordance' that must hold, for a given agent and to a reasonable degree, to let the agent use a tool for its intended function. A hammer may be light enough for a person to lift, but it will not drive home a nail if its handle is so slippery that it flies out of his grasp or if its head is so narrow that it misses the target.

## 3 Using Knowledge of Affordances

Humans and other intelligent agents engage in two broad classes of knowledge-based cognition. One involves interpreting situations and events in the environment, in some cases the activities of other agents. For instance, we may observe someone stacking some boxes but appear to have difficulty lifting one that is too heavy. The simplest variant is intention recognition, which assigns an agent's behavior to some known category, such as picking up a hammer or stacking a box. A more complex version, plan understanding (e.g., Meadows et al. 2014), infers an agent's multi-step plan, including goals it aims to achieve. Our next claim involves two facets of this performance task:

- *Affordances enable both proposal of hypotheses during plan understanding and their evaluation.*

To clarify hypothesis creation, suppose that we observe someone holding a nail and reaching in the direction of two objects, a hatchet and a screwdriver. The hatchet's structure, specifically its handle and the flat side of its head, can be used to hammer the nail, suggesting this as a candidate intention. The latter occurs because the hatchet's description, obtained through perception and inference, matches the affordance conditions associated with hammering a nail. The screwdriver does not lend itself structurally to this activity, so it would not produce a comparable hypothesis.

The graded nature of affordances helps during evaluation of candidate explanations. Given a set of observations, some intentions and plans will be more plausible than others. For example, suppose we observe someone in a room picking up a shoe that has a flat heel. We might hypothesize that he plans to put the object on his foot or that he plans to use it to hammer a nail. The shoe can be used for both activities, but it matches the affordance concept for placing on a foot much better than it does the one for hammering. We can use this degree of match in our evaluation of the two hypotheses and conclude that the first alternative is more plausible.

The second performance task concerns generating activities that support one's goals. As before, the simplest cases involve selection of primitive actions, such as grasping a glass or lifting a held box. More complicated variants involve chaining sequences of actions into multi-step plans to achieve the agent's goals. This suggests another tenet:

- *Affordances aid both the proposal of actions during plan generation and their evaluation.*

For instance, suppose we want a nail embedded in a wall and we have two tools, a hatchet and a screwdriver. We might use means-ends analysis to propose a hammering activity that achieves the goal and then realize the hatchet, held in a particular orientation, satisfies the affordance concept for hammering, but the screwdriver does not. Or we might use forward chaining to identify which affordances match the current situation, retrieve their associated activities, and consider the resulting states. Hammering the nail with the reversed hatchet is an applicable action that achieves the goal, but no screwdriver-related activities are applicable. If the nail were a screw, the situation would be inverted.

Affordances can also influence evaluation of candidate intentions during the planning process. Suppose, again, that we want a nail embedded in the wall, and that we have generated two possible intentions: hammering the nail with a reversed hatchet and hammering it with a shoe. Both satisfy the relational conditions of the graded affordance for hammering, but the hatchet would match its specification better than the shoe. The reasons involve both the relative abilities for grasping the two tools and their capacities for driving the nail into the wall even when they are held firmly.

## 4 Acquiring Knowledge of Affordances

Now that we have discussed the representation and use of internal affordances, we can turn briefly to their acquistion from experience. Recall that affordance concepts describe the conditions under which an activity has a particular effect for an agent. The AI community has pursued two different approaches to learning about agents' activities that suggest a final theoretical postulate:

- *Primitive affordances are learned inductively whereas complex affordances are learned analytically.*

When an agent first interacts with a new object or situation, it has little knowledge on which to build. In response, learning the conditions under which an action will have desired effects – the affordance concept – is primarily empirical. For example, this can occur by attempting to grasp different objects, with induction comparing configurations of successful and unsuccesful cases (e.g., Shen and Simon 1989).

In contrast, acquisition of complex affordances occurs in the presence of existing components, enabling use of analytic methods like those used to determine conditions on macro-operators (Iba 1989). This involves composing the conditions of actions not satisfied by the effects of those that occur before them. For instance, if we have affordance concepts for grasping a hammer's handle, lifting it, and hitting a nail with its head, then each of these would appear as components of a complex affordance for hammering a nail. Interactions among these elements may require inductive refinement, but creation of an initial concept can occur analytically based on a single training case. Li et al. (2012) have adapted this compositional method to acquire definitions for new conceptual predicates, in some cases recursive ones, that serve as conditions on learned hierarchical skillls.

## 5 Related Research

Recent years have seen growing interest in internalized affordances within the AI and robotics communities. Horton, Chakraborty, and St. Amant (2012) review many of these efforts, which often use visual processing to classify objects as appropriate for actions. Sahin et al. (2007) and Zech et al. (2017) also offer insightful surveys of computational research on the topic. We should examine how our theoretical claims relate to the growing body of work in this area.

- *Affordances are concepts that map relations between situations and agents on the effects of actions.*

A review of the literature reveals that some aspects of this statement are widely accepted but not others. Treatments of affordances have always involved mapping objects or situations onto action relevance, and many efforts to learn such mappings produce conceptual descriptions or classifiers. However, the notion that affordances involve *interactions* between features of agents and features of objects has been much less common. Stoffregen (2003) provides an early and clear statement of this claim, but his treatment was informal and, to our knowledge, AI and robotics papers have only rarely incorporated his insight. We maintain that this important idea deserves more attention in the computational literature than it has received.

- *Affordances are graded concepts that match situations to greater or lesser degrees.*

Prior researchers have not discussed this idea directly. For instance, Sarathy and Scheutz (2016) describe an approach that uses probabilistic rules to infer affordances of objects for actions. Their framework shares our assumption that affordances are reified concepts, but not that these mental structures are graded. Zech et al. (2017) consider dynamic affordances that vary with changing properties of objects, but they remain Boolean in each case. They also suggest that agents choose among objects based on appropriateness to a given outcome, but stop short of proposing degrees of affordance. Of course, probabilistic approaches can predict how features of the agent and situation affect an action's chance of success, but graded affordances can also encode the time, effort, and difficulty of achieving an objective. Thus, this claim seems like an important contribution to the literature.

- *Complex affordances are decomposable into elements that denote different aspects of usability.*

This idea appears in a few places but has not been explored in detail. Zech et al. review a few papers that discuss a hierarchy of affordances, including Ellis and Tucker's (2000) experimental studies of 'micro-affordances' as 'potentiated components' of higher-level activities (e.g., turning a wrist while reaching for an object). However, computational researchers have generally focused on a single level of analysis. Therefore, the decomposition of complex affordances into simpler elements, and the compositional semantics it requires, is a notion that merits substantially more effort than the community has given it to date.

- *Affordances enable the proposal and evaluation of hypotheses during plan understanding.*

This theoretical tenet is both uncontroversial and supported in the literature, although few publications state it in these terms. For instance, Sindlar and Meyer (2010) report a system that uses logical reasoning about affordances to generate hypotheses about a BDI agent's intentions in a video game, but also uses numeric scores to evaluate them. In contrast, Freedman, Jung, and Zilberstein (2015) describe a probabilistic approach that ranks all candidate activities, using information about tool affordances for evaluation but not hypothesis generation. We encourage researchers who work in this area to be more explicit about the ways in which affordances guide their systems' decision making.

- *Affordances aid the proposal and evaluation of actions during plan generation.*

This postulate is also supported by publications in the area. One example comes from Ugur, Oztop, and Sahin (2011), who use learned object affordances during planning to propose candidate actions whose conditions match the current state, but not to evaluate them. In contrast, Boularias et al. (2015) use information about affordances, acquired by reinforcement learning, to evaluate alternative actions by comparing the values expected from their application.

- *Primitive affordances are learned inductively whereas complex affordances are learned analytically.*

Nearly all computational research in this arena has focused on acquiring primitive affordances and has relied exclusively on inductive methods, which is consistent with the first half of our claim. For instance, Kjellström, Romero, and Kragić (2010) describe a statistical approach to learning primitive affordances from observation for use in activity recognition, whereas Ugur et al. (2011) learn action models from exploration that map continuous features of objects to effect categories. Similarly, Boularias et al. (2015) report a system that estimates the expected values of actions in different situations, which they view as affordances, from delayed rewards. More interesting is recent work by Sridharan, Meadows, and Gomez (2017) that learns primitive affordances inductively and then combines them analytically into composite affordances on finding that sequences of actions achieve the agent's goals. However, this is the only work we have found that addresses the second half of our final tenet.

In summary, a number of theoretical claims about affordances appear to be novel, while others have received little attention. Taken together, they offer a new perspective that can drive work on embodied agents in interesting directions.

## 6 Concluding Remarks

In the preceding pages, we presented an account of affordances in intelligent systems. Our theory postulated these structures are reified concepts that specify when skills have particular effects for given agents, that allow graded membership, and that can be composed from more basic affordances. An intelligent system can use such structures to hypothesize and evaluate candidate plans that help understand others' behavior and achieve its own goals. Finally, such an agent can acquire affordance concepts from experience through a mixture of inductive and analytic learning mechanisms. We saw that others have explored some of these ideas, but that some appear novel, and there is no existing account of affordances that combines them into a unified theory.

In future research, we should incorporate these ideas into an implemented system, ideally an existing agent architecture that makes assumptions which are largely consistent with the new postulates (e.g., Li et al. 2012). We should also demonstrate the extended architecture on scenarios that illustrate the representation, use, and acquisition of graded, composite affordances for agents with different abilities. Finally, we should carry out experiments that test the benefits of affordance-driven processing over alternative approaches to intelligent systems. If studies reveal that this leads to better explanations, more effective plans, and reduced search, they will serve as evidence that supports the theory.

## Acknowledgements

This research was supported by Grant N00014-17-1-2434 from ONR, which is not responsible for its contents.

## References

Boularias, A.; Bagnell, J.; and Stentz, A. 2015. Learning to manipulate unknown objects in clutter by reinforcement. *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 1336-1342. Austin, TX: AAAI Press.

Choi, D. 2010. Nomination and prioritization of goals in a cognitive architecture. *Proceedings of the Tenth International Conference on Cognitive Modeling*, 25–30. Philadelphia, PA.

Ellis, R.; and Tucker, M. 2000. Micro-affordance: The potentiation of components of action by seen objects. *British Journal of Psychology* 91:451–471.

Freedman, R. G.; Jung, H.-T.; and Zilberstein, S. 2015. Temporal and object relations in unsupervised plan and activity recognition. *Proceedings of AAAI Fall Symposium on AI for Human-Robot Interaction*, 51–59. Arlington, VA: AAAI Press.

Horton, T. E.; Chakraborty, A.; and St. Amant, R. 2012. Affordances for robots: A brief survey. *Avant* 3:71–84.

Iba, G. A. 1989. A heuristic approach to the discovery of macro-operators. *Machine Learning* 3:285–317.

Gibson, J. J. 1977. The theory of affordances. In R. E. Shaw and J. Bransford (Eds.), *Perceiving, acting, and knowing*. Hillsdale, NJ: Lawrence Erlbaum.

Kjellström, H.; Romero, J.; and Kragić, D. 2011. Visual object-action recognition: Inferring object affordances from human demonstration. *Computer Vision and Image Understanding* 115:81–90.

Li, N.; Stracuzzi, D. J.; and Langley, P. 2012. Improving acquisition of teleoreactive logic programs through representation extension. *Advances in Cognitive Systems* 1:109–126.

Meadows, B.; Langley, P.; and Emery, M. 2014. An abductive approach to understanding social interactions. *Advances in Cognitive Systems* 3:87–106.

Sahin, E.; Cakmak, M.; Dogar, M. R.; Ugur, E.; and Ucoluk, G. 2007. To afford or not to afford: A new formalization of affordances toward affordance-based robot control. *Adaptive Behavior* 15:447–472.

Sarathy, V.; and Scheutz, M. 2016. A logic-based computational framework for inferring cognitive affordances. *IEEE Transactions on Cognitive and Developmental Systems*, *8*.

Shen, W-M.; and Simon, H. A. 1989. Rule creation and rule learning through environmental exploration. *Proceedings of the Eleventh International Joint Conference on Artificial intelligence*, 675–680. Detroit: Morgan Kaufmann.

Sindlar, M.; and Meyer, J.-J. 2010. Affordance-based intention recognition in virtual spatial environments. *Proceedings of the Thirteenth International Conference on Principles and Practice of Multi-Agent Systems*, 304–319. Kolkata, India.

Sridharan, M.; Meadows, B; Gomez, R. 2017. What can I not do? Towards an architecture for reasoning about and learning affordances. *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling*, 461–469. Pittsburgh, PA: AAAI Press.

Stoffregen, T. A. 2003. Affordances as properties of the animal-environment system. *Ecological Psychology* 15: 115–134.

Ugur, E.; Oztop, E.; and Sahin, E. 2011. Goal emulation and planning in perceptual space using learned affordances. *Robotics and Autonomous Systems* 59:580–595.

Vera, A.; and Simon, H. A. 1993. Situated action: A symbolic interpretation. *Cognitive Science* 17:7-48.

Zech, P.; Haller, S.; Lakani, S. R.; Ridgeand, B.; Ugur, E.; and Piater, J. 2017. Computational models of affordance in robotics: A taxonomy and systematic classification. *Adaptive Behavior* 25:235–271.

# Flexible Goal-Directed Agents' Behavior via DALI MASs and ASP Modules

**Stefania Costantini, Giovanni De Gasperis**
Dip. di Ingegneria e Scienze dell'Informazione e Matematica (DISIM),
Università di L'Aquila, Coppito I-67100, L'Aquila, Italy
email: {stefania.costantini, giovanni.degasperis}@univaq.it

**Abstract**

This paper describes the architecture that integrates DALI MASs (Multi-Agent Systems) and ASP (Answer Set Programming) modules for reaching goals in a flexible and timely way, where DALI is a computational-logic-based fully implemented agent-oriented logic programming language and ASP modules includes solvers that allow affordable and flexible planning capabilities. The proposed DALI MAS architecture exploits such modules for flexible goal decomposition and planning, with the possibility to select plans according to a suite of possible preferences and to re-plan upon need. We present an abstract case-study concerning DALI agents which cooperate for exploring an unknown territory under changing circumstances in an optimal or at least suboptimal fashion. The architecture can be exploited not only by DALI agents, but rather by any kind of logical agent.

## Introduction

Adaptive autonomous agents are capable of adapting to partially unknown and potentially changing environments (Knudson and Tumer 2011), (Jiming 2001). This requires agents to be capable of various forms of commonsense reasoning and planning over a distributed multi agent architecture. A related work based on procedural reasoning system and belief desire intention (BDI) architecture is PROPHETA (Fichera et al. 2017), an object oriented procedural Python-based multi agent framework with a declarative language approach, used to control autonomous robots. Since (Costantini 2011), we advocated agent architectures capable of smooth integration of several modules/components representing different behaviors/forms of reasoning, possibly based upon different formalisms. Therefore, the overall agent's behavior can be seen as the result of dynamic combination of these behaviors, also in consequence of the evolution of the agent's environment.

We proposed in particular to adopt Answer Set Programming (ASP) modules, where ASP (cf., among many, (Baral 2003; Leone 2007; Truszczyński 2007) and the references therein) is a successful logic programming paradigm suitable for planning and reasoning with affordable complexity; many efficient implementations of ASP solvers are freely available like: CLASP (Gebser et al. 2007), Cmodels (Lierler 2005), DLV (Leone et al. 2006b), Smodels (Elkabani, Pontelli, and Son 2004) . The DALI agent-oriented language and framework was invented, designed and developed in our research group (De Gasperis, Costantini, and Nazzicone 2014; Costantini and Tocchio 2002; 2004; Costantini 2015a); the framework has been lately augmented with a plugin for the invocation of answer set solvers so to build specific modules. The ASP modules can be exploited in agents in a variety of ways: for instance in the case of reasoning about possibility and necessity, and a greater set of reasoning contexts. We have recently enhanced the integration by adopting ASP modules for planning purposes, allowing an agent or a MAS to choose among the various plans that can be obtained by means of suitable preferences.

In this paper, we show an architecture based on DALI and ASP modules to cope with complex goals, but that can be easily generalized to other agent-oriented frameworks; goals that can take profit from the subdivision into subgoals if one of the following (or both) conditions as met:

- the instance size of the planning problem to be solved for reaching the goal is too big for efficient and timely solution, the instance can be partitioned into sub-problems and the sub-solutions can and must be re-combined/merged together;
- the goal naturally splits into sub-goals where plans can/must be devised separately, and then re-combined/merged together at a later stage.

The architecture exploits not a single DALI agent but a distributed MAS (Multi-Agent System), with suitable components for generating and executing plans; it allows to distribute goals and sub-goals while controlling the generation/exploitation of solutions, and possible (even partial) re-planning in case of environmental changes.

We introduce an ideal case study to show how DALI agents can cooperate in order to explore an unknown territory, such as what can happen in the real world upon occurrence of some kind of catastrophic-like disruptive events (earthquake, fire, flooding, terrorist attack), were geo-localized information can easily become obsolete in few seconds and rescue planning is needed, no matter what is the difficulty.

We propose a solution based upon a MAS instead of a monolithic software solution because we consider important that each software component, i.e. agent, should partially retain its autonomy during asynchronous event processing, in the context of agent-oriented software engineering methodologies (Gomez-Sanz and Fuentes-Fernández 2015) . In fact, in this way each agent can be enriched with high-level reasoning/control behaviors that can coexists with the planning/executing activity. The MAS solution also permits to distribute the computational effort among cloud computing facilities and embedded computers so to increase overall robustness by means of advanced features such as self-monitoring and self-diagnostic, as shown in (Bevar et al. 2012). As discussed below the MAS can be based upon a controller agent which partitions a planning problem, established certain features (e.g., related to plan selection), assigns tasks of planning, re-planning and plan execution. ASP modules are meant to be exploited for planning purposes. Qualitative aspects of the proposed solution consist in: (1) the general MAS structure, that can be customized in order to cope with real-world problems; (2) the interaction between the MAS and the ASP module(s); (3) the adoption of user preferences for choosing among possible plans.

The paper is structured as follows. In the first two sections we recall ASP and the DALI language and framework. We then present the proposed MAS architecture, and an abstract case study. Finally we discuss the proposal and conclude.

## Answer Set Programming in a Nutshell

"Answer set programming" (ASP) is a well-established logic programming paradigm adopting logic programs with default negation under the *answer set semantics*, which (Gelfond and Lifschitz 1988; 1991) is a view of logic programs as sets of inference rules (more precisely, default inference rules). In fact, one can see an answer set program as a set of constraints on the solution of a problem, where each answer set represents a solution compatible with the constraints expressed by the program. For the applications of ASP, the reader can refer for instance to (Baral 2003; Leone 2007; Truszczyński 2007). However, planning is among the more suitable an successful applications of ASP , cf (Son 2017; Romero, Schaub, and Son 2017) and the references therein, were planning in ASP is analyzed even under incomplete information.

Syntactically, a program (or, for short, just "program") $\Pi$ is a collection of *rules* of the form:

$$H \leftarrow L_1, \ldots, L_m, \, not \; L_{m+1}, \ldots, not \; L_{m+n}$$

where $H$ is an atom, $m \geqslant 0$ and $n \geqslant 0$, and each $L_i$ is an atom. Symbol $\leftarrow$ is usually indicated with :- in practical systems. An atom $L_i$ and its negated counterpart $not \, L_i$ are called *literals*. The left-hand side and the right-hand side of the clause are called *head* and *body*, respectively. A rule with empty body is called a *fact*. A rule with empty head is a *constraint*, where a constraint of the form $\leftarrow L_1, ..., L_n$. states that literals $L_1, \ldots, L_n$ cannot be simultaneously true in any answer set.

Unlike a conventional logic program, a ASP program may have several answer sets, each of which represent a consistent solution to given problem and constraints, or may have no answer set at all, which means that no solution can be found. Whenever a program has no answer sets, it is said that the program is *inconsistent* (w.r.t. *consistent*). In the case of planning, each answer set (if any exists) represents a plan.

All solvers provide a number of additional features useful for practical programming, that we will introduce only whenever needed. Solvers are periodically checked and compared over well-established benchmarks, and over challenging sample applications proposed at the yearly ASP competition (cf. (Calimeri et al. 2012), (Gebser, Maratea, and Ricca 2016) for recent reports).

## The DALI language: Framework and Applications

DALI (Costantini and Tocchio 2002; 2004) is an Agent-Oriented Logic Programming language, (Costantini 2015a) for a comprehensive and updated list of references. A DALI agent is triggered by several kinds of asynchronous events: external events, internal, present and past events. A DALI MAS does not explicitly requires using a global clock mechanism, but temporal logic can be implemented inside agents.

**External events** are syntactically indicated by the postfix *E*. Reaction to each such event is defined by a reactive rule, where the special token :>. The agent remembers to have reacted by converting an external event into a *past event* (postfix *P*). An event perceived but not yet reacted to is called "present event" and is indicated by the postfix *N*.

In DALI, **actions** (indicated with postfix *A*) may have or not preconditions: in the former case, the actions are defined by actions rules, in the latter case they are just action atoms. An action rule is characterized by the new token :<. Similarly to events, actions are recorded as past actions.

**Internal events** is what makes a DALI agent agent proactive. An internal event is syntactically indicated by the postfix *I*, and its description is composed of two rules. The first one contains the conditions (knowledge, past events, procedures, etc.) that must be true so that the reaction (in the second rule) may happen. Thus, a DALI agent is able to react to its own conclusions. Internal events are automatically attempted with a default internal frequency customizable by means of directives in the agent initialization file, where the frequency will depend upon the very nature of each such event, and the degree of criticality for the agent.

The DALI communication architecture implements the DALI/FIPA protocol (Foundation for Intelligent Physical Agents 2003), which consists of the main FIPA primitives, plus few new primitives which are particular to DALI. The architecture may also include a filter on communication based on ontologies and forms of commonsense reasoning, as shown in previous works.

The DALI programming environment at current stage of development (De Gasperis, Costantini, and Nazzicone 2014) offers a multi-platform folder environment, built upon Sicstus Prolog programs, shell scripts, Python scripts to integrate external applications, a JSON/HTML5/jQuery web user interface to integrate into DALI applications, with a Python/Twisted/Flask web server capable to interact with A DALI MAS at the backend. We have recently devised a cloud DALI implementation, reported in (Costantini, De

Gasperis, and Nazzicone 2017; Costantini et al. 2017). In fact, as we have since long been convinced of the potential usefulness of the DALI logical agent-oriented programming language in the cognitive robotic domain, in the abovementioned papers we have presented the extensions to the basic pre-existing DALI implementation with a number of useful new features, and in particular allow a DALI MAS to interact with robots over messages buses like ROS, YARP, Redis event broker. As shown in (Costantini, De Gasperis, and Nazzicone 2017), the DALI framework has been extended to "DALI 2.0" by using open sources packages, protocols and web based technologies. DALI agents can thus be developed to act as high level cognitive robotic controllers, and can be automatically integrated with conventional embedded controllers. The web compatibility of the framework allows real-time monitors and graphical visualizers of the underline MAS activity to be specified, for checking the interaction between an agent and the related robotic subsystem. The cloud package ServerDALI allows a DALI MAS to be integrated into any practical environment. In (Costantini et al. 2017) paper we have illustrated the new "Koiné DALI" framework, where a Koiné DALI MAS can cooperate without problems with other MASs, programmed in other languages, and with object-oriented applications. In summary, the enhanced DALI can be used for multi-MAS applications and hybrid multi-agents and object-oriented applications, and can be easily integrated into preexistent applications.

The DALI framework has been experimented, e.g., in applications for user monitoring and training, in emergencies management (like first aid triage assignment), in security or automation contexts, like home automation or processes control, and, more generally, in every situation that is characterized by asynchronous events (either simple events and/or events that are correlated to other ones even in complex patterns). An architecture encompassing DALI agents and called, F&K (Friendly-and-Kind) system (Aielli et al. 2016) has been proposed for (though not restricted to) applications the eHealth domain. F&Ks are "knowledge-intensive" systems, providing flexible access to dynamic, heterogeneous, and distributed sources of knowledge and reasoning, within a highly dynamic computational environment consisting of computational entities, devices, sensors, and services available in the Internet and in the cloud. As a suitable general denomination for systems such as F&Ks we propose "Dynamic Proactive Expert Systems" (DyPES): in fact, such systems are aimed at supporting human experts and personnel or human users in a knowledgeable fashion, so they are reminiscent of the role of traditional expert systems. However, they are proactive in the sense that such systems have objectives (e.g., monitoring patients, managing resources, exploring territories, etc.) that they pursue autonomously, requiring human intervention only when needed. They are also dynamic, because they are able to exploit not only a predefined knowledge base: rather, they are equipped with a number of reasoning modules, and they are able to locate other such modules, and the necessary knowledge and reasoning auxiliary resources. In fact, DyPESs are characterized by "Knowledge-intensity", in the sense that in general a large amount of heterogeneous information and data must be retrieved, shared and integrated in order to reason within the system's domain. DyPESs can be Cyber-Physical Systems integrating software and physical components (Khaitan and McCalley 2015), and can be able to perform Complex Event Processing, i.e., to actively monitor event data so as to make automated decisions and take time-critical actions (DALI has been in fact empowered with CEP capabilities (Costantini 2015b)).

Agents (and in particular robotic agents) have complex goals that may need to be decomposed, either hierarchically or anyway into related subgoals; moreover, such goals may change in time depending upon the interaction with the environment. Prolog-based logical agents such as DALI agents but also agents written in other agent-oriented computational-logic-based languages (e.g., AgentSpeak (Rao and Georgeff 1991; Bordini and Hübner 2010), GOAL (Hindriks 2009; 2010), 3APL (Dastani et al. 2004; Dastani, van Birna Riemsdijk, and Meyer 2005)) can devise and execute plans. However, they are not easily able to decompose goals into subgoals, evaluate (based upon preferences) alternative plans, and re-plan if needed, possibly for some subgoals only; implementing such features within a single agent would in fact make the agent code heavy to understand and execute.

We have since long equipped DALI with a plugin for invoking ASP solvers and thus executing ASP modules. When this module is used for planning, it would be possible to choose among the generated plans based upon qualitative and quantitative user preferences; the preference strategies implemented so far are: (i) shortest plan; (ii) minimal-cost plan; (iii) plan including a minimum/maximum number of a certain kind of actions; we intend to implement plan evaluation based upon preferences on resource consumption, following the principles of (Costantini and Formisano 2010; 2009; Costantini, Formisano, and Petturiti 2010).

Below we propose a DALI MAS architecture aimed at goal decomposition, sub-goal assignment, planning and re-planning concerning complex goals.

## The ASP-MAS Architecture

In this section we illustrate the features of the proposed architecture. The DALI MAS is intended to fulfill the so-called *bounded rationality principle* (Gigerenzer 2004), which we translate that a plan for reaching a goal shall to be devised and executed in a timely manner before a ultimate $T_{max}$ deadline. Consequently, there is a second deadline $T_{PlanMax} < T_{Max}$ by which a plan has to be computed and selected, so that the remaining time is sufficient to execute that plan. Parameters $T_{PlanMax}$ and $T_{Max}$ are indeed dependent of the problem domain. At the current state of development they have to be determined by the MAS-ASP designer and stay constant always during run-time phase.

We also consider the hypothesis that for each problem $P$ proposed to the MAS, a trivial solution plan can always be computed in time $T_{Pt}$ by using a well tested deterministic algorithm, such that $T_{Pt}$ is a negligible time compared to $T_{Ps}$, which is the minimum time needed to generate an acceptable sub-optimal plan.

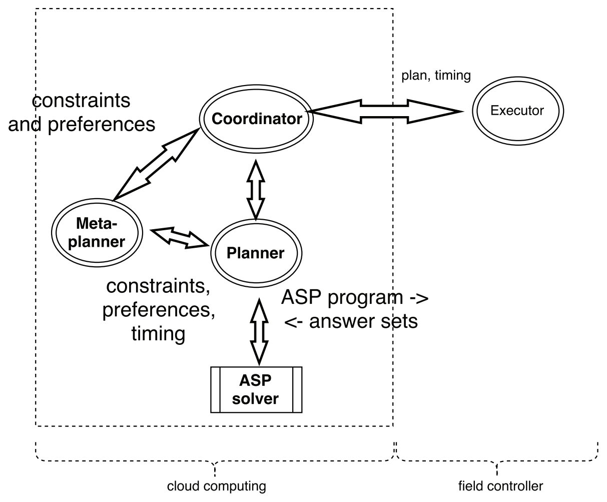


Figure 1: DALI ASP-MAS architecture: **Coordinator**, **Meta-Planner**, **Planner**, **Executor agents**. The MAS can be deployed over a cloud computing architecture, thus distributing and balancing the required computational resources. The ASP module is executed via an external solver, configurable depending on the required capabilities. The **executor** agent is supposed to actually execute the plan, possibly working "in the field", i.e., embedded in a mobile robot or some other ad-hoc facility or mechanism. Constraints can be used to codify knowledge about the environment, like obstacles, target coordinates, resources, depending on the problem domain.

Thus, given the input set $T_{PlanMax}, T_{Max}, G, N, C$, where $G$ is the goal, $N$ is the instance size of the problem to be solved (if applicable), $C$ is the constraints set which models the dynamics and knowledge about the environment, the MAS operates via the following steps, not necessarily in sequence, but in parallel whenever it is possible:

(i) Decompose the overall goal into suitable subgoal;

(ii) For each subgoal, generate an a sub-plan within the $T_{PlanMax}$ deadline;

(iii) Execute the plan within the $T_{Max}$ deadline deploying over the set of executors;
in case of failure (insufficient time to execute), maximize the length of the partially executed plan;

(iv) In case of a change of conditions in the environment, i.e. constraints change, re-plan, possibly limiting this activity to specific subgoals resulting from the partitioning.

Since each ASP module may possibly find more than one plan for given (sub-)goal, it is useful (as said before) to apply a given metrics by which a plan could be preferred to another one. The proposed DALI ASP-MAS architecture is shown in Figure 1 and the agent behaviors are here described .

- **COORDINATOR** agent: this agent synchronizes all the actions of the MAS and updates the global state of goal solving. Its task are the following.

  (a) Ensure the proper activation of the MAS and overall self checking.
  (b) Interact with the external world and whenever needed acquire new constraints for the MAS or revise the present goals.
  (c) Control the $T_{PlanMax}$ and $T_{Max}$ deadlines.
  (d) Decompose the goal into subgoals.
  (e) For each subgoal, instantiate a **META-PLANNER** agent, possibly providing as input the preference criterion for plan selection.
  (f) receive from each **META-PLANNER** agent the sub-plan to be executed up to $T_{PlanMax}$ and deploy the overall plan to the **EXECUTOR** agents set, each is in charge of sub-plan execution within maximum time $T_{Max} - T_{PlanMax}$.
  (h) If time elapses, or new events occur, cancel the current running plan and if applicable send a replan indication to the **META-PLANNER**.
  (h) Logs all events to a log server.

- **META-PLANNER** agent, whose tasks are the following.

  (a) Receive the triggering event from the **COORDINATOR** with new constraints to start the search for a new plan.
  (b) Generate input set of constraints and specific data for the **PLANNER** agent while monitoring its performances. If **PLANNER** agent does not deliver before $T_{PlanMax} - T_{Pt}$, cancel the plan request and ask **PLANNER** to generate a trivial plan .
  (c) Apply plan selection accorded to preferences, either local or set by **COORDINATOR** agent. It also exploits the given preference criterium in order to select the plan which is closer to present preferences whenever the **PLANNER** returns more than one answer.
  (d) If requested by **COORDINATOR**, ask **PLANNER** for re-planning with updated input set of contraints.

- **PLANNER** agent, which receives as input the time constraints $T_{PlanMax}, T_{Max}, C_{\%}, N, F$ from **META-PLANNER** generate the ASP program which then generates all possible sub-plan via the ASP module, if possible within the $T_{PlanMax}$ deadline. If more than a single answer is produced by the ASP solver, it returns all available plans to the **META-PLANNER**. If no solution exists, it generates a trivial plan (if possible). The $C_{\%}$ parameter encode knowledge about the sub-optimality of the desired plan type, which coincide with the Hamiltonian plan at 100%, or refers to sub-optimal plans for lower percentages.

- **EXECUTOR**: each agent puts into action in the real world the specific sub-plan provided by the **COORDINATOR**, if possible within the $T_{Max}$ deadline, and notifies the **COORDINATOR** upon completion. The executor agent in general executes plans (also) embodied in a physical components in a Cyber-Physical System, and/or by means of robotic elements of various kinds. In Figure 1, **EXECUTOR** is designated as "field controller" as plan execution is situated into some environment.

Summarizing, the final execution made the EXECUTORs depends on the following information:

- timing parameters, ASP program templates, static constraints imposed by the designer
- selected goals and preferences by the user
- the environment model built upon sensors perceptions which define dynamic constraints
- consistency and self-checking rules in the knowledge base
- available energy and resources, which may also have non trivial impact on hardening the constraints set.

Since in general this is a hard-NP problem, most probably only sub-optimal plans can be generated, but with a controllable desirable quality by balancing user preferences, accuracy, and weak vs. hard constraints. The resulting behavior should be similar to what a rational human expert would do in similar circumstances, with the advantage of not being limited also by human errors due to over fatigue and less concentration. So the human could dedicate himself to supervise the overall system behavior under less cognitive load stress and intervene with appropriate common sense reasoning when needed, most probably when the system is producing too many trivial plans.

## Abstract Case Study

The ASP-MAS architecture presented above has been inspired and motivated by a case-study that has been actually implemented and experimented, and presented in (Costantini, De Gasperis, and Nazzicone 2015). The overall goal in the case study is to explore an unknown territory upon occurrence of some kind of catastrophic-like disruptive event (earthquake, fire, flooding, terrorist attack, etc.). The similarity comes from the idea that after such event, most of the available geo-localized information can became obsolete in a very short time and important decision have to be made in order to save lives and/or deliver rescue services. So there is a contemporary need to re-scan the territory to know were is possible to engage rescue equipments, and to generate an actually rescue plan that covers the maximum possible area were is needed. So there are places were is impossible to go (i.e. *forbidden cells*) and places were victims have to be rescued (i.e. *to_reach cells*).

For simplicity, we have modeled the territory (also called "area") as a set of a $N * N$ parts represented as chessboards, i.e., squares of cells, where some cells are marked as unreachable/forbidden, and are therefore considered as "holes" in the chessboard. This represents the fact that the agents may be notified by an external authority or by other sources of the actual impossibility of traversing that location because of some kind of obstruction/danger. The forbidden/unreachable locations, and their respective constraints set, can change in time as the scenario evolves.

For the sake of experiments, the EXECUTOR agent is embodied by a robot explorer/rescuerer [1] that each agent employs for exploration of the territory; this robot has been represented (in the case study) as a chess' knight piece, which performs knight leaps. This is to signify that a real robot (whatever its kind) will in practice have limited possibilities of movement. In this way, the problem of exploration of a single piece of territory can be modeled as a variant of the well-known "knight tour with holes" problem, for which well-known ASP solutions exist. The ultimate objective would be that of devising an Hamiltonian path, thus fully exploring the given piece of territory while skipping the forbidden squares. As however the Hamiltonian path option may results computationally intractable with reasonable instance size (already from sizes $\geq 8$, or 10 using the most recent ASP more efficient solvers ), we resorted to sub-optimal solutions that the MAS is capable to generate, which adopt soft constraints in order to visit each square as few times as possible.

The Knight Tour with holes problem has constituted a benchmark in recent ASP competitions, aimed at comparing ASP solvers performances. We performed a number of modifications to the original version (Calimeri and Zhou 2014) concerning: the representation of holes; the objective of devising a path which, though not Hamiltonian, guarantees a required degree of coverage with the minimum number of multiple-traversals; simple forms of loop-checking for avoiding at least trivial loops. For the sake of completeness, below is the sketch of our solution, formulated for the DLV ASP solver (Leone et al. 2006a), though it might be easily reformulated for other solvers. The key modifications to the base solution are the following.

- We modified the *reached* constraint, and transformed it into a soft constraint, so as not to be forced to finding a Hamiltonian path.

```
reached(X,Y) :- move(1,1,X,Y).
reached(X2,Y2) :-
    reached(X1,Y1), move(X1,Y1,X2,Y2).
:~ cell(X,Y),
   not forbidden(X,Y), not reached(X,Y).
```

- We added a coverage-satisfaction rule, where $coverage$ denotes the required degree of coverage and $number\_forbidden$ the number of holes, and $V$ is the instance size, i.e., the chessboard edge. The maximum possible coverage is 100% of the available cells, i.e., $M = V * V$, while the minimum coverage $N$ is computed in terms of $coverage$, considering the holes. Suitable application of the *count* DLV constraint (Leone et al. 2006a) guarantees the desired coverage.

```
coverage(95).
number_forbidden(5).
cov(N) :-
   N <= #count{X,Y : reached(X,Y)} <= M,
   size(V), coverage(Z),
   number_forbidden(F),
   M = V * V, N2 = M * Z,
   N3 = N2 /100, N = N3 - F.
```

Experimental results have demonstrated the usefulness of the proposed MAS architecture, that is actually able to effectively cope with real-world instance sizes. The architecture in this case study works as follows.

[1] not necessary a robot, also a human guided ambulance, or a combination of UAV and human guided vehicles

- The COORDINATOR agent partitions the territory that must be explored into a number of (possibly overlapping) sections (chessboards) of reasonable size (maximum 10x10 cells), each one to be assigned to a META-PLANNER instance.
- Each plan to be executed (exploration to be performed) is assigned to a separate (EXECUTOR)EXLORER agent, specifically assigned to that territory section. Each instance of the META-PLANNER agent relies upon its own associated instance of the planner agent.
- different preference policies can possibly be associated with different sections of the territory to be explored, according to directions provided by the user/environment.
- The COORDINATOR will devise re-planning for each portion of the territory for which the unreachable location have changed.

Reasonable metrics measure plans returned by the ASP module in terms of: (i) number of cells that have to be visited when using coverage, (ii) length of the path, (iii) presence of loops (when the Hamiltonian constraint is released); (iv) plan cost, in case there is a specific cost associated to each cell. Preference criteria can then be defined by selecting one metric, or by combining different metrics: for instance, a criterium may consist in preferring the shortest path, if it does not exceed a certain cost.

## Concluding Remarks

We have proposed an ASP-MAS architecture for flexible goal decomposition, plan formation and execution that delivers acceptable solution to complex problems under the "*bounded rationality principle*". In real application, a MAS for each (class of) goal(s) would be designed, implemented and located into the DALI cloud. In fact, all components of the MAS will be programmed according to the goal to be reached, i.e., to the problem to be solved. Each agent that needs to solve a goal refers to the suitable MAS. As mentioned, the DALI framework allows uniform access also to agents written in other languages/formalisms. So, the proposed solution is not DALI-specific but rather can be generally adopted.

## References

Aielli, F.; Ancona, D.; Caianiello, P.; Costantini, S.; De Gasperis, G.; Di Marco, A.; Ferrando, A.; and Mascardi, V. 2016. FRIENDLY & KIND with your health: Human-friendly knowledge-intensive dynamic systems for the e-health domain. In *Highlights of Practical Applications of Scalable Multi-Agent Systems. The PAAMS Collection - International Workshops of PAAMS 2016, Proceedings*, volume 616 of *Communications in Computer and Information Science*, 15–26. Springer.

Baral, C. 2003. *Knowledge representation, reasoning and declarative problem solving*. Cambridge University Press.

Bevar, V.; Muccini, H.; Costantini, S.; De Gasperis, G.; and Tocchio, A. 2012. A multi-agent system for industrial fault detection and repair. In *Advances on Practical Applications of Agents and Multi-Agent Systems.*, Advances in Intelligent and Soft Computing, 47–55. Springer, Berlin Heidelberg. Paper and demo.

Bordini, R. H., and Hübner, J. F. 2010. Semantics for the jason variant of agentspeak (plan failure and some internal actions). In Coelho, H.; Studer, R.; and Wooldridge, M., eds., *ECAI 2010 - 19th European Conference on Artificial Intelligence, Proceedings*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, 635–640. IOS Press.

Calimeri, F., and Zhou, N.-F. 2014. Knight tour with holes ASP encoding. See http://www.mat.unical.it/aspcomp2013/files/links/benchmarks/encodings/aspcore-2/22-Knight-Tour-with-holes/encoding.asp.

Calimeri, F.; Ianni, G.; Krennwallner, T.; and Ricca, F. 2012. The answer set programming competition. *AI Magazine* 33(4):114–118.

Costantini, S., and Formisano, A. 2009. Modeling preferences and conditional preferences on resource consumption and production in ASP. *Journal of of Algorithms in Cognition, Informatics and Logic* 64(1).

Costantini, S., and Formisano, A. 2010. Answer set programming with resources. *Journal of Logic and Computation* 20(2):533–571.

Costantini, S., and Tocchio, A. 2002. A logic programming language for multi-agent systems. In *Logics in Artificial Intelligence, Proceedings of the 8th Europ. Conf.,JELIA 2002*, LNAI 2424. Springer-Verlag, Berlin.

Costantini, S., and Tocchio, A. 2004. The DALI logic programming agent-oriented language. In *Logics in Artificial Intelligence, Proceedings of the 9th European Conference, Jelia 2004*, LNAI 3229. Springer-Verlag, Berlin.

Costantini, S.; De Gasperis, G.; Pitoni, V.; and Salutari, A. 2017. Dali: A multi agent system framework for the web, cognitive robotic and complex event processing. In *Proceedings of the 32nd Italian Conference on Computational Logic*, volume 1949 of *CEUR Workshop Proceedings*, 286–300. CEUR-WS.org. http://ceur-ws.org/Vol-1949/CILCpaper05.pdf.

Costantini, S.; De Gasperis, G.; and Nazzicone, G. 2015. Exploration of unknown territory via DALI agents and ASP modules. In Omatu, S.; Malluhi, Q. M.; Rodríguez-González, S.; Bocewicz, G.; Bucciarelli, E.; Giulioni, G.; and Iqba, F., eds., *Distributed Computing and Artificial Intelligence, 12th International Conference, DCAI 2015, Salamanca, Spain, June 3-5, 2015*, volume 373 of *Advances in Intelligent Systems and Computing*, 285–292. Springer.

Costantini, S.; De Gasperis, G.; and Nazzicone, G. 2017. DALI for cognitive robotics: Principles and prototype implementation. In Lierler, Y., and Taha, W., eds., *Practical Aspects of Declarative Languages - 19th International Symposium, Proceedings*, volume 10137 of *Lecture Notes in Computer Science*, 152–162. Springer.

Costantini, S.; Formisano, A.; and Petturiti, D. 2010. Extending and implementing RASP. *Fundam. Inform.* 105(1-2):1–33.

Costantini, S. 2011. Answer set modules for logical agents.

In de Moor, O.; Gottlob, G.; Furche, T.; and Sellers, A., eds., *Datalog Reloaded: First International Workshop, Datalog 2010*, volume 6702 of *LNCS*. Springer. Revised selected papers.

Costantini, S. 2015a. The DALI agent-oriented logic programming language: Summary and references 2015.

Costantini, S. 2015b. Ace: a flexible environment for complex event processing in logical agents. In Matteo Baldoni, L. B., and Dastani, M., eds., *Engineering Multi-Agent Systems, Third International Workshop, EMAS 2015, Revised Selected Papers*, volume 9318 of *Lecture Notes in Computer Science*. Springer.

Dastani, M.; van Riemsdijk, B.; Dignum, F.; and Meyer, J.-J. C. 2004. A programming language for cognitive agents goal directed 3apl. In Dastani, M.; Dix, J.; and Fallah-Seghrouchni, A. E., eds., *Programming Multi-Agent Systems, First International Workshop, PROMAS 2003, Selected Revised and Invited Papers*, volume 3067 of *Lecture Notes in Computer Science*, 111–130. Springer.

Dastani, M.; van Birna Riemsdijk, M.; and Meyer, J.-J. C. 2005. Programming multi-agent systems in 3apl. In *Multi-agent programming*. Springer. 39–67.

De Gasperis, G.; Costantini, S.; and Nazzicone, G. 2014. Dali multi agent systems framework, doi 10.5281/zenodo.11042. DALI GitHub Software Repository. DALI: http://github.com/AAAI-DISIM-UnivAQ/DALI.

Elkabani, I.; Pontelli, E.; and Son, T. C. 2004. Smodels with clp and its applications: A simple and effective approach to aggregates in asp. In *International Conference on Logic Programming*, 73–89. Springer.

Fichera, L.; Messina, F.; Pappalardo, G.; and Santoro, C. 2017. A python framework for programming autonomous robots using a declarative approach. *Science of Computer Programming* 139:36–55.

Foundation for Intelligent Physical Agents. 2003. FIPA Interaction Protocolo Specifications.

Gebser, M.; Kaufmann, B.; Neumann, A.; and Schaub, T. 2007. clasp: A conflict-driven answer set solver. In *International Conference on Logic Programming and Nonmonotonic Reasoning*, 260–265. Springer.

Gebser, M.; Maratea, M.; and Ricca, F. 2016. What's hot in the answer set programming competition. In *AAAI*, volume 16, 4327–4329.

Gelfond, M., and Lifschitz, V. 1988. The stable model semantics for logic programming. In Kowalski, R., and Bowen, K., eds., *Proceedings of the 5th International Conference and Symposium on Logic Programming (ICLP/SLP'88)*. The MIT Press. 1070–1080.

Gelfond, M., and Lifschitz, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Computing* 9:365–385.

Gigerenzer, G. 2004. Fast and frugal heuristics: The tools of bounded rationality. *Blackwell handbook of judgment and decision making* 62:88.

Gomez-Sanz, J. J., and Fuentes-Fernández, R. 2015. Understanding agent-oriented software engineering methodologies. *The Knowledge Engineering Review* 30(4):375–393.

Hindriks, K. V. 2009. Programming rational agents in goal. In *Multi-Agent Programming*. Springer US. 119–157.

Hindriks, K. 2010. A verification logic for goal agents. In Dastani, M. M.; Hindriks, K.; and Meyer, J.-J. C., eds., *Specification and Verification of Multi-agent Systems*. Springer.

Jiming, L. 2001. *Autonomous agents and multi-agent systems: explorations in learning, self-organization and adaptive computation*. World Scientific.

Khaitan, S. K., and McCalley, J. D. 2015. Design techniques and applications of cyberphysical systems: A survey. *IEEE Systems Journal* 9(2):350–365.

Knudson, M., and Tumer, K. 2011. Adaptive navigation for autonomous robots. *Robotics and Autonomous Systems* 59(6):410–420.

Leone, N.; Pfeifer, G.; Faber, W.; Eiter, T.; Gottlob, G.; Perri, S.; and Scarcello, F. 2006a. The dlv system for knowledge representation and reasoning. *ACM Transactions on Computational Logic* 7(3):499–562.

Leone, N.; Pfeifer, G.; Faber, W.; Eiter, T.; Gottlob, G.; Perri, S.; and Scarcello, F. 2006b. The dlv system for knowledge representation and reasoning. *ACM Transactions on Computational Logic (TOCL)* 7(3):499–562.

Leone, N. 2007. Logic programming and nonmonotonic reasoning: From theory to systems and applications. In Baral, C.; Brewka, G.; and Schlipf, J., eds., *Logic Programming and Nonmonotonic Reasoning, 9th International Conference, LPNMR 2007*.

Lierler, Y. 2005. cmodels–sat-based disjunctive answer set solver. In *International Conference on Logic Programming and Nonmonotonic Reasoning*, 447–451. Springer.

Rao, A. S., and Georgeff, M. 1991. Modeling rational agents within a BDI-architecture. In *Proceedings of the Second Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'91)*, 473–484. Morgan Kaufmann.

Romero, J.; Schaub, T.; and Son, T. C. 2017. Generalized answer set planning with incomplete information. In Bogaerts, B., and Harrison, A., eds., *Proceedings of the 10th Workshop on Answer Set Programming and Other Computing Paradigms co-located with the 14th International Conference on Logic Programming and Nonmonotonic Reasoning, ASPOCP@LPNMR 2017*, volume 1868 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Son, T. C. 2017. Answer set programming and its applications in planning and multi-agent systems. In Balduccini, M., and Janhunen, T., eds., *Logic Programming and Nonmonotonic Reasoning - 14th International Conference, LPNMR 2017, Proceedings*, volume 10377 of *Lecture Notes in Computer Science*, 23–35. Springer.

Truszczyński, M. 2007. Logic programming for knowledge representation. In Dahl, V., and Niemelä, I., eds., *Logic Programming, 23rd International Conference, ICLP 2007*, 76–88.

# Constraint-Based Online Transformation of Abstract Plans into Executable Robot Actions

**Till Hofmann,[1] Victor Mataré,[2] Stefan Schiffer,[1,2]
Alexander Ferrein,[2] Gerhard Lakemeyer[1]**
[1]Knowledge-Based Systems Group,
RWTH Aachen University, 52056 Aachen, Germany
[2]Mobile Autonomous Systems and Cognitive Robotics,
FH Aachen University of Applied Sciences, 52066 Aachen, Germany

**Abstract**

In this paper, we are concerned with making the execution of abstract action plans for robotic agents more robust. To this end, we propose to model the internals of a robot system and its ties to the actions that the robot can perform. Based on these models, we propose an online transformation of an abstract plan into executable actions conforming with system specifics. With our framework, we aim to achieve two goals. First, modeling the system internals is beneficial in its own right in order to achieve long term autonomy, system transparency, and comprehensibility. Second, separating the system details from determining the course of action on an abstract level leverages the use of planning for actual robotic systems.

## Introduction

Despite promising advances in planning systems, they see surprisingly little use in actual robotics environments. We believe this is because solving a planning task by itself is not sufficient to accomplish high-level behavior control of a robotic system. For one, the robot's platform (i.e., its hardware and low-level software components) often requires additional constraints that are ignored during planning, e.g., a domestic service robot participating in RoboCup@Home (Wisspeintner et al. 2009) must calibrate its arm before performing any manipulation tasks. During planning, we do not want to plan for all the requirements of the underlying platform, as this would increase the problem size significantly and would make it infeasible in practice. However, ignoring those constraints at the behavior level and dealing with them at the lower levels is often impossible, because platform constraints may entail changes to the action plan.

Another reason for such a separation of high-level behavior and low-level platform is a design problem: When modelling the domain, an agent programmer usually does not want to deal with the robot platform. On the other hand, a platform designer should not need to consider and adapt the high-level behavior when modifying the platform. Also, a robot often has to deal with failed actions, unexpected changes, and exogenous events. Thus, a considerable amount of monitoring is required when executing a high-level plan on a robot.

For these reasons, we propose a framework that allows the modelling of the robot platform and its constraints independent of the behavioral component. While designing the platform, the user designs a self model of the robot and defines all the constraints of the platform. The world model of the agent can be designed without taking low-level constraints into account. During execution, the abstract action plan is transformed into a concrete executable plan that satisfies the constraints of the lower levels.

To actually achieve a separation between the problem domain and platform-related execution concerns, the platform needs a certain degree of "self-awareness" in terms of its components, their capabilities, their states and their interdependencies. Our goal in this paper is to sketch out requirements for a logically founded constraint language that can be used by platform experts to explicitly model component state transitions, dependencies among them, error conditions and possible recovery strategies, including the potential need for human assistance. The result is an agent system capable of self-maintenance by generating platform-specific monitoring and recovery strategies from the platform model and a platform-independent action plan. This eliminates much of the expert intervention that is required to keep robots running in dynamic domains, while providing a generic framework that helps in decoupling strategic decision-making from any platform details.

## Foundations & Related Work

Especially the research into planning systems that is focused on temporal coordination of (concurrent) actions is of particular interest to our endeavour (Tsamardinos, Muscettola, and Morris 1998; Jónsson et al. 2000; Kim, Williams, and Abramson 2001; Lemai and Ingrand 2004). In theory, it would allow generalizing both the domain logic and the platform details as a temporal planning problem.

Temporal optimization and parallelization of platform-dependent operations is also being performed successfully at the task execution level. Keith et al. (2009) employ a temporal network that describes platform constraints to re-order and optimize the manipulator trajectories specified by a sequential plan. Konečný et al. (2014) separate the strategic planning layer that only handles an abstract domain conceptualization from the detailed execution strategy that makes a plan executable on a real robot. However, the *Consistency Based Execution Monitoring* directly maps abstract, but fully grounded plan elements to a domain-specific execution strategy, without specifying an explicit platform model.

Kunze, Roehm, and Beetz (2011) introduce the Semantic Robot Description Language (SRDL) to bridge the gap between purely kinematic description languages and the more abstract level at which task specifications are usually formulated. They leverage the Web Ontology Language (Bechhofer et al. 2004) to model how domain-specific actions depend on platform-specific components that are required to realize them. Waibel et al. (2011) use SRDL to implement a shared knowledge base that allows robots to improve their search and execution strategies with previous observations possibly made by other robots. In this case, the knowledge base covers both platform-specific and domain-specific knowledge within a common deduction engine based on Description Logic (Baader 2003). The works based on SRDL are related to our work in their purpose, but differ significantly in that the SRDL model is purely a translation layer that sits between the abstract action plan and the executive layer. As such, SRDL specifications cannot be used to modify execution strategies at runtime, and thus cannot be used to dynamically deduce error recovery strategies. Mansouri and Pecora (2016) describe a constraint-based approach to hybrid reasoning with a meta-CSP that describes the different types of knowledge. The CSP is solved by a meta-solver that combines different kinds of reasoners. CHIMP (Stock et al. 2015) uses HTNs to solve such constraint-based hybrid reasoning tasks. HTN-based task decomposition approaches often model platform details as part of the planning problem. Dvorák et al. (2014) limit the problem size by delegating execution monitoring to a PRS subsystem with a simple success/failure interface.

Based on the Situation Calculus (McCarthy and Hayes 1969), the action language GOLOG allows a programmer to intermix imperative programming with planning on a logically formulated domain model (Levesque and Lakemeyer 2008). READYLOG (Ferrein and Lakemeyer 2008) extends the search functionality of GOLOG to allow for decision-theoretic planning. Finzi and Pirri (2005) provide a theoretical integration of the Situation Calculus with temporal constraints. De Giacomo, Reiter, and Soutchanski (1998) define an execution monitor in Golog that allows to react to unexpected changes during execution. Hofmann et al. (2016) interleave PDDL-based planning with Golog-based execution for monitoring purposes. Schiffer, Wortmann, and Lakemeyer (2010) describe an online transformation of a READYLOG program by inserting actions to satisfy qualitative temporal platform-specific constraints, under the assumption that agent domain and platform domain are disjunct.

## Approach

Our goal is to design a framework that allows the user to formulate a platform constraint model that describes internal and external dependencies of component states, both in terms of hardware and software. An agent framework can then turn an abstract plan into a platform-specific execution and monitoring strategy that satisfies these constraints. This allows a separation of the high-level program from the specific platform properties while complying with the platform constraints. In the following, we present the different components of such a framework.

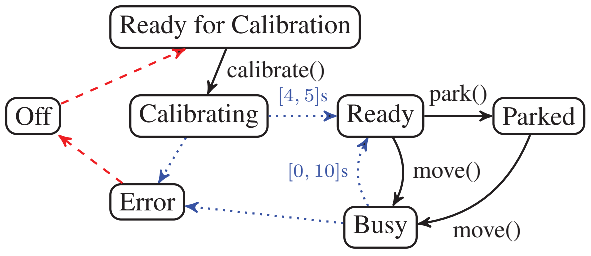


Figure 1: A finite state machine as a platform model for the Katana arm with three types of transitions: agent actions (black/solid), system events (blue/dotted), exogenous events (red/dashed). The edges are annotated with their action and expected time bounds.

### Platform Models

Figure 1 shows an example for a model of a robotic manipulator arm, the Katana. Before the Katana arm can be used, it needs to be calibrated. Initially, the arm is turned off. It can only start its calibration process from a specific calibration position, so a human assistant must move the arm into the right position and then turn it on, which brings the arm into the state *Ready for Calibration*. From that state, the agent can decide to start the calibration. Note that this usually does not happen automatically, because the agent first has to make sure that it is in a location that allows an arm calibration, and second it may not need the arm at all. Since calibration is time-consuming, it should only be done if the arm is actually required. When the calibration is finished, the component driver triggers a transition to either the *Error* state or the *Ready* state. Similar to Schiffer, Wortmann, and Lakemeyer (2010), we model system components as state automata. But as the example in Fig. 1 shows, we need to differentiate between different kinds of transitions: 1. actions by the agent (black), 2. events triggered by the system (blue), 3. exogenous events (red).

**Suitable Automata Models** The platform model shown in Figure 1 is a finite state automaton with multiple edge types. However, more expressive automata models may be required to represent platform components. Consider a navigation stack that depends on the states of several low-level components, e.g., collision avoidance and localization. Each of these components will be modeled separately, but we might also want to formulate constraints on composite states covering multiple components. Hierarchical state machines as described in Girault, Lee, and Lee (1999) may be suitable to formulate such component-level abstractions. Timed transitions, such as the transition $Calibrating \rightarrow Ready$ may be modeled with timed automata (Alur and Dill 1994). While we will not change the foundation of our high-level reasoning, i.e., a situation calculus-based framework, we might consider a Petri-Net-based model such as the one described by Ziparo et al. (2011) for the component description as well.

## Constraints

Platform constraints define properties that must always hold during the execution of the action plan. Based on Figure 1 and using Allen's interval relations (Allen 1983), we can define multiple constraints that must hold for the arm:

1. To calibrate the arm, the robot must be *at* a *free* location (i.e., a location without close objects).

$$\mathit{free}(\mathit{at}(x))\ \textbf{during}\ \mathit{state}(\mathit{arm}) = \mathit{Calibrating}$$

2. When starting to pick up an object, the arm must be ready or parked.

$$\mathit{state}(\mathit{arm}) = \mathit{Ready}\ \textbf{meets}\ \mathit{pickup}(x) \vee$$
$$\mathit{state}(\mathit{arm}) = \mathit{Parked}\ \textbf{meets}\ \mathit{pickup}(x)$$

3. Whenever the robot is moving, the arm must be parked.

$$\mathit{state}(\mathit{arm}) = \mathit{Parked}\ \textbf{during}$$
$$\mathit{state}(\mathit{navigation}) = \mathit{Moving}$$

**Quantitative Temporal Constraints** The examples above are qualitative temporal constraints. However, some components also require quantitative temporal constraints. Consider an RGBD camera that is used for perception. We can formulate the following constraints about the camera:

1. The camera needs some time to initialize, and therefore needs to be started one second before it can be used:

$$\mathit{state}(\mathit{camera}) = \mathit{Running}\ \textbf{before}_{\geq 1s}\ \mathit{detect}(x)$$

2. On the other hand, image processing is expensive, and thus should only be turned on if it is actually used within the next two seconds:

$$\mathit{state}(\mathit{camera}) \neq \mathit{Running}\ \textbf{unless}_{\leq 2s}\ \mathit{detect}(x)$$

The constraints above will be formulated in a temporal extension of the Situation Calculus and may refer to states of system components, fluents, and actions. While previous work only allowed qualititative temporal constraints (Schiffer, Wortmann, and Lakemeyer 2010), we want to allow for quantitative temporal constraints. In order to do so, we will extend the Situation Calculus based on Reiter (1996) and Gabaldon (2003) with qualitative and quantitative temporal aspects and embed the Metric Interval Temporal Logic (MITL) (Alur, Feder, and Henzinger 1996) into the Situation Calculus.

## Events, Temporal Constraints, and Concurrency

The model of the Katana arm shown in Figure 1 has three kinds of edges: 1. Action edges that are directly triggered by the agent and are therefore under agent control, 2. Events that are triggered by the component itself, e.g. to end a durative action, 3. Exogenous events that are triggered by an external participant not under the agent's direct control, e.g., a human. Previously, both kinds of events were modeled as explicit exogenous actions with respective waiting actions. In our approach, we want to make use of concurrency in Golog with the *waitFor* construct (Grosskreutz and Lakemeyer 2003).

If we want to use the model of a system component to plan for a certain system configuration, e.g., a calibrated arm, we need to know about *expected* events. As an example, if the agent decides to start the calibration, it expects the calibration to finish successfully. If this was not the case, the agent could not cause state changes of system components in a meaningful way, as the outcome of any event transitions would be unknown. In addition to the information which transition is to be expected, we also annotate system events with expected time bounds. This allows the agent not only to reason about which event will occur, but also when it will occur. In the Katana example, we annotate the edge $\mathit{Calibrating} \rightarrow \mathit{Ready}$ with the expected time bounds $[4, 5]$, i.e., we expect the calibration to take at least four and at most five seconds. This way, the agent knows that it needs to start the calibration at least five seconds before it can use the arm.

## Action Plan Transformation & Constraint Satisfaction

Given a platform-specific constraint model, an abstract action plan can be transformed into a platform-specific action plan that satisfies all constraints. To create such a plan, first the Golog interpreter determines an abstract action plan as usual. Next, the constraints are transformed into constraint networks (Dechter, Meiri, and Pearl 1991; Meiri 1996). In contrast to Finzi and Pirri (2005), we will not make use of *timelines*, but instead restrict our approach to interleaved and possibly true concurrency in order to allow a simpler formalization. Additionally, our approach will support quantitative constraints. The resulting constraint network will be evaluated with existing constraint solvers. A solution of the constraint network will determine the order of events with their interval limits. Platform constraints, e.g., $\mathit{state}(\mathit{arm}) = \mathit{Ready}$, must be transformed into actions by determining a suitable action sequence based on the platform model. The method of determining this action sequence depends on the underlying state machine model. For a simple state machine as shown in Figure 1, the actions can be determined by searching for a sequence of transitions that result in the desired state. For other, more expressive models, more complex methods may be necessary.

In some cases, such as the calibration of the Katana arm, inserting a single action may suffice. In other cases, the original action plan must be modified, e.g. to actively seek out localization features before some delicate manipulation task can be performed. Thus, a clear separation of the abstract agent and the plan transformation is not always possible and significant modifications of the original plan may be necessary. For this reason, the transformation of the abstract action plan into an executable plan will be part of the high-level agent and implemented within the Golog interpreter.

## Conclusion

We presented a concept for an agent system with an explicit model of the robotic platform and its constraints. The robotic platform is modeled with state automata based on timed automata and hierarchical state machines and allows multiple transition types for agent actions, system events, and exogenous events. Based on these models, the user can formulate constraints in an extension of the Situation Calculus, which allows to define platform-specific, quantitative temporal constraints. During execution, the abstract action plan is modified to satisfy all constraints of the underlying platform. The proposed agent system allows the user to separate behavior control and platform management while taking into account that the constraints may require significant changes to the abstract action plan, which are handled by the agent system during execution.

## Acknowledgments

This work was supported by the German National Science Foundation (DFG) under grant numbers GL-747/23-1 and FE 1077/4-1.

## References

Allen, J. F. 1983. Maintaining knowledge about temporal intervals. *Commun ACM* 26(11):832–843.

Alur, R., and Dill, D. L. 1994. A theory of timed automata. *Theoretical Computer Science* 126(2):183–235.

Alur, R.; Feder, T.; and Henzinger, T. A. 1996. The benefits of relaxing punctuality. *J ACM* 43(1):116–146.

Baader, F. 2003. *The description logic handbook: Theory, implementation and applications*.

Bechhofer, S.; van Harmelen, F.; Hendler, J.; Horrocks, I.; McGuinness, D. L.; Patel-Schneider, P. F.; and Stein, L. A. 2004. OWL Web Ontology Language Reference. Technical report, W3C.

De Giacomo, G.; Reiter, R.; and Soutchanski, M. 1998. Execution Monitoring of High-Level Robot Programs. *Proc. of the 6th Int'l Conf. on Knowledge Representation and Reasoning (KR)*.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49(1-3):61–95.

Dvorák, F.; Barták, R.; Bit-Monnot, A.; Ingrand, F.; and Ghallab, M. 2014. Planning and acting with temporal and hierarchical decomposition models. In *Tools with Artificial Intelligence (ICTAI), 2014 IEEE 26th International Conference on*, 115–121. IEEE.

Ferrein, A., and Lakemeyer, G. 2008. Logic-based robot control in highly dynamic domains. *Robotics and Autonomous Systems* 56(11):980–991.

Finzi, A., and Pirri, F. 2005. Representing flexible temporal behaviors in the situation calculus. In Kaelbling, L. P., and Saffiotti, A., eds., *Proc. of the 19th Int'l Joint Conf. on Artificial Intelligence (IJCAI-05)*, 436–441.

Gabaldon, A. 2003. Compiling control knowledge into preconditions for planning in the situation calculus. In Gottlob, G., and Walsh, T., eds., *Proc. of the 18th Int'l Joint Conf. on Artificial Intelligence (IJCAI-03)*, 1061–1066.

Girault, A.; Lee, B.; and Lee, E. A. 1999. Hierarchical finite state machines with multiple concurrency models. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 18(6):742–760.

Grosskreutz, H., and Lakemeyer, G. 2003. ccGolog – a logical language dealing with continuous change. *Logic Journal of the IGPL* 11(2):179–221.

Hofmann, T.; Niemueller, T.; Claßen, J.; and Lakemeyer, G. 2016. Continual Planning in Golog. In *Proc. of the 30th Conf. on Artificial Intelligence (AAAI)*.

Jónsson, A. K.; Morris, P. H.; Muscettola, N.; Rajan, K.; and Smith, B. D. 2000. Planning in interplanetary space: Theory and practice. In Czhien, S.; Kambhampati, S.; and Knoblock, C. A., eds., *Proc. of the 15th Int'l Conf. on Artificial Intelligence Planning Systems, (AIPS-00)*, 177–186.

Keith, F.; Mansard, N.; Miossec, S.; and Kheddar, A. 2009. From discrete mission schedule to continuous implicit trajectory using optimal time warping. In *Proc. of the 19th Int'l Conf. on Automated Planning and Scheduling*.

Kim, P.; Williams, B. C.; and Abramson, M. 2001. Executing reactive, model-based programs through graph-based temporal planning. In *Proc. of the 17th Int'l Joint Conf. on Artificial Intelligence (IJCAI-01)*, 487–493.

Konečný, Š.; Stock, S.; Pecora, F.; and Saffiotti, A. 2014. Planning domain+ execution semantics: A way towards robust execution? In *2014 AAAI Spring Symposium Series – Qualitative Representations for Robots*.

Kunze, L.; Roehm, T.; and Beetz, M. 2011. Towards semantic robot description languages. In *IEEE Int'l Conf. on Robotics and Automation (ICRA 2011)*, 5589–5595.

Lemai, S., and Ingrand, F. 2004. Interleaving temporal planning and execution in robotics domains. In McGuinness, D., and Ferguson, G., eds., *Proc. of the 19th Nat'l Conf. on Artificial Intelligence (AAAI-04) and 16th Conf. on Innovative Applications of Artificial Intelligence (IAAI-04)*, 617–622.

Levesque, H., and Lakemeyer, G. 2008. Chapter 23 Cognitive Robotics. In Frank van Harmelen, V. L., and Porter, B., eds., *Handbook of Knowledge Representation*, volume 3 of *Foundations of Artificial Intelligence*. 869–886.

Mansouri, M., and Pecora, F. 2016. A robot sets a table: a case for hybrid reasoning with different types of knowledge. *Journal of Experimental & Theoretical Artificial Intelligence* 28(5):801–821.

McCarthy, J., and Hayes, P. 1969. Some philosophical problems from the standpoint of artificial intelligence. In Meltzer, B., and Michie, D., eds., *Machine Intelligence 4*. 463–502.

Meiri, I. 1996. Combining qualitative and quantitative constraints in temporal reasoning. *Artificial Intelligence* 87(1-2):343–385.

Reiter, R. 1996. Natural actions, concurrency and continuous time in the situation calculus. In Aiello, L. C.; Doyle, J.; and Shapiro, S. C., eds., *Proc. of the 5th Int'l Conf. in Principles of Knowledge Representation and Reasoning (KR-96)*, 2–13.

Schiffer, S.; Wortmann, A.; and Lakemeyer, G. 2010. Self-Maintenance for Autonomous Robots controlled by ReadyLog. In Ingrand, F., and Guiochet, J., eds., *Proc. of the 7th*

*IARP Workshop on Technical Challenges for Dependable Robots in Human Environments (DRHE2010)*, 101–107.

Stock, S.; Mansouri, M.; Pecora, F.; and Hertzberg, J. 2015. Online task merging with a hierarchical hybrid task planner for mobile service robots. In *Proc. of the Int'l Conf. on Intelligent Robots and Systems (IROS)*, 6459–6464.

Tsamardinos, I.; Muscettola, N.; and Morris, P. H. 1998. Fast transformation of temporal plans for efficient execution. In *Proc. of the 15th Nat'l Conf. on Artificial Intelligence (AAAI-98) and 10th Innovative Applications of Artificial Intelligence Conf. (IAAI-98)*, 254–261.

Waibel, M.; Beetz, M.; Civera, J.; D'Andrea, R.; Elfring, J.; Galvez-Lopez, D.; Haussermann, K.; Janssen, R.; Montiel, J.; Perzylo, A.; Schiessle, B.; Tenorth, M.; Zweigle, O.; and van de Molengraft, R. 2011. RoboEarth - A World Wide Web for Robots. *IEEE Robotics Automation Magazine* 18(2):69–82.

Wisspeintner, T.; Van Der Zant, T.; Iocchi, L.; and Schiffer, S. 2009. Robocup@ home: Scientific competition and benchmarking for domestic service robots. *Interaction Studies* 10(3):392–426.

Ziparo, V. A.; Iocchi, L.; Lima, P. U.; Nardi, D.; and Palamara, P. F. 2011. Petri net plans. *Autonomous Agents and Multi-Agent Systems* 23(3):344–383.

# Creating and Using Tools in a Hybrid Cognitive Architecture

**Dongkyu Choi**
Department of Aerospace Engineering
University of Kansas
Lawrence, KS 66045 USA
dongkyuc@ku.edu

**Pat Langley, Son Thanh To**
Institute for the Study of Learning and Expertise
2164 Staunton Court, Palo Alto, CA 94306 USA
patrick.w.langley@gmail.com
son.to@knexusresearch.com

**Abstract**

People regularly use objects in the environment as tools to achieve their goals. In this paper we report extensions to the Icarus cognitive architecture that let it create and use combinations of objects in this manner. These extensions include the ability to represent virtual objects composed of simpler ones and to reason about their quantitative features. They also include revised modules for planning and execution that operate over this hybrid representation, taking into account both relational structures and numeric attributes. We demonstrate the extended architecture's behavior on a number of tasks that involve tool construction and use, after which we discuss related research and plans for future work.

## Introduction

The ability to create and use complex tools is a distinctive feature of human cognition. People use objects in their surroundings to help achieve goals, sometimes combining multiple objects into a new tool that fits their need. This involves planning but focuses on constructing physical artifacts to achieve other ends, rather than generating isolated action sequences. For example, scenes from a popular television series, MacGyver, often depicts the protagonist creating tools from materials that seem unrelated to his objectives. The character ingeniously uses objects in ways for which they were not intended, often combining them into a tool for his purpose. Current AI systems, including our current work, do not demonstrate such creative abilities.

In this paper, we report progress toward intelligent agents that exhibit the ability to create and use physical tools. Our approach extends an existing cognitive architecture to support this capacity. In the next section, we present a scenario that illustrates how tool construction and use can help an agent achieve its goals. Next we briefly review Icarus, an architecture for physical agents, and we describe extensions to its representation and processes that let it create and use tools. After this, we report runs in a simulated environment, drawing on the scenario presented earlier, that demonstrate the revised system's abilities. We conclude by discussing related work and plans for future research.

## A Motivating Scenario

We can clarify the challenge of tool creation and use with a scenario. Consider a robot that wants to escape from inside a crumbled building. Its goal is to move from a location inside the building to another one outside, but between them is a wide gap in the floor that the robot cannot traverse and an opening in the wall that is too high for it to reach without other support. The robot observes some wooden planks of different lengths and thicknesses. Knowing its own weight and the maximum height it can climb, it stacks planks across the gap to build a bridge that will support its weight. The robot then crosses the bridge and thus traverses the gap. In a similar fashion, it builds a staircase to the opening on the wall, goes up the staircase, and escapes from the building.

In this scenario, the robot manipulates objects in its environment and assembles them into tools which it then uses to achieve its goal. To create the right tool, it considers both structural and numeric factors. Wooden planks laid over the gap can serve as a basic bridge, but they must be long enough to cross the gap and strong enough to hold the robot's weight. A single plank may appear qualitatively sufficient, but a second plank may be needed to make the bridge strong enough. For an effective staircase, the building blocks must be arranged to give enough footing on each step and the steps should be no higher than the robot can climb.

We can view bridges and staircases as tools that are constructed from available components. Computing the load a bridge must hold or the height of a step requires quantitative reasoning about objects' positions and dimensions, but the agent must first devise some qualitative structure that its numbers describe. We believe the scenario provides a reasonable challenge for testing an intelligent agents' ability to create and use tools, as it requires a combination of qualitative and quantitative reasoning.

## A Brief Review of Icarus

Icarus (Langley, Choi, and Rogers 2009) is a cognitive architecture that provides an infrastructure for building intelligent agents that operate in physical settings, simulated or actual. As with other architectures like Soar (Laird et al. 1986) and ACT-R (Anderson and Lebiere 1998), it makes commitments about the representation of content, the memories that store that information, and the processes that manipulate it. Icarus incorporates many ideas from cognitive psy-

Table 1: Sample ICARUS concepts for the staircase problem.

```
((on ?o1 ?o2)
 :elements ((block ?o1 ^x ?x1 ^y ?y1 ^length ?length1)
            (block ?o2 ^x ?x2 ^y ?y2 ^length ?length2
                       ^height ?height2))
 :tests    ((*overlapping ?x1 ?length1 ?x2 ?length2)
            (= ?y1 (+ ?y2 ?height2))))

((staircase ?o ?o1)
 :elements   ((block ?o ^height ?h))
 :conditions ((on ?o ?o1)
              (staircase ?o1 ?o2)
              (step-size ?step))
 :tests      ((<= ?h ?step)))
```

Table 2: Sample ICARUS skills for the bridge problem.

```
((pick-up ?o)
 :elements   ((robot ?robot)
              (block ?o))
 :conditions ((clear ?o) (not (holding ?robot ?any)))
 :actions    ((*pick-up ?robot ?o)))
 :effects    ((holding ?robot ?o))

((build-bridge ?block ?bottom)
 :elements   ((block ?block))
 :conditions ((bridge ?top ?bottom))
 :subskills  ((stack ?block ?top))
 :effects    ((bridge ?block ?bottom))
```

chology, but it emphasizes construction of intelligent systems that carry out complex activities rather than fitting the results of psychological experiments. In this section, we review the architecture, starting with assumptions for representation and memories and then describing its mechanisms for inference, reactive execution, and problem solving.

## Representation and Memories

ICARUS distinguishes between two forms of long-term knowledge: *concepts* that underlie inference and procedural *skills* that support activity. The framework also separates *percepts* from the environment from *beliefs* inferred about them. The former describe observed objects in terms of their attributes, typically numeric, while the latter take the form of relational literals like *(on A B)*. This distinction will figure centrally later in the paper. The conceptual knowledge base links percepts to beliefs through a set of defined *concepts*. Each conceptual rule specifies the conditions that must match to infer a belief of a given type. The conditions of a *primitive* concept refer only to percepts and their attribute values, whereas the conditions of a *nonprimitive* concept also refer to more basic conceptual predicates.

Table 1 shows some ICARUS concepts that describe relations and situations for the staircase scenario. The first conceptual rule, for the predicate *on*, is primitive, as it has only an *:elements* field, which describes perceived objects and their attributes, along with a *:tests* field that constrains the matched variables. This concept refers to two *block* objects and checks numeric relations between their positions, lengths, and heights. The second concept, for the predicate *staircase*, is nonprimitive, as it refers to other concepts in its *:conditions* field. These include the concepts like *on*, *step-size*, and *staircase*, so the definition is recursive. Thus, concepts are organized into a hierarchy, with more complex predicates defined in terms of simpler ones.

ICARUS skills build on its conceptual knowledge. Each skill clause includes generalized percepts, conditions that must hold for application, and effects that its application produces. A *primitive* skill clause refers to some action that the agent can execute directly in the environment, whereas a *nonprimitive* skill clause refers to other, more basic, skills.

Table 2 shows examples of ICARUS skills relevant to the bridge problem in our scenario. The first skill clause, *pick-up*, refers to two perceived objects, a *robot* and a *block*, and has two conditions, one positive (for *clear*) and the other negative (for *holding*). This clause is primitive because it includes the executable action **pick-up*. The second skill, *build-bridge*, mentions one percept and one conceptual condition, but it is nonprimitive because it includes the subskill *stack*. Such references organize skills into a hierarchy in which primitive clauses serve as terminal nodes, much as in a hierarchical task network (e.g., Nau et al. 2003).

## Cognitive Processes in ICARUS

The architecture utilizes its concepts and skills during processing, which operates in four-step cycles. First, ICARUS deposits percepts from the environment in a perceptual buffer. The system does not model the extraction of percepts from sensors, but they serve as plausible outputs of sensory processing. Second, the architecture combines its conceptual knowledge with these percepts to infer beliefs that hold for the current situation. ICARUS matches primitive conceptual clauses against perceived objects to generate low-level beliefs, then matches nonprimitive concepts against them to produce higher-level beliefs. For example, the first clause in Table 1 generates a belief about the *on* relation when a block's *y* position equals that of another block plus its height and when the **overlapping* test is true.

Once ICARUS has inferred beliefs about the current situation, an execution stage attempts to find a path downward through the skill hierarchy that it can carry out in the environment. This module starts with a top-level goal, retrieves a skill clause that should achieve it and has conditions satisfied by current beliefs. If this skill instance is primitive, the architecture executes its associated action; if not, then it considers matched subskills. This recursive process returns a path through the skill hierarchy whose execution should bring the agent closer to its goal(s). When ICARUS cannot find such an applicable path, it invokes a problem-solving module that carries out search for sequences of skills which achieve the current goals. Execution and problem solving are tightly interleaved, with the system carrying out selected skill instances when applicable and resorting to problem solving when it encounters an impasse.

We should note that, although ICARUS grounds its concepts and skills in quantitative percepts and actions, the in-

Table 3: An ICARUS concept that illustrates the extended numeric representation.

```
((bridge ?b ?g ?leftend ?rightend)
 :elements    ((block ?b ^x ?leftend ^length ?ln)
               (gap ?g ^left ?gl ^right ?gr))
 :attributes  (?left is (- ?gl 1)
               ?right is (+ ?gr 1)
               ?rightend is (+ ?leftend ?ln))
 :tests       ((<= ?leftend ?left)
               (>= ?rightend ?right)))
```

ference, execution, and problem-solving modules primarily produce qualitative and relational structures. This does not keep the architecture from operating in continuous domains like simulated urban driving (Langley et al. 2009; Choi 2011), but we will see that it raises challenges for the construction and use of complex tools.

## Numeric Representation and Processing

As noted earlier, reasoning about tools often requires that an agent operate over not only qualitative aspects of the environment, but also its quantitative properties. In this section, we discuss two extensions to ICARUS that let the architecture support numeric processing, the first involving representation and the second concerning planning.

### Representational Extensions

ICARUS receives and processes perceptual elements that include types, names, and attribute-value pairs for objects in the world. The original system can represent symbolic relations among objects and concepts can include simple tests on numeric attributes. But it cannot reason about numeric relations or specify arithmetic computations and associate their results with a new variable. In previous research, this limitation has caused problems when using ICARUS to control physical robots, where the continuous domain requires encoding of numeric constraints. Naturally, this issue also arises in tool creation and use. To address the problem, we extended the conceptual formalism to specify arithmetic combinations of numeric attributes and associate them with new variables that can appear elsewhere in the concept.

Table 3 shows a sample concept that uses this extended notation. The clause includes a new field, *:attributes*, that specifies desired numeric calculations and their variable assignments. This specific clause states that the position of a block's right side (denoted by the variable *?rightend*) can be computed from its left side position, *?leftend*, and its length, *?ln*. The concept also specifies how to compute the left and right positions, *?left* and *?right*, for a spatial gap with one unit margins at both ends. These values are also used, along with the left and right ends, in two inequality tests.

This extension lets ICARUS specify numeric calculations and how to reuse their results elsewhere in a conceptual clause, complementing the qualitative structures it could already express. However, this only describes the environmental situation, not how agent's actions will alter it. In response, we also extended the notation for skills to incorporate details about quantitative effects of their execution.

Table 4: An ICARUS skill for creating a bridge that illustrates the extended numeric formalism.

```
((fill-gap-center ?b ?g)
 :elements   ((block ?b ^x ?x0 ^length ?l ^weight ?w)
              (robot ?robot ^weight ?weight
                            ^status ?status ^holding ?b)
              (gap ?g ^left ?gl ^right ?gr))
 :actions    ((*fill-gap-center ?robot ?b ?gl ?gr))
 :effects    ((bridge ?b ?g ?x0 (+ ?x0 ?l))
              (block ?b ^x (/ (- (+ ?gl ?gr) ?l) 2) ^y 0
                        ^len ?l ^weight ?w)
              (robot ?robot ^weight (- ?weight ?w)
                     ^status ?status ^holding nothing)))
```

Table 4 shows a skill that takes advantage of this extension. The main change is in the *:effects* field, which describes the outcome of a skill's successful execution. Previously, this field could only include symbolic effects about relational beliefs that would become true or false after application. In the new notation, the field can describe expected changes not only in symbol structures, but also in the numeric attributes of objects. The skill will not only cause the symbolic relation *(bridge ...)* to become true, but also change the block's *x* position to the value of the expression, *(/ (- (+ ?gl ?gr) ?l) 2)*, and reduce the robot's *weight* by *?w*.

### Extensions to Processing

The original architecture could match against numeric attributes of perceived objects, but it could neither perform mathematical calculations over these numbers nor allow the results in concept heads. The representational changes to concepts and skills remedies these limitations, but taking advantage of them also required us to augment ICARUS's information processing along two fronts. The first deals with inference, which now calculates the values of arithmetic expressions in concepts and binds them to specified variables that may appear in the heads. These numeric values, in turn, can influence inference of symbolic beliefs at higher levels, as they are carried upward through the hierarchy during the conceptual inference process.

These changes to the formalism require no alteration of the execution module, but they do necessitate changes to problem solving. In response, the revised module computes not only symbolic relations during its mental execution of skills but also numeric values associated with them. The new problem solver utilizes forward chaining, which lets the system update numeric attributes of an object, add new literals, or delete existing literals from the state using information encoded in skills' *:effects* field. Such mental execution has direct effects on the projected state, but indirect changes can also occur, which the architecture determines by invoking the inference module. As a result, the problem solver can generate plans that satisfy both symbolic and numeric requirements specified in the agent's goals.

## Encoding and Processing Virtual Objects

Despite its new ability to reason about quantitative aspects of the environment, the extended ICARUS still cannot recognize an existing object as a potential tool or reason about how to create one from available elements. This is because the architecture only recognizes primitive objects as distinct entities, not combinations of them. In contrast, people readily view composite structures as objects themselves, describe numeric features associated with them, and reason about them as unified entities. To create and utilize complex tools, ICARUS needs the ability to reify and process such *virtual* objects in its environment.

### Representational Extensions

The ability to include numeric attributes in concept heads paves the way to handling virtual objects. Without this extension, the architecture can infer beliefs only as symbolic literals, which makes them different from perceived objects in that they lack numeric attributes. Previously, for example, a *bridge* concept that describes a composite object could only produce a symbolic belief that informs the agent about its existence. In contrast, the new version can calculate the values for numeric attribute associated with the *bridge* entity, such as its thickness and weight limit.

However, computing such numeric attributes is not enough. We also need some way to associate them with the virtual object, which requires giving it a symbolic identifier in the same manner as percepts. This extension effectively eliminates the distinction in the original ICARUS between beliefs and percepts, so the new architecture stores them in a single working memory. The only remaining differences are that percepts come directly from an external environment, while beliefs are inferred, and that beliefs include a symbolic relation, while percepts lack them. Of course, we can apply this idea recursively to specify higher-level virtual objects in terms of lower-level ones.

For example, the two conceptual clauses for *bridge* that appear in Table 5 not only describe the class of *situations* in which one or more blocks cover a gap, but also specify a new *virtual object* that denotes the bridge. This composite object has its own attributes, such as its left position, right position, and weight, the values of which are calculated from the attribute values of its component objects.

### Implications for Processing

Once the extended ICARUS has created virtual objects, it can use them as if they were objects perceived directly in the environment. The second, recursive, clause for *bridge* concept shown in Table 5 lets the system recognize situations in which a block is stacked on a bridge and generate another virtual object that is also a *bridge*, but one with a higher weight limit than the original one.

As the table shows, the new notation also changes the syntax for the `:elements` field. Here the expression *A is B* states that one should associate an identifier *A* with *B*, which may be a percept or a relational belief. Recall that percepts enter the perceptual buffer with such identifiers, but

Table 5: Some ICARUS concepts that specify virtual objects.

```
((bridge ?b ^gap ?gl ^left ?l ^top-left ?tl
          ^top-right ?tr ^right ?r ^weight ?weight)
 :elements (?b is (block ?b ^x ?tl ^y 0 ^len ?len
                         ^weight ?weight)
           ?gl is (gap ?gl ?gr))
 :tests    ((<= ?tl (- ?gl 1))
           (>= (+ ?tl ?len) (+ ?gr 1)))
 :attributes (?l is ?tl
              ?tr is (+ ?tl ?len)
              ?r is (+ ?tl ?len)))

((bridge ?b ^gap ?gl ^left ?l ^top-left ?tl
          ^top-right ?tr ^right ?r ^weight ?weight)
 :elements (?b is (block ?b ^x ?tl ^y ?y ^len ?len
                         ^weight ?w)
           ?b1 is (bridge ?b1 ^gap ?gl ^left ?l
                       ^top-left ?tl1 ^top-right ?tr1
                       ^right ?r ^weight ?w1)
           ?b1 is (block ?b1 ^x ?tl1 ^y ?y1
                       ^len ?len1 ^weight ?w2)
           ?gl is (gap ?gl ?gr))
 :tests    ((<= ?tl (- ?gl 1))
           (>= (+ ?tl ?len) (+ ?gr 1))
           (= (+ ?y1 1) ?y)
           (<= (+ ?tl1 1) ?tl)
           (<= (+ ?tl ?len 1) ?tr1))
 :attributes (?weight is (+ ?w ?w1)
              ?tr is (+ ?tl ?len)))
```

that ICARUS must name its beliefs before it can associate numeric attributes with them. The extended architecture retains the identifiers for these virtual objects in working memory, so they can appear as arguments in higher-level beliefs that result from conceptual inference.

What we have described suffices for ICARUS to draw inferences about composite objects, but not to use them for driving agent activity. Of course, virtual objects can also appear in the *effects* field of skills, which means that the problem solver can form expectations about their creation or destruction upon execution. This means, for example, that the agent can use its hierarchical skills to form plans that involve constructing composite objects which enable later steps that achieve its goals. But it can also use search to generate plans entirely from primitive skills and, by invoking the inference process, deduce that an action sequence has the side effect of creating a complex virtual object that it can use as a tool.

## Demonstrations of the Extended Architecture

To confirm that the extended system behaves as intended, we carried out demonstration runs on the scenario described earlier. Here an ICARUS agent controls a simulated mobile robot to reach its destination. In one case, there is a chasm between the initial and the goal location; in another problem, the goal is at a higher location than the robot can reach directly. In both cases, the agent can use blocks of different sizes to build a bridge or staircase, which it can then use.

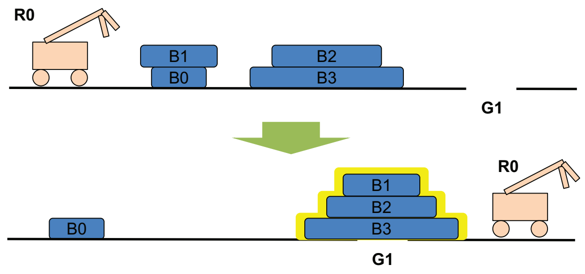


Figure 1: Initial and final states for one version of the bridge problem. The robot, *R0*, starts on the lefthand side and must use blocks to build a bridge over the gap, *G1*, to reach its goal on the righthand side.

## Simplifying Assumptions

The primary aim of these demonstration runs was to show that the extensions to ICARUS, described earlier, support the creation and use of tools. For this reason, we introduced four simplifying assumptions that made the planning and execution tasks somewhat easier than they would be in a realistic simulation:

- Although ICARUS allows durative skills that require repeated application to achieve their effects, in the runs all skills produce results in one step;
- The 2D simulated environments let agents pick up and stack objects without first needing to approach them or to move around obstacles;
- Agents must use planning to find a sequence of skills that construct composite objects that can serve as tools, but skills for using them operate in one step; and
- We provided agents with hierachical concepts for tools that appear as conditions on these tool-using skills, effectively serving as affordances (Zech et al. 2017).

Ideally, future demonstrations should use more realistic simlated environments that eliminate these assumptions. Nevertheless, the reported runs offer clear proof of concept that the extended architecture can represent, reason about, construct, and use tools to achieve goals in continuous settings.

## Creating and Traversing a Bridge

In the first setting, the robot must build a bridge to cross the chasm, using long wooden blocks of different lengths and strengths. The agent knows that, for the robot to traverse the bridge safely, it must: (1) cover the chasm by a margin of at least a foot at each end; (2) withstand the robot's weight and any payloads; and (3) if it is made from stacked blocks, include a staircase at each end with steps no higher than a foot and at least a foot wide. The agent has no skill that directly creates a bridge, so it must use problem solving to find some plan to build one that satisfies these requirements. The system must then execute this plan, building the bridge in the environment and crossing the chasm to reach its destination.

For this problem, we gave ICARUS four concepts and four

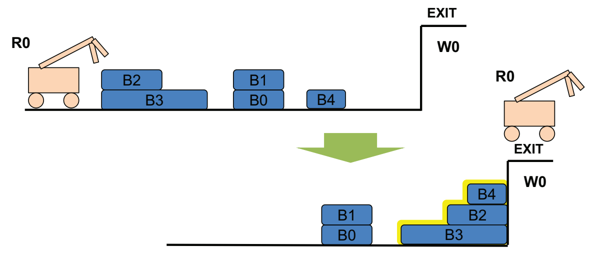


Figure 2: Initial and final states for one version of the staircase construction and climbing problem.

primitive skills, including the ones shown in Tables 4 and 5. Using this knowledge, the agent can recognize situations in which a block is stacked on another, detect a bridge composed of blocks, pick up a block to either stack it on another or cover a chasm, and finally cross the bridge when it is complete. Figure 1 shows an initial state in which the robot perceives itself, a chasm, and four blocks that are two, four, six, and eight units in length and that have weight limits of one, five, one, and two, respectively. Block *B1* is on block *B0* and block *B2* is on block *B3*.

Given these initial and goal states, the problem solver uses forward-chaining search to find a plan that achieve its goal in nine steps. During this process, ICARUS first considers a bridge that only withstands a weight of two units, which is insufficient for the robot to cross. Next the system considers stacking a second block on the first to create a bridge with the maximum load of three units. This is still not sufficient, so it stacks yet another block, making a bridge that is strong enough for it to cross the chasm safely.

Once it has found this plan, the ICARUS agent executes it in the simulated environment over 29 cycles, first picking up *B2* to clear *B3* and stacking *B2* on *B1*. Next the system picks up the longest block *B3* and covers the gap with it. Then the robot picks up another block, *B2*, and stacks it on *B3* to create a stronger bridge, after which it stacks *B1* on the result to make it even stronger. At this point, the robot traverses the reinforced bridge to reach its goal.

We ran the extended architecture on 20 similar problems that involved four blocks of random lengths and weight limits. The system executed plans that had the average duration of 29.6 cycles with a standard deviation of 10.7 cycles. We also ran it, with the same knowledge, on a slightly different goal description in which the robot must carry a certain block as its payload across the chasm. In this altered scenario, the ICARUS agent generated a similar plan, this time requiring that it construct an even stronger bridge, then pick up the payload for delivery. Again, the robot executed this plan in the simulated world to achieve its goal.

## Constructing and Climbing a Staircase

In the second scenario, the robot must escape from a room in which the exit is higher on the wall than it can reach without assistance. The environment contains long wooden blocks of different lengths that the agent can use to build a staircase for reaching the exit. The system knows that a staircase must: (1) have steps that are no taller than a foot for the robot to climb successfully and at least a foot wide so it can step on them safely; (2) be no further than a foot from the wall at its highest point; and (3) have a height that is within a foot of the exit's height. The robot must build a staircase that satisfies all these requirements before it can ascend and exit the room.

For this problem, we provided ICARUS with seven concepts and four primitive skills. The robot could use this knowledge to recognize situations in which one block is on top of another, categorize a virtual object as a staircase, pick up a block to either stack it on another or place it on the ground, and leave the room when it reaches the exit. Figure 2 shows one example of this scenario in which the robot perceives itself, the wall, and five blocks with lengths of 1.5, 1.5, 3, 4.5, and 1, respectively, and with heights of one unit.

The problem solver uses forward search to generate a plan that, in 13 steps, achieves the exit goal. During planning, ICARUS mentally constructs a staircase from three blocks that will let it leave the room, but only after considering shorter stairways. Once it has found this plan, the robotic agent executes it in the simulated environment, which takes 41 cognitive cycles. This involves picking up block *B4* to clear the area around the wall and stacking it on block *B1*. The agent then picks up block *B2* to clear *B3* and stacks *B2* on *B4*. The robot continues stacking the blocks *B3*, *B2*, and *B4*, in that order. At this point, it recognizes that it has built an acceptable staircase, so the robot climbs the stairs and exits the room, achieving its goal.

As another demonstration run, we used a variation on this problem that required the system to combine a number of shorter blocks to form steps for the staircase. This involved generating a more complex plan with additional steps that led to more virtual objects, greater search during planning, and longer execution times than in the first run, but the system handled them without any special difficulty.

In summary, the runs have demonstrated that the extended architecture can represent and reason about numeric attributes and virtual objects during inference, problem solving, and execution. This lets the revised ICARUS infer beliefs that incorporate numeric attributes, associate them with composite entities that its actions produce, and use this content to generate and carry out plans that achieve symbolic goals subject to numeric constraints. Together, these abilities support the construction of tools, such as bridges and staircases, from available components and their use once built.

## Related Research

The extensions to ICARUS that let it create and use tools have clear precedents that merit discussion. We focus here on two contributions that we consider most important – reasoning over numeric attributes and using virtual objects. We have discussed the architecture's forward-chaining planning module elsewhere (To et al. 2015). We will not repeat our observations here except to note that it can use primitive skills, hierarchical ones, and their combination to generate plans, although the first option requires more search.

Research in cognitive architectures (Langley, Laird, and Rogers 2009) has emphasized symbolic representation and processing, due to their focus on high-level cognitive tasks. Nevertheless, well-established frameworks like Soar and ACT-R adopt an attribute-value notation that can easily encode the types of numeric object-based inputs we assume in both working memory and production rules. Both architectures have been used to control robotic agents, which certainly requires quantitative processing. However, they treat numeric manipulation as a special case of symbol processing, rather than giving them equal status, at the architecture level, as does the extended version of ICARUS.

Other paradigms also support a combination of symbolic and numeric processing. For example, logic programming emphasizes symbolic notations but can incorporate quantitative values and constraints, although they do not typically operate over time, as do ICARUS agents. AI planning systems also focus on symbolic tasks but have been adapted to include numeric content (e.g., Coles et al. 2012). These describe activity over time, but work in this tradition seldom supports the storage and use of hierarchical skills. Most robotic systems emphasize low-level numeric processing to the exclusion on high-level cognition. Hybrids like the 3T architecture (Bonasso et al. 1997) support both, but they adopt separate, specialized notations rather than offering a unified framework for cognition and action. Perhaps the closest robotics work (Levihn and Stilman 2014; Erdogan and Stilman 2014), also concerned with tool creation, propagates physical constraints to ensure a symbolic planner considers only acceptable configurations of objects.[1]

As for the virtual objects, most production-system architectures (e.g., Klahr, Langley, and Neches 1987) support rules that introduce new symbols, with associated attribute values, in elements they add to working memory. However, they do not elevate their creation to the architectural level or make theoretical claims about the way such objects are defined, processed, and used by other mechanisms. Our extended framework associates virtual objects with concept instances that reside in belief memory, so that any conceptual rule in long-term memory can generate them during the inference process. This allows a tight integration with other components of the ICARUS architecture.

Otherwise, the paradigm most relevant to our use of virtual objects is scene understanding (e.g., Antanas et al. 2012), which attempts to infer models of the environment from images or videos. Classical approaches construct a hierarchy of entities, from edges to angles to surfaces to 3D object models (Binford 1982). ICARUS' virtual objects are directly analogous to these intermediate entities, and its calculation of derived attribute values maps directly on computations of angles and volumes in vision systems. However, work in this paradigm has focused on scene interpretation, not with goal-directed activity. Thus, although such systems might be able to describe and recognize tools like bridges and stairs, they cannot use them to achieve objectives.

[1]Brown and Sammut (2012) report a novel approach to learning tool usage by the analysis of training cases, but their research has different aims than our own.

## Plans for Future Work

We have shown that the extended ICARUS can represent and reason about tools, it can construct such tools from available objects, and it can then use them to achieve its goals. Nevertheless, we must still address a number of challenges that our work to date has left unexamined. The most obvious limitations involve the system's dependence on handcrafted knowledge about composite tools.

ICARUS already includes mechanisms for learning hierarchical skills from successful problem solving (Langley et al. 2009), and we can use this ability to acquire structures for constructing bridges, staircases, and similar artifacts, as well as ones for using them after they have been created. The latter will be useful in more realistic environments that require sequences of actions for tool use, such as taking repeated steps up a staircase. These mechanisms acquire new skills from individual solutions obtained through search, so learning can be very rapid.

A more challenging hurdle involves the acquisition of concepts that recognize composite tools. Here we plan to draw on another extension to ICARUS (Li et al. 2012) that, when it uses a problem solution to create a new skill, also defines a new conceptual predicate that describes the conditions under which that skill will achieve the relevant goals. These conceptual rules may be disjunctive or even recursive, so the mechanism should be able to produce concepts for recognizing bridges, staircases, and other tools that may have arbitrary numbers of components.

However, we can best take advantage of this ability by separating the issues of tool construction and tool use. If we present an ICARUS agent with a problem that it can solve with an existing configuration of objects, say two blocks that cover a gap, it could learn both a hierarchical skill for using that configuration and a concept that recognizes similar 'bridge' configurations in the future. Given such knowledge, it could then solve, and learn from, new problems that require the construction of a bridge before its traversal. This decomposition is not strictly necessary, but inventing the bridge concept from scratch would require more search than determining how to build one after having used another.

These are certainly not the only challenges that remain before we have a mature account of tool construction and use. For instance, numeric simulation of durative operators, as in Langley et al.'s (2016) PUG architecture, seems relevant to determining whether an agent can use a tool to achieve its goals. The ability to interleave planning, execution, and monitoring is also important in settings where tools are not fully reliable. However, the creation and use of tools is one of the distinguishing features of human intelligence, so we should not be surprised that many open problems remain.

## Concluding Remarks

In this paper, we reported extensions to the ICARUS architecture that support the creation and use of tools. These included the ability to associate numeric attributes with concepts and skills, as well as calculate their values during inference, execution, and problem solving. Another augmentation let conceptual rules refer to new, complex objects that were composed from existing ones and to derive values for their numeric attributes during the process of conceptual inference. Together, these capabilities let the extended architecture not only represent and reason about tools it creates from components available in the environment, but also use those tools to achieve its goals.

We demonstrated this new functionality in two simulated environments, one that involved creating and traversing a bridge and another that required constructing and climbing a staircase. We will not claim that other approaches, such as AI planning methods, cannot handle the same tasks, but they would not represent or recognize the fact that tools played a key role in their solutions. Humans clearly exhibit this ability, and we believe that ICARUS' approach to tool creation and use has many similarities. Nevertheless, we have taken only the first steps, and future work should include demonstrations in more realistic environments and use of learning mechanisms to acquire tool-related concepts and skills.

## Acknowledgements

We dedicate this work in memory of Mike Stilman, a friend and colleague who inspired us to bridge the gap between robotics and cognitive systems. This research was supported by Grants N00014-12-1-0143 and N00014-15-1-2517 from the Office of Naval Research.

## References

Anderson, J. R.; and Lebiere, C. 1998. *The atomic components of thought*. Mahwah, NJ: Erlbaum.

Antanas, L.; Frasconi, P.; Costa, F.; Tuytelaars, T.; and Raedt, L. D. 2012. A relational kernel-based framework for hierarchical image understanding. In G. Gimel'farb et al., Eds., *Structural, syntactic, and statistical pattern recognition*, 171–180. Berlin: Springer.

Binford, T. O. 1982. Survey of model-based image analysis systems. *International Journal of Robotics Research* 1:18–64.

Bonasso, R. P.; Firby, R. J.; Gat, E.; Kortenkamp, D.; Miller, D. P.; and Slack, M. G. 1997. Experiences with an architecture for intelligent, reactive agents. *Journal of Experimental & Theoretical Artificial Intelligence* 9:237–256.

Brown, S.; and Sammut, C. 2013. A relational approach to tool-use learning in robots. In F. Riguzzi and F. Elezn, Eds., *Inductive Logic Programming*, 1–15. Berlin: Springer.

Choi, D. 2011. Reactive goal management in a cognitive architecture. *Cognitive Systems Research* 12:293–308.

Coles, A. J.; Coles, A. I.; Fox, M.; and Long, D. 2012. COLIN: Planning with continuous linear numeric change. *Journal of Artificial Intelligence Research* 44:1–96.

Erdogan, C.; and Stilman, M. 2014. Incorporating kinodynamic constraints in automated design of simple machines. *Proceedings of the 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2931–2936. Chicago: IEEE Press.

Fikes, R.; and Nilsson, N. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2:189–208.

Klahr, D.; Langley, P.; and Neches, R. Eds. 1987. *Production system models of learning and development*. Cambridge, MA: MIT Press.

Laird, J. E.; Newell, A.; and Rosenbloom, P. S. 1987. Soar: An architecture for general intelligence. *Artificial Intelligence* 33:1–64.

Langley, P.; Barley, M.; Meadows, B.; Choi, D.; and Katz, E. P. 2016. Goals, utilities, and mental simulation in continuous planning. *Proceedings of the Fourth Annual Conference on Cognitive Systems*. Evanston, IL.

Langley, P.; Choi, D.; and Rogers, S. 2009. Acquisition of hierarchical reactive skills in a unified cognitive architecture. *Cognitive Systems Research* 10:316–332.

Langley, P., Laird, J. E., and Rogers, S. 2009. Cognitive architectures: Research issues and challenges. *Cognitive Systems Research* 10:141–160.

Levihn, M.; and Stilman, M. 2014. Using environment objects as tools: Unconventional door opening. *Proceedings of the 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2502–2508. Chicago: IEEE Press.

Li, N., Stracuzzi, D. J., and Langley, P. 2012. Improving acquisition of teleoreactive logic programs through representation extension. *Advances in Cognitive Systems* 1:109–126.

Nau, D. S.; Au, T.-C.; Ilghami, O.; Kuter, U.; Murdock, J. W.; Wu, D.; and Yaman, F. 2003. SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research* 20:379–404.

To, S. T.; Langley, P.; and Choi, D. 2015. A unified framework for knowledge-lean and knowledge-rich planning. *Proceedings of the Third Annual Conference on Cognitive Systems*. Atlanta, GA.

Zech, P.; Haller, S.; Lakani, S. R.; Ridgeand, B.; Ugur, E.; and Piater, J. 2017. Computational models of affordance in robotics: A taxonomy and systematic classification. *Adaptive Behavior* 25:235–271.

# Robot Behavioral Exploration and Multi-Modal Perception Using Dynamically Constructed Controllers

**Saeid Amiri,[1] Suhua Wei,[1] Shiqi Zhang,[1] Jivko Sinapov,[2] Jesse Thomason,[3] Peter Stone[3]**
[1] Department of Electrical Engineering and Computer Science, Cleveland State University
[2] Department of Computer Science, Tufts University
[3] Department of Computer Science, The University of Texas at Austin
{s.amiri@vikes; s.wei@vikes; s.zhang9@}.csuohio.edu; jsinapov@cs.tufts.edu
t{jesse; pstone}@cs.utexas.edu

**Abstract**

Intelligent robots frequently need to explore the objects in their working environments. Modern sensors have enabled robots to learn object properties via perception of multiple modalities. However, object exploration in the real world poses a challenging trade-off between information gains and exploration action costs. Mixed observability Markov decision process (MOMDP) is a framework for planning under uncertainty, while accounting for both fully and partially observable components of the state. Robot perception frequently has to face such mixed observability. This work enables a robot equipped with an arm to dynamically construct query-oriented MOMDPs for object exploration. The robot's behavioral policy is learned from two datasets collected using real robots. Our approach enables a robot to explore object properties in a way that is significantly faster while improving accuracies in comparison to existing methods that rely on hand-coded exploration strategies.

## 1 Introduction

Service robots are increasingly present in everyday environments, such as homes, offices, airports, and hospitals, where a common task is to retrieve an object for a user. Consider the request, "*Please fetch me the red, empty bottle*." A key problem for the robot is to decide whether a particular candidate object matches the properties in the query. For certain words (e.g., *heavy*, *soft*, etc.), visual classification of the object is insufficient as the robot would need to perform an action (e.g., lift the object to determine whether it is heavy or not). Multi-modal perception research has focused on combining information arising from such multiple sensory modalities.

Given multi-modal perception capabilities, a robot needs to decide which actions (possibly out of many) to perform on an object, i.e., generate a behavioral policy for a given request. For instance, to obtain an object's color, a robot simply needs to adjust the pose of its camera, whereas sensing the content of a container requires two actions: grasping and shaking. The robot needs to select actions in such a way that the information gain about object properties is maximized while the cost of actions is minimized. It should be noted that the robot needs to use sequential reasoning in this action selection process, e.g., a shaking action would make sense only if a grasping action has been (successfully) executed. Also, robot perception capabilities are imperfect, so the robot sometimes needs to take the same action more than once. Probabilistic planning algorithms aim at computing action policies to help select actions toward maximizing long-term utility (information gain in our case), while considering the uncertainty in non-deterministic action outcomes.

Markov decision processes (MDPs) (Puterman 1994) and partially observable MDPs (POMDPs) (Kaelbling, Littman, and Cassandra 1998) enable an agent to plan under uncertainty with full and partial observability respectively. However, the observability of real-world domains is frequently mixed: some components of the current state can be fully observable while others are not. A mixed observability Markov decision process (MOMDP) is a special form of POMDP that accounts for both fully and partially observable components of the state (Ong et al. 2010). In this work, we model robot multi-modal perception problems using MOMDPs because of the mixed observability of the world that the robot interacts with (e.g., whether an object is in hand or not is fully observable, but object properties such as color and weight are not). Referring to our model as a MOMDP (as opposed to a POMDP) is not of practical importance in this paper. It is mainly for ease of describing the domain.

Robot behavioral exploration policies are learned from the experience of a robot interacting with objects in the real world. We use datasets that include tens of objects and nearly one hundred properties. In such domains, it frequently takes a prohibitively long time to compute effective behavioral exploration policies. To tackle this issue, we dynamically construct MOMDP-based controllers to model a minimum set of domain variables that are relevant to current user queries (e.g. "red, empty bottle"). This strategy ensures a small state set and enables us to generate high-quality robot action policies in a reasonable time (e.g., $\leq 2$ seconds). Our experiments show that the policies of the constructed controllers improve recognition accuracy and reduce exploration cost when compared to baseline strategies that deterministically or randomly use predefined sequences of actions.

## 2 Related Work

Recent research in robotics has shown that robots can learn to classify objects using computer vision methods as well as non-visual perception coupled with actions performed on the objects (Högman, Björkman, and Kragic 2013; Sinapov et al. 2014; Thomason et al. 2016). For example, a robot can learn to determine whether a container is full or not based on the sounds produced when shaking the container (Sinapov and Stoytchev 2009); or learn whether an object is soft or hard based on the haptic sensations produced when pressing it (Chu et al. 2015). Past work has shown that robots can associate (or *ground*) these sensory perceptions with human language predicates in vision space (Alomari et al. 2017; Whitney et al. 2016; Krishnamurthy and Kollar 2013; Matuszek et al. 2012) and joint visual and haptic spaces (Gao et al. 2016).

Nevertheless, there has been relatively little emphasis on enabling a robot to *efficiently* select actions at test time when it is tasked with classifying a new object. The few approaches for tackling action selection, e.g., (Rebguns, Ford, and Fasel 2011; Fishel and Loeb 2012; Sinapov et al. 2014), assume that only one target property needs to be identified (e.g., the object's identity in the case of object recognition). In contrast, we address the problem where a robot needs to recognize multiple properties about an object, e.g., "is the object a *red empty bottle*?".

Sequential decision-making frameworks, such as MDPs, POMDPs and MOMDPs, can be used for probabilistic planning toward achieving long-term goals, while accounting for non-deterministic action outcomes and different observabilities (Kaelbling, Littman, and Cassandra 1998; Ong et al. 2010). As a result, these frameworks have been applied to object exploration in robotics. For instance, POMDPs were used for suggesting visual operators and regions of interests for exploring multiple objects on a tabletop scenario (Sridharan, Wyatt, and Dearden 2010), and more recent work used a robotic arm to move objects enabling better visual analysis (Pajarinen and Kyrki 2015). However, interaction with objects in these lines of research relies heavily on robot vision while other sensing modalities, such as audio and haptics, are not considered.

Behavioral policies of multi-modal object exploration have been learned in simulation using deep reinforcement learning methods (Denil et al. 2017), where *force* was directly used in the interactions with objects. The simulation environment used in that work makes it possible to run large numbers of trials, but limits its applicability on real robots.

## 3 Theoretical Framework

Next, we describe the theoretical framework used by the robot to learn predicate recognition models and generate efficient policies when tasked with identifying whether a set of predicates hold true for a new object.

### 3.1 Multi-Modal Predicate Learning

In this work, the robot learns predicate recognition models using the methodology described in (Sinapov, Schenck, and Stoytchev 2014; Thomason et al. 2016), briefly summarized here. In this methodology, the robot uses behaviors (e.g., *look*, *grasp*, *lift*) coupled with sensory modalities (e.g., *color*, *haptics*, *audio*) to identify whether a predicate (i.e., a word that a human may use to describe an object) holds true for an object.

Let $\mathcal{P}$ be the set of predicates, let $\mathcal{B}$ be the set of behaviors (i.e., actions), and let $\mathcal{C}$ be the set of sensorimotor contexts, where each context $c \in \mathcal{C}$ corresponds to a combination of a behavior and sensory modality (e.g., *look-color*, *lift-haptics*). For each predicate $p$, and context $c$, the robot learns a classifier using data points $[x_i^c, y_i]$, where $x_i^c$ is the $i^{th}$ observation feature vector in context $c$, and $y_i = true$ if the predicate $p$ holds true for the object in trial $i$, and $false$ otherwise.

Let $\mathcal{C}_b \subset \mathcal{C}$ be the set of sensorimotor contexts associated with behavior $b \in \mathcal{B}$. When executing action $b$, the robot queries the classifiers associated with contexts $\mathcal{C}_b$ and combines their outputs to estimate a score (normalized in the range of 0.0 to 1.0) for each predicate $p \in \mathcal{P}$. In other words, each behavior acts as a classifier itself. At the end of the training stage, the robot performs internal cross-validation and stores the confusion matrix $C_p^b \in \mathbb{R}^{2\times 2}$ for predicate $p$ and behavior $b$. Next, we describe the problem of generating an action policy when identifying whether a set of predicates hold true for an object that was not present during training.

### 3.2 MOMDP-based Controllers

Behaviors (or actions[1]), such as *look* and *drop*, have different costs and different accuracies in predicate recognition. At each step, the robot has to decide whether more exploration behaviors are needed, and, if so, select the exploration behavior that produces the most information. In order to sequence these behaviors toward maximizing information gain, subject to the cost of each behavior (e.g., the time it takes to execute it), it is necessary to further consider preconditions and non-deterministic outcomes of the actions. For instance, *shaking* and *dropping* actions make sense only if a preceding *grasping* action succeeds; and, in practice, *grasping* actions are unreliable and succeed with probability.

In this work, we assume action outcomes are fully observable and object properties are not. For instance, a robot can reliably sense whether a *grasping* action is successful, but it cannot reliably sense the color of a bottle or whether that bottle is full. Due to this mixed observability and unreliable action outcomes, we use mixed observability MDPs (MOMDPs) (Ong et al. 2010) to model the sequential decision-making problem for object exploration. We next present how we formalize our object exploration problem within the MOMDP framework.

A MOMDP is fundamentally a factored POMDP with mixed state variables. The fully observable state components are represented as a single state variable $x$ (in our case, the *robot-object status*, e.g., the object is in hand or not), while the partially observable components are represented as state

[1]The terms of "behavior" and "action" are used interchangeably in this paper.

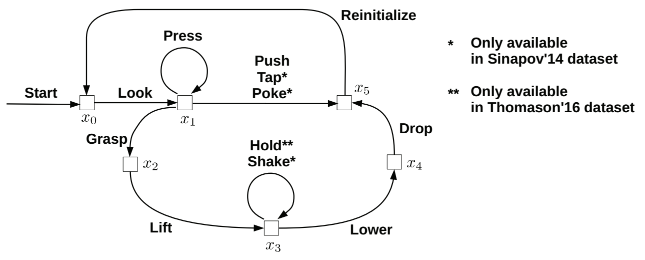


Figure 1: A simplified version of the transition diagram in space $\mathcal{X}$ for object exploration. This figure only shows the probabilistic transitions led by *exploration actions*. *Report actions* that deterministically lead transitions from $x_i \in \mathcal{X}$ to the *term* state are not included.

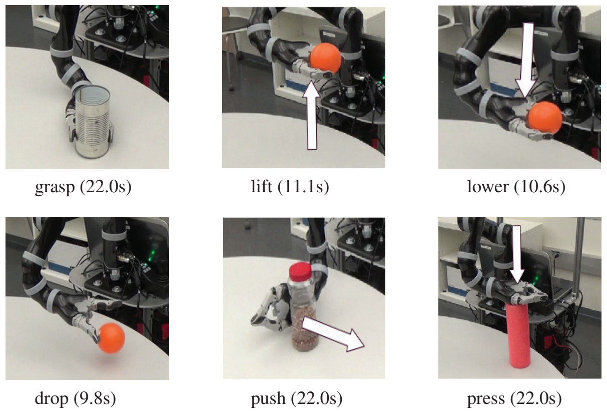


Figure 2: The behaviors, and their durations in seconds (behaviors are from the **Thomason16** dataset detailed in Sec. 4). In addition, the *hold* (1.0s) behavior was performed by holding the object in place. The *look* (0.5s) behavior was also performed by taking a visual snapshot of the object using the robot's sensors prior to exploration.

variable $y$ (in our case, the *object properties*, e.g., the object is heavy or not). As a result, $(x,y)$ specifies the complete system state, and the state space is factored as $S = \mathcal{X} \times \mathcal{Y}$, where $\mathcal{X}$ is the space for fully observable variables and $\mathcal{Y}$ is the space for partially observable variables.

Formally, a MOMDP model is specified as a tuple,

$$(\mathcal{X},\ \mathcal{Y},\ A,\ T_{\mathcal{X}},\ T_{\mathcal{Y}},\ R,\ Z,\ \mathcal{O},\ \gamma),$$

where $A$ is the action set, $T_{\mathcal{X}}$ and $T_{\mathcal{Y}}$ are the transition functions for fully and partially observable variables respectively, $R$ is the reward function, $Z$ is the observation set, $\mathcal{O}$ is the observation function, and $\gamma$ is the discount factor.

The definitions of $A$, $R$, $Z$, $\mathcal{O}$, and $\gamma$ of a MOMDP are identical to these of POMDPs (Kaelbling, Littman, and Cassandra 1998), except that $Z$ and $\mathcal{O}$ are only applicable to $\mathcal{Y}$, the partially observable components of the state space. $\gamma$ is the discount factor that specifies the planning horizon. We formalize our object exploration problem as a MOMDP (as a special form of POMDP) mainly for ease of describing the fully and partially observable variables in our domain.

Next, we present how each component of our MOMDP model is specified for our object exploration problem.

## 3.3 State Space Specification

The state space of our MOMDP-based controllers has two components of $\mathcal{X}$ and $\mathcal{Y}$. The global state space $S$ includes a Cartesian product of $\mathcal{X}$ and $\mathcal{Y}$,

$$S = \{(x,y) \mid x \in \mathcal{X} \text{ and } y \in \mathcal{Y}\}$$

$\mathcal{X}$ is the state set specified by fully observable domain variables. In our case, $\mathcal{X}$ includes a set of six states $\{x_0, \cdots, x_5\}$, as shown in Figure 1, and a terminal state *term* $\in \mathcal{X}$ that identifies the end of an episode. $x \in \mathcal{X}$ is fully observable, and the robot knows the current state of the robot-object system, e.g., whether grasping and dropping actions are successful or not.

$\mathcal{Y}$ is the state set specified by partially observable domain variables. In our case, these variables correspond to $N$ object properties that are queried about, $\{v_0,\ v_1,\ \cdots,\ v_{N-1}\}$, where the value of $v_i$ is either *true* or *false*. Thus, $|\mathcal{Y}| = 2^N$.

For instance, given an object description that includes three properties (e.g., "a *red empty bottle*"), $\mathcal{Y}$ includes $2^3 = 8$ states. Since $y \in \mathcal{Y}$ is partially observable, it needs to be estimated through observations. It should be noted that there is no state transition in the space of $\mathcal{Y}$, as we assume object properties do not change over the course of robot action.

## 3.4 Actions and Transition System

We present the transition system of our MOMDP-based controllers by first introducing the action set and then the transition probabilities. $A : A^e \cup A^r$ is the action set. $A^e$ includes the object *exploration* actions pulled from the literature of robot exploration, as shown in Figure 1, and $A^r$ includes the *reporting* actions used for object property identification.

**Exploration actions:** Figure 1 shows all exploration actions except for action *ask* that is allowed in any state $x \in \mathcal{X}$. Among the actions, *tap*, *poke*, and *shake* are only available in the dataset of (Sinapov, Schenck, and Stoytchev 2014) and *hold* is only available in the dataset of (Thomason et al. 2016). As one of the main contributions, our approach enables a robot to automatically figure out what actions are useful given a user query by learning from the datasets. Pictures of a robot executing some of the exploration actions are shown in Figure 2.

**Reporting actions:** $A^r$ includes a set of actions that are used for reporting the object's properties and can deterministically lead the state transition to *term* (terminal state). For instance, if a user queries about "a blue, heavy can", there will be three binary variables specifying each of properties is true or false. As a result, there will be eight reporting actions. For $a \in A^r$, we use $s \odot a$ (or $y \odot a$) to represent that the report of $a$ matches the underlying values of object properties (i.e., a correct report) and use $s \oslash a$ (or $y \oslash a$) otherwise.

$T_{\mathcal{X}} : \mathcal{X} \times A \times \mathcal{X} \rightarrow [0,1]$ is the state transition function in the fully observable component of the current state. $T_{\mathcal{X}}$ includes a set of conditional probabilities of transitions from $x \in \mathcal{X}$—the fully observable component of the current state—to $x' \in \mathcal{X}$, the component of the next state, given $a \in A$ the current action. Reporting actions and illegal exploration actions (e.g., *dropping* an object in state $x_1$—before a successful grasp) lead state transitions to *term* with 1.0 probability.

Most exploration actions are unreliable and succeed probabilistically. For instance, $p(x_4,\ drop,\ x_5) = 0.95$ in our case, indicating there is small probability the object is stuck in the robot's hand. The success rate of action *look* is 1.0 in our case, since without changing positions of either the camera or the object it does not make sense to keep running the same vision algorithms and hence it is not allowed.

$T_{\mathcal{Y}} : \mathcal{Y} \times A \times \mathcal{Y} \rightarrow [0,1]$ is the state transition function in the partially observable component of the current state. It is an identity matrix in our case, (we assume) because object properties do not change during the process of the robot's exploration actions.

## 3.5 Reward Function and Discount Factor

$R : S \times A \rightarrow \mathbb{R}$ is the reward function. Each *exploration action*, $a^e \in A^e$, has a cost that is determined by the time required to complete the action. These costs are empirically assigned according to the datasets used in this research. The costs of *reporting actions* depend on whether the report is correct.

$$R(s,a) = \begin{cases} r^-, & \textbf{if } s \in S,\ a \in A^r,\ s \oslash a \\ r^+, & \textbf{if } s \in S,\ a \in A^r,\ s \odot a \end{cases}$$

where $r^-$ (or $r^+$) is negative (or positive) given an incorrect (or correct) report. Unless otherwise specified, $r^- = -500$ and $r^+ = 500$ in this paper.

Costs of other exploration actions are within the range of $[0.5, 22.0]$ (corresponding reward is negative), except that action *ask* has the cost of 100.0. $\gamma$ is a discount factor, and $\gamma = 0.99$ in our case. This setting gives the robot a relatively long planning horizon.

## 3.6 Observations and Observation Function

$Z : Z^h \cup \emptyset$ is a set of observations. Elements in $Z^h$ include all possible combinations of object properties and have one-one correspondence to elements in $A^r$ and $\mathcal{Y}$. For instance, when the query is about "a *red empty bottle*", there exists an observation $z \in Z^h$ that represents "the object's color is red; it is not empty, and it is a bottle." Actions that produce no information gain (*reinitialize*, in our case), and reporting actions in $A^r$ result in a $\emptyset$ (none) observation.

$O : S \times A \times Z \rightarrow [0,1]$ is the observation function that specifies the probability of observing $z \in Z$ when action $a$ is executed in state $s$: $O(s,a,z)$. In this work, the probabilities are learned from performing cross-validation on the robot's training data. As described in Section 3.1, predicate learning produces confusion matrix $C_p^b \in \mathbb{R}^{2\times 2}$ for each predicate $p$ and each behavior $b$, where $b$ corresponds to one of the exploration actions shown in Figure 1.

$$\begin{aligned} O(s,a,z) &= Pr(\mathbf{p}^s, b, \mathbf{p}^z) \\ &= C^b_{p_0}(p^s_0, p^z_0) \cdot C^b_{p_1}(p^s_1, p^z_1) \cdots C^b_{p_{N-1}}(p^s_{N-1}, p^z_{N-1}) \end{aligned}$$

where behavior $b$ corresponds to action $a$; $\mathbf{p}^s$ and $\mathbf{p}^z$ are the vectors of *true* and *observed* values (0 or 1) of the predicates; $p^s_i$ (or $p^z_i$) is the true (or observed) value of the $i^{th}$ predicate; and $N$ is the total number of predicates in the query.

## 3.7 Dynamically Constructed Controllers

State set $\mathcal{Y}$ can be very large, due to the large number of predicates and the exponentially increasing number of their combinations. For example, one of the datasets in our experiments contains 81 predicates, resulting in $2^{81}$ possible states. Due to limited computational resources, it would be intractable for a robot to generate a far-sighted policy for identifying an object according to all 81 predicates.

Recent research decomposes a sequential decision-making problem into two tractable subproblems that respectively focus on high-dimensional reasoning (e.g., objects with many properties) and long-horizon planning (e.g., tasks that require many actions) (Zhang, Khandelwal, and Stone 2017). Based on that approach, we dynamically construct controllers that include a minimum set of predicates, instead of modeling all of them, in the $\mathcal{Y}$ component. In addition to $\mathcal{Y}$, the following components depend on the user query: reporting actions $A^r$, object property combinations $Z^h$, and the reward and observation functions (due to the involvement of $\mathcal{Y}$). As a result, our query-oriented, MOMDP-based controllers are relatively very small, and typically include fewer than 100 states at runtime.

It should be noted that we use MOMDP, as a special form of POMDP, to model our domain mainly for the ease of describing the mixed observability over $\mathcal{X}$ and $\mathcal{Y}$ (Section 3.3). Our approach enables automatic generation of complete MOMDP models. One can encode such MOMDP models in such a way that existing POMDP solvers (e.g., (Kurniawati, Hsu, and Lee 2009)) can be used to generate policies, as we do in this work.

# 4 Experimental Results

We evaluate the proposed method using two datasets in which a robot explored a set of objects using a variety of exploratory behaviors and sensory modalities, and show that for both our proposed MOMDP model outperforms baseline models in exploration accuracy and overall exploration cost. Two datasets of **Sinapov14** and **Thomason16** have been used in the experiments, where **Thomason16** has a much more diverse set of household objects and a larger number of predicates that arose naturally during human-robot interaction gameplay.

**Sinapov14 Dataset:** In this dataset, the robot explored 36 different objects using 11 prototypical exploratory behaviors: *look*, *grasp*, *lift*, *shake*, *shake-fast*, *lower*, *drop*, *push*, *poke*, *tap*, and *press* 10 different times per object. The objects are lidded containers with the same shape and varied

Figure 3: Objects in the **Thomason16** dataset (Left) and the one used in the illustrative example in Section 4.1 (Right).

along 3 different attributes: 1) color: *red*, *green*, *blue*; 2) weight: *light*, *medium*, *heavy*; and 3) contents: *beans*, *rice*, *glass*, *screws*. These variations result in the $3 \times 3 \times 4 = 36$ objects bearing combinations of these attributes in the set $P$ that the robot is tasked with learning. It should be noted that costs of actions in the two datasets are different, because the datasets were collected using different robots.

**Thomason16 Dataset:** In this dataset, the robot explored 32 common household objects using 8 exploratory actions: *look*, *grasp*, *lift*, *hold*, *lower*, *drop*, *push*, and *press*. Each behavior was performed 5 times on each object. The dataset was originally produced for the task of learning how sets of objects can be ordered and is described in greater detail by (Sinapov et al. 2016).

For the *look* behavior, *color*, *shape*, and *deep* features (the penultimate layer of the trained VGG network (Simonyan and Zisserman 2014)) are available. For the remaining behaviors, the robot recorded *audio*, *proprioceptive* (finger positions for *grasp*), and *haptic* (i.e., joint forces) features produced by the interaction with the object. These modalities result in $|C| = 7 \times 2 + 1 \times 3 = 17$ sensorimotor contexts.

The set of predicates $\mathcal{P}$ consisted of 81 words used by human participants to describe objects in this dataset during an interactive gameplay scenario described by (Thomason et al. 2016). Example predicates include the words *red*, *heavy*, *empty*, *full*, *cylindrical*, *round*, etc. Unlike the **Sinapov14** dataset, here the objects vary greatly, and the predicate recognition problem is much more difficult.

## 4.1 Illustrative Example

We now describe an example in which a robot is tasked with identifying properties of a given object. We randomly selected an object from the **Thomason16** dataset: a blue and red bottle full of water (Figure 3). We then randomly selected properties, in this case "yellow" and "metallic," and asked the robot to identify whether the object has each of the properties or not. The selected object was not part of the robot's training set used to learn the predicate recognition models and the MOMDP observation model. The robot should report negative to both properties while minimizing the overall cost of exploration actions.

Given this user query, we generate a MOMDP model that includes 25 states. We then generate an action policy using past work's methods (Kurniawati, Hsu, and Lee 2009).

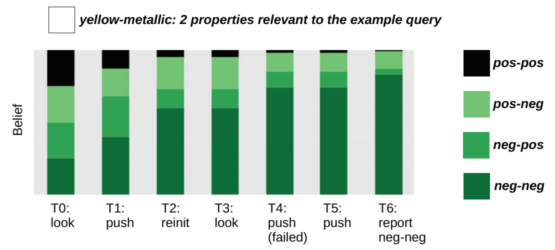


Figure 4: Action selection and belief change in the exploration of a red and blue bottle full of water, given a query of *yellow* and *metallic*.

Currently, building the model takes almost no time, and we uniformly gave two seconds for policy generation using the model (same in all experiments). The time for computing the policy is insignificant relative to the time for exploratory behaviors (which is what we are really trying to minimize).

Figure 4 shows the belief change in this process. The initial distributions over $\mathcal{X}$ and $\mathcal{Y}$ are $[1.0, 0.0, \cdots]$ and $[0.25, 0.25, 0.25, 0.25]$ respectively. The policy suggests "look" first. We queried the dataset to make an observation, *neg-neg* in this case. The belief over $\mathcal{Y}$ is updated based on this observation: $[0.41, 0.28, 0.19, 0.13]$, where the entries represent *neg-neg*, *neg-pos*, *pos-neg*, and *pos-pos* respectively. There is a (fully observable) state transition in $\mathcal{X}$, from $x_0$ to $x_1$, so the belief over $\mathcal{X}$ becomes $[0.0, 1.0, 0.0, \cdots]$. Based on the updated beliefs, the policy suggests taking the "push" action, which results in another *neg-neg* observation. Accordingly, the belief over $\mathcal{Y}$ is updated to $[0.60, 0.13, 0.22, 0.05]$, which indicates that the robot is more confident that the object is neither "yellow" nor "metallic". After actions of *reinitialize*, *look*, *push*, and *push* (this first *push* action was unsuccessful, and produced the $\emptyset$ observation), the belief over $\mathcal{Y}$ becomes $[0.84, 0.04, 0.12, 0.01]$. The policy finally suggests reporting *neg-neg*, making it a successful trial with an overall cost of 167 seconds, which results in a reward of $500 - 167 = 333$ (an incorrect report would have resulted in $-667$ reward).

**Remarks:** It should be noted that the classifiers associated with each behavior and word will produce an output even in cases where the sensory signals from that behavior are irrelevant to the word. For instance, although the sensory signals relevant to "push" are haptics and audio, the first "push" action results in an observation of "yellow". It was "yellow:neg", because the training set prior of most objects are not yellow. The robot favors actions that distinguish 'easy' predicates (*look* distinguishes *yellow* well in this case) because there is the discount factor (0.99): If an action is useful, the robot will prefer taking it early. The more the action is delayed, the more the expected reward is discounted.

## 4.2 Results

Next, we describe the experiments we conducted to evaluate the proposed MOMDP-based multi-modal perception strategy for object exploration. The goal was to increase the

Table 1: Performances of MOMDP-based and two baseline planners in cost (second) and accuracy on the **Sinapov14** dataset. Numbers in parenthesis denote the Standard Deviations over 400 trials.

| Properties | Method | Overall cost (std) | Accuracy |
|---|---|---|---|
| Two | Random Plus | 17.56 (30) | 0.245 |
| | Predefined Plus | 37.10 (0.00) | 0.583 |
| | MOMDP (Ours) | 29.85 (12.87) | 0.860 |
| Three | Random Plus | 10.12 (21.77) | 0.130 |
| | Predefined Plus | 37.10 (0.00) | 0.373 |
| | MOMDP (Ours) | 33.87 (8.78) | 0.903 |

accuracy in identifying properties of a novel object while reducing the overall action costs required in this process. In all evaluation runs, the object that needs to be identified was not part of the robot's training set when learning the predicate recognition models or the MOMDP parameters. The following baseline action strategies are used in experiments, where belief is updated using Bayes' rule except for *Random*:

- *Random*: Actions are randomly selected from $A$ that includes both reporting and legal exploration actions. A trial is terminated any of the reporting actions.
- *Random Plus*: Actions are randomly selected from legal exploration actions. Under an exploration budget, one selects the reporting action that makes the best sense (i.e., that corresponding to $y$ with the highest belief).
- *Predefined*: An action sequence is strictly followed: *ask, look, press, grasp, lift, lower* and *drop*.[2] Under an exploration budget or in early terminations caused by illegal actions, the robot selects the reporting action that makes the best sense.
- *Predefined Plus*: The same as *Predefined* except that unsuccessful actions are repeated until achieving the desired result(s).

**Sinapov14 Dataset:** In each trial, we place an object that has three attributes (color, weight and content) on a table and then generate an object description that includes the values of two or three attributes. This description matches the object in only half of the trials. When two (or three) attributes are queried, $\mathcal{Y}$ includes four (or eight) states plus *term* state, resulting in $\mathcal{S}$ that includes 25 (or 49) states. The other components of the dynamically constructed MOMDPs grow accordingly, given an increasing number of queried attributes.

Experimental results are reported in Table 1. Not surprisingly, randomly selecting actions produces low accuracy. The overall cost is smaller in more challenging trials (all three properties are questioned), because in these trials there are relatively fewer exploration actions (more properties produce more reporting actions), making the agent more likely to take a reporting action. Our MOMDP-based multimodal perception strategy reduces the overall action cost while significantly improving the reporting accuracy. Our performance improvement is achieved by repeating actions as needed, selecting legal actions (e.g., *lift* is legal only if the current state is $x_2$) that produce the most information or have the potential of doing so in the future, and even arbitrarily reporting without "wasting" exploration actions given queries where the exploration actions are not effective.

[2]Action *ask* was used only in the **Thomason16** experiments, because other exploration actions are not as effective as in **Sinapov14**.

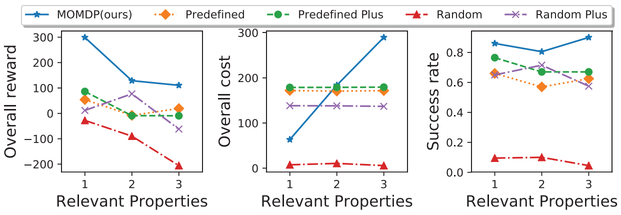


Figure 5: Evaluations of five actions strategies on the **Thomason16** dataset. Comparisons are made in three categories of *overall reward* (Left), *exploration cost* (Middle), and *success rate* (Right).

**Thomason16 Dataset:** In this set of experiments, a user query is specified by randomly selecting one object and $N$ properties ($1 \leq N \leq 3$), on which the robot is questioned. Each data point is an average over 200 trials, where we conducted pairwise comparisons over the five strategies, i.e., the strategies were evaluated using the same set of user queries. A trial is successful only if the robot reports correctly on all properties. It should be noted that most of the contexts are misleading in this dataset due to the large number of object properties, so it happens that more exploration actions confuse the robot more if the actions are not carefully selected. Figure 5 shows the experimental results. Overall reward is computed by subtracting overall action cost from the reward yielded by the reporting action (either a big bonus or a big penalty). We do not compute standard deviations in this dataset, because the diversity of the tasks results in problems of very different difficulties.

We can see our MOMDP-based strategy consistently performs the best in terms of the overall reward and overall accuracy. When more properties are queried, the MOMDP-based controllers enable the robot to take more exploration actions (Middle subfigure), whereas the baselines could not adjust their question-asking strategy accordingly.

The last experiment aims to experimentally evaluate the need of dynamically constructed controllers. We constructed MOMDP controllers including two relevant and an increasing number of irrelevant properties (i.e., the ones that are not queried). Results are shown in Figure 6. We can see, the quality of the generated action policies decreases soon (from higher than 150 to lower than 25 in reward), when more irrelevant properties are included in the MOMDPs. We did not include six or more irrelevant properties, because the solver cannot produce any policy in one and a half minutes.

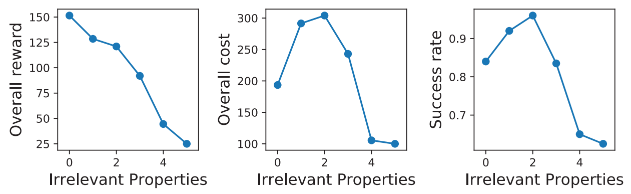


Figure 6: A "super" MOMDP that models two relevant and (an increasing number of) irrelevant properties, in comparison to dynamically constructed controllers used in this work.

## 5 Conclusions and Future Work

We investigate using mixed observability Markov decision processes (MOMDPs) to help robots select actions for multimodal perception in object exploration tasks. Our approach can dynamically construct a MOMDP model given an object description from a human user (e.g., "*a blue heavy bottle*"), compute a high-quality policy for this model, and use the policy to guide robot behaviors (such as "look" and "shake") toward maximizing information gain. The dynamically built controllers enable the robot to focus on a minimum set of domain variables that are relevant to the current object and query. The MOMDP models are constructed using two existing datasets collected with robots interacting with objects in the real world. Experimental results show that our object exploration approach enables the robot to identify object properties more accurately without introducing extra cost from exploration actions compared to a baseline that suggests actions following a predefined action sequence.

This research primarily focuses on a robot exploring objects in a tabletop scenario. In future work, we plan to investigate applying this approach to tasks that involve more human-robot interaction and mobile robot platforms, where exploration would require navigation actions and perceptual modalities such as human-robot dialog. Finally, in the two datasets used in this paper, the robot's manipulation actions were always successful but that would not always be the case in a real-world scenario; therefore we plan to extend our framework to situations in which the robot's actions may fail (in terms of manipulation) or cause undesirable outcomes (e.g., dropping an object may break it).

## Acknowledgments

A portion of this work has taken place in the Learning Agents Research Group (LARG) at UT Austin. LARG research is supported in part by NSF (IIS-1637736, IIS-1651089, IIS-1724157), Intel, Raytheon, and Lockheed Martin. Peter Stone serves on the Board of Directors of Cogitai, Inc. The terms of this arrangement have been reviewed and approved by the University of Texas at Austin in accordance with its policy on objectivity in research.

## References

Alomari, M.; Duckworth, P.; Hogg, D. C.; and Cohn, A. G. 2017. Natural language acquisition and grounding for embodied robotic systems. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, 4349–4356.

Chu, V.; McMahon, I.; Riano, L.; McDonald, C. G.; He, Q.; Perez-Tejada, J. M.; Arrigo, M.; Darrell, T.; and Kuchenbecker, K. J. 2015. Robotic learning of haptic adjectives through physical interaction. *Robotics and Autonomous Systems* 63:279–292.

Denil, M.; Agrawal, P.; Kulkarni, T. D.; Erez, T.; Battaglia, P.; and de Freitas, N. 2017. Learning to perform physics experiments via deep reinforcement learning. In *International Conference on Learning Representations*.

Fishel, J., and Loeb, G. 2012. Bayesian exploration for intelligent identification of textures. *Frontiers in Neurorobotics* 6:4.

Gao, Y.; Hendricks, L. A.; Kuchenbecker, K. J.; and Darrell, T. 2016. Deep learning for tactile understanding from visual and haptic data. In *International Conference on Robotics and Automation*, 536–543. IEEE.

Högman, V.; Björkman, M.; and Kragic, D. 2013. Interactive object classification using sensorimotor contingencies. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2799–2805. IEEE.

Kaelbling, L. P.; Littman, M. L.; and Cassandra, A. R. 1998. Planning and acting in partially observable stochastic domains. *Artificial Intelligence* 101(1):99–134.

Krishnamurthy, J., and Kollar, T. 2013. Jointly learning to parse and perceive: Connecting natural language to the physical world. *Transactions of the Association for Computational Linguistics* 1:193–206.

Kurniawati, H.; Hsu, D.; and Lee, W. S. 2009. SARSOP: efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *Robotics: Science and Systems Conference*, 65–72. The MIT Press.

Matuszek, C.; FitzGerald, N.; Zettlemoyer, L.; Bo, L.; and Fox, D. 2012. A joint model of language and perception for grounded attribute learning. In *Proceedings of the 29th International Conference on Machine Learning*.

Ong, S. C.; Png, S. W.; Hsu, D.; and Lee, W. S. 2010. Planning under uncertainty for robotic tasks with mixed observability. *The International Journal of Robotics Research* 29(8):1053–1068.

Pajarinen, J., and Kyrki, V. 2015. Robotic manipulation of multiple objects as a POMDP. *Artificial Intelligence*.

Puterman, M. L. 1994. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

Rebguns, A.; Ford, D.; and Fasel, I. R. 2011. Infomax control for acoustic exploration of objects by a mobile robot. In *Lifelong Learning*.

Simonyan, K., and Zisserman, A. 2014. Very deep convolutional networks for large-scale image recognition. *CoRR* abs/1409.1556.

Sinapov, J., and Stoytchev, A. 2009. From acoustic object recognition to object categorization by a humanoid robot. In *Proc. of the RSS 2009 Workshop-Mobile Manipulation in Human Environments*.

Sinapov, J.; Schenck, C.; Staley, K.; Sukhoy, V.; and Stoytchev, A. 2014. Grounding semantic categories in behavioral interactions: Experiments with 100 objects. *Robotics and Autonomous Systems* 62(5):632–645.

Sinapov, J.; Khante, P.; Svetlik, M.; and Stone, P. 2016. Learning to order objects using haptic and proprioceptive exploratory behaviors. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence*.

Sinapov, J.; Schenck, C.; and Stoytchev, A. 2014. Learning relational object categories using behavioral exploration and multimodal perception. In *IEEE International Conference on Robotics and Automation*, 5691–5698.

Sridharan, M.; Wyatt, J.; and Dearden, R. 2010. Planning to see: A hierarchical approach to planning visual actions on a robot using POMDPs. *Artificial Intelligence* 174(11):704–725.

Thomason, J.; Sinapov, J.; Svetlik, M.; Stone, P.; and Mooney, R. J. 2016. Learning multi-modal grounded linguistic semantics by playing I Spy. In *Proceedings of the Twenty-Fifth international joint conference on Artificial Intelligence*.

Whitney, D.; Eldon, M.; Oberlin, J.; and Tellex, S. 2016. Interpreting Multimodal Referring Expressions in Real Time. In *International Conference on Robotics and Automation*.

Zhang, S.; Khandelwal, P.; and Stone, P. 2017. Dynamically constructed (PO)MDPs for adaptive robot planning. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, 3855–3863.

# Human-Agent Teaming as a Common Problem for Goal Reasoning

**Matthew Molineaux,**[1] **Michael W. Floyd,**[1] **Dustin Dannenhauer,**[2] **David W. Aha**[3]
[1]Knexus Research Corporation; Springfield, VA | {first.last}@knexusresearch.com
[2]NRC Postdoctoral Fellow; NRL; Navy Center for Applied Research in AI; Washington, DC | dustin.dannenhauer.ctr@nrl.navy.mil
[3]Naval Research Laboratory; Navy Center for Applied Research in AI; Washington, DC | david.aha@nrl.navy.mil

**Abstract**

Human-agent teaming is a difficult yet relevant problem domain to which many goal reasoning systems are well suited, due to their ability to accept outside direction and (relatively) human-understandable internal state. We propose a formal model, and multiple variations on a multi-agent problem, to clarify and unify research in goal reasoning. We describe examples of these concepts, and propose standard evaluation methods for goal reasoning agents that act as a member of a team or on behalf of a supervisor.

## 1 Introduction

An important focus of research on intelligent agents is to achieve goals quickly and reliably. In recent years, goal reasoning researchers have considered the issue of *goal change*, a process by which an agent can shift the overall focus of its activities. This change can be prompted by a nameless outside goal source and/or an internal motivation model. In this work, we advocate modeling the other agents whose goals an agent attempts to achieve. With this model change, it becomes clear that goal reasoning agents are particularly well-suited to being team players. We define a human-agent teaming model and problem, and discuss how future goal reasoning research can leverage it.

Research on goal reasoning has investigated multiple framework abstractions for algorithms and agent architectures (e.g., Goal-Driven Autonomy (Molineaux, Klenk, and Aha 2010) and the Goal Lifecycle (Roberts et al. 2014)), but has not focused on common problems. Areas such as reinforcement learning and automated planning have benefited greatly from such a focus, receiving additional attention from competitions and comparing results via easy-to-use benchmarks. While one problem may not suffice to compare all goal reasoning agents, a small number of common problems could facilitate comparative publications, and thereby focus goal reasoning research. This paper focuses on elaborating this position, and a candidate formal framework for describing classes of problems; we expect that future work will specify concrete representations and initial problems.

In Section 2, we provide a formal description of a general *human-agent teaming* problem, along with several important variations that are commonly encountered in goal reasoning research. We then discuss some examples of the concepts described in Section 3, and discuss useful metrics for comparison in Section 4. Finally, in Section 5 we conclude.

## 2 Models of Goal Reasoning for Human-Agent Teaming

In recent work, goal reasoning systems have explicitly reasoned over the presence of other agents and their goals. For example, goal reasoning agents may be aware that their opponent in a real-time strategy game is attempting to defeat them (Weber, Mateas, and Jhala 2010; Jaidee, Muñoz-Avila, and Aha 2013; Dannenhauer and Muñoz-Avila 2015), that other agents may attack them (Bonnano et al. 2016), or that other agents may impede them (Cox 2013). Other work has described explicit exchange of goals and other information between agents and humans for the purpose of general collaborative tasks (Geib et al. 2016), control of unmanned vehicles (Richards and Stedmon 2017), and autonomous community formation (Golpayegani and Clarke 2016). The framework presented here is designed to facilitate communication and comparison of agents that work together in these ways. Concepts described here help with the modeling of the goals, plans, and motivations of other agents, especially those that reason over goals themselves. In the spirit of the successful reinforcement learning problem (Sutton and Barto 1998), we describe a simple set of functions and informational items intended to be general enough to be easily applied and used by all agents that solve these problems. In order to keep this framework generic and approachable, we avoid committing to representations and functions that many agents may not be able to provide.

In our model (Figure 1), a team is situated in an environment. This team can comprise goal reasoning agents, human teammates, and other software agents. At each time $t$ ($t \in T$, the set of discrete time points at which communications occur), each *teammate* observes the environment. The environment's state is given by $s_t$ ($s_t \in S$, the set of all environment states), and teammate $m$ ($m \in M$, the set of teammates) receives an observation $o_t^m$ ($o_t^m \in O$, the set of all observations). The environment creates individualized observations for each agent; we model the observation generation process as a function $obs^m : S \rightarrow O$. Teammates can perform

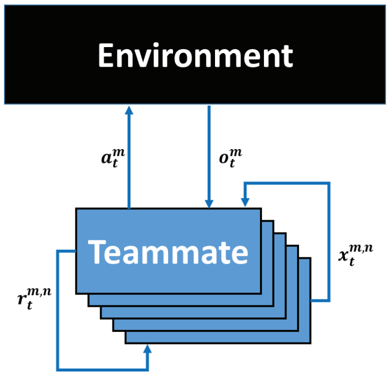


Figure 1: Human-Agent Teaming Problem Model

an action at each time $t$, denoted $a_t^m$ ($a_t^m \in A$, the set of all actions). Changes in the environment are dependent on these actions as well as the prior state, which we model as the transition function $\lambda : S \times A^{|M|} \rightarrow S$. This generic representation allows for description of a wide variety of environments, including those with heterogeneous observability, exogenous events, and role-based actions; however, it does not permit continuous time.

Acting as teammates imposes some extra requirements on an agent. Work in human factors (Klein et al. 2004) has recognized four distinct requirements for acting as a member of team. Loosely summarized, they are: (1) agree on common goals; (2) direct and take direction from other teammates; (3) predict the behavior of other teammates and act in a way they can predict; and (4) maintain a common understanding of the shared environment. To support these requirements, in our framework teammates communicate via requests and explanations. A teammate $m$ can make a *request* $r_t^{m,n}$ of another teammate $n \in M$ ($r_t^{m,n} \in R$, the set of all requests). Requests should describe everything agent $m$ desires of agent $n$ at time $t$. They are used both for direction and describing desired changes to common goals. Our model makes no specific commitment to representation; however, we expect that goal reasoning agents might directly exchange lists of goals, preferences, and constraints.

Explanations are intended to communicate information about an agent's internal state that motivates that agent's current behavior (e.g., "I moved the box because it was blocking my vision", "My battery is low so my movement range is limited"). Each teammate $m$ provides an *explanation* $x_t^{m,n}$ to each other teammate $n$ ($x_t^{m,n} \in X$, the set of all explanations). These explanations should help other teammates to understand an agent's actions and predict their future actions, to facilitate coordination. One particular area of importance is that an agent should explain why it does or does not pursue another agent's request; if an agent does not, for example, have sufficient resources to succeed, this may prompt the requester to provide resources or assistance.

Note that the explanations described here are proactive and not query-based. While query-based explanations are an important problem, a clean separation of agent-based coordination and decision-making issues from natural-language issues will permit objective evaluations and comparisons without human interaction issues. We expect, however, that an external query interface could be provided that translates queries into informational requests.

Each teammate $m$ uses the various pieces of information they have received over time[1] (i.e., observed environment states, received requests, and received explanations) along with their sent requests (and, implicitly, their internal motivations) to guide their action selection policy $\pi^m : O^{|T|} \times R^{|M|} \times X^{|M| \times |T|} \times R^{|M|} \rightarrow A$. This policy is expected to be dynamic, and may be influenced by an agent's interactions with its teammates, as well as by the environment. A typical goal reasoning agent's policy may involve considering and reselecting goals and replanning to achieve them, but the model accommodates various types of policies.

We also model the *satisfaction* of each teammate, which describes how well an agent's desires are being met. Satisfaction is a function of an agent's observations (which may indicate the achievement of desired states), requests made and received (which help determine the success and failure of collaboration), and explanations received (which may justify failures or provide confidence in the current collaboration): $sat^m : O^{|T|} \times R^{|M|} \times X^{|M| \times |T|} \times R^{|M|} \rightarrow \mathbb{R}$. The satisfaction of the entire team can also be modelled as a function of each teammate's satisfaction ($f(sat^1, \ldots, sat^{|M|})$); optimizing this measure incorporates an agent's own satisfaction, as well as the estimated satisfaction of each of its $|M| - 1$ teammates.

To exemplify how our model could be used in practice, we describe it in terms of four variations on the human-agent teaming problem that describe existing goal reasoning work: *single supervisor*, *silent teammates*, *silent assistant*, and *rebel agent*. These examples are not meant to be exhaustive, but instead to show that our model can represent common team structures encountered in goal reasoning research.

## 2.1 Single Supervisor

Even autonomous goal reasoning agents often receive goals or tasks from an outside source. In this framework, we model that source as an agent who makes requests and wants explanations to understand what the agent is doing to fulfill them. This results in the Single Supervisor version of the human-agent teaming problem model, shown in Figure 2. In this version, an agent has a single teammate whose satisfaction it wishes to maximize, referred to as the *supervisor*. While both teammates can sense and act in the environment[2], the superior-subordinate relationship results in requests and explanations being unidirectional (i.e., the agent cannot make requests of the supervisor and the supervisor does not explain itself to the agent). As such, the agent's action selection policy does not include explanations it has received or

[1]We assume that, since the requests at the current time contain the complete request to/from each agent, the policy does not need to consider past requests. If this is not the case, the action selection policy can be extended to include past requests.

[2]Although the supervisor does not need to be situated in the environment.

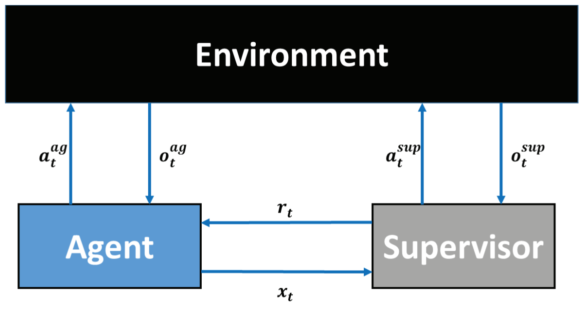


Figure 2: Single Supervisor version of the Human-Agent Teaming Problem Model

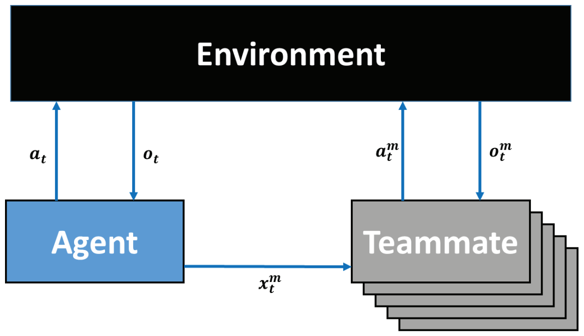


Figure 3: Silent Teammates version of the Human-Agent Teaming Problem Model

requests it has sent, and only deals with a single teammate (the policy is simplified to $\pi : O^{|T|} \times R \rightarrow A$). The primary performance measure for this problem is the supervisor's true satisfaction, measured either at the termination of interaction, or as an average over time.

## 2.2 Silent Teammates

In the Silent Teammates version of the human-agent teaming problem model (Figure 3), an agent operates as a member of a human-agent team, but does not receive any direct requests from its teammates. This is an unusual teaming arrangement, but necessary when a team is communication-restricted in some way (possibly to avoid giving an adversary knowledge). In this problem, the agent does not make requests of other teammates, nor expect explanations from them. However, the agent still provides an explanation on demand, to assist teammates in understanding when they have questions. An example of such a goal reasoning agent is the Autonomous Squad Member (ASM), an agent controlling an unmanned ground vehicle that is embedded in a team of humans (Gillespie et al. 2015). The ASM agent must infer and respond to teammates' desires (e.g., follow along, provide cover in a fight) without explicit requests. This results in an action selection policy that inputs only observations: $\pi : O^{|T|} \rightarrow A$. Similarly, the satisfaction function does not include requests: $sat^m : O^{|T|} \times X^{|M| \times |T|} \rightarrow \mathbb{R}$. The primary performance measure in this problem is the team's overall satisfaction.

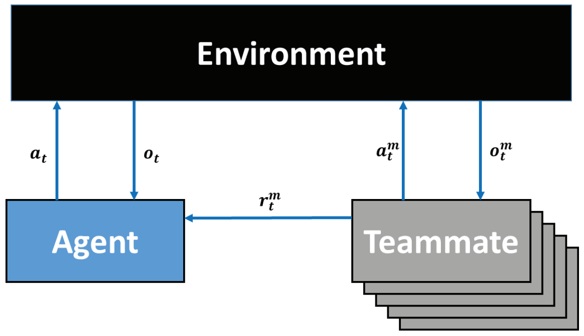


Figure 4: Silent Assistant version of the Human-Agent Teaming Problem Model

## 2.3 Silent Assistant

The Silent Assistant version is a multi-agent teaming problem with no explanation requirement (Figure 4). In this example, the agent assists one or more other agents by acting on their requests, but does not provide explanations, receive explanations, or make requests of others (i.e., it does not initiate coordination). An example of such a goal reasoning agent is the Tactical Battle Manager (TBM), an agent that controls an unmanned air vehicle while serving as a wingman for an aircraft controlled by a human pilot (Floyd et al. 2017). The TBM operates autonomously but receives explicit tasks from a human pilot. The lack of communication from the agent is largely due to the real-time adversarial nature of the domain; goal changes are motivated by dangerous situations or opportunistic targets, so explanations are not a primary requirement for this system. Additionally, since the TBM is a human pilot's wingman, it serves a subordinate role and therefore does not generate requests. As such, the agent's action selection function and the satisfaction functions do not include explanations or requests from the agent ($\pi : O^{|T|} \times R^{|M|} \rightarrow A$, $sat^m : O^{|T|} \times R^{|M|} \rightarrow \mathbb{R}$). The primary performance measure is the team satisfaction function.

## 2.4 Rebel Agent

The previous three problem versions we described assume that the agent's primary drive is to satisfy teammates' requests. In the Rebel Agent version (Coman, Gillespie, and Muñoz-Avila 2015), an agent has internal goals or motivations that differ from (and may conflict with) those of its teammates. There are two ways in which a rebel agent can be represented using our model. The simplest method is to consider the agent as a member of its team but having internal motivations that are unknown to its teammates. Thus, when attempting to maximize team satisfaction it may prioritize its own satisfaction above the satisfaction of its teammates (e.g., provide them with different weights). The ARTUE agent (Molineaux, Klenk, and Aha 2010) is a rebel agent that receives explicit requests in the form of goals that it may choose to ignore in order to achieve goals more important to it. A more complex representation would be to consider the agent to be a member of two teams concurrently (i.e., in Figure 1 the agent would be at the intersection of two teams). For example, consider an agent that is a member of a *corporate catering team*, but is also a member of a *vegetarian team*. While the agent contributes toward achieving catering goals (e.g., host a successful event, maximize profit) it may choose actions to maximize the vegetarian team's satisfaction (e.g., minimize the amount of meat used). In the internally motivated case, the primary performance measure is a team/rebel satisfaction function $f(sat^1, \ldots, sat^{|M|}, mot(s_{final}))$, where $mot(s_{final})$ describes how well a rebel's internal motivations are satisfied in the true final state of the environment. In the dual-membership case, the primary performance measure is a combined function of two (or potentially more) team satisfaction functions: $f_C(f_1(sat^1, \ldots, sat^{|M|}), f_2(sat^1, \ldots, sat^{|M|}))$.

## 2.5 Assumptions

Consideration of important assumptions is necessary for this framework. Existence of the transition and observation functions means that environments can be static or dynamic, deterministic or probabilistic, and fully or partially observable. Existence of the policy and satisfaction functions of teammates implies that we should also consider whether to assume complete or incomplete knowledge about these functions, and whether information given regarding them (i.e., requests and explanations) is perfect or noisy. This cuts across all problems, and those purporting to address these problems should state their assumptions regarding these functions.

## 3 Examples

Requests and explanations can take many forms including natural language utterances, structured text, or low-level state representations. In this section we provide examples of requests, explanations, and how they can be used.

**Requests:** In general, we expect requests to vary in complexity across agents. An example complex request representation might be a tuple $\langle S_{avoid}, F_{prefs}, G, C\rangle$, including constraints $S_{avoid} \subset S$ in the form of states to avoid (e.g., "battery should never fall below 10%"), preference functions $F_{prefs} : S \times S \rightarrow \{True, False\}$ (e.g., "spend as little money as possible"), goal states $G \subset S$ (with or without priorities), and context $C$ that describes why achievement of a particular goal is desired (e.g., the reason for requesting an agent to cook food could be because (1) 'supervisor is hungry' or (2) 'supervisor needs to bring food to a dinner party later'). Context and preferences are especially relevant for goal reasoning agents, as these can guide which goals should be considered when goal change is warranted. Additionally, the reasons for a supervisor's request of a goal are likely to be useful in making goal change decisions; for example, the context may include a higher-level goal of which the current request is a subgoal (e.g., a "cook food" goal is a subgoal of a $\neg$hungry goal).

**Explanations:** An important reason for explanations is that goal reasoning agents may change their local objectives (i.e., subgoals) in response to changes in the environment preventing the accomplishment of the original task. Thus, whenever an agent changes its goal, an explanation could be a tuple $\langle g_{failed}, c_{failed}, g_{new}, p_{new}\rangle$ composed of a failed goal $g_{failed}$, description of state properties that prevent goal achievement $c_{failed}$, new goal $g_{new}$, and new plan $p_{new}$. Note that in this framework, explanations are always proactive for simplicity of discussion; to support reactive explanations, an external interface could store this information to present to a human in answer to specific queries.

**A Supervisor Requests Cake:** We now describe an example of the Single Supervisor problem: first, a human supervisor $\sigma$ makes a request of a chef agent $\alpha$ to "*bake me a chocolate cake that I can eat when I get home*". Here, the request $r_t^{\sigma,\alpha}$ is the tuple $\langle \emptyset, \emptyset, \{\{$*exists(chocolate-cake), on(chocolate-cake,table)*$\}\}, \{$*hungry(me), wants(me, chocolate)*$\}\rangle$, which describes a single goal state based on the original English utterance (translating human utterances to goals has garnered attention in the human-robot interaction community, see (Briggs, McConnell, and Scheutz 2015) for an example). No constraints or preferences are provided.

The chef agent $\alpha$ represents its supervisor's satisfaction function $sat^\sigma$ as a weighted average of (1) the percentage of his desires that are satisfied in the current state and (2) the time delay between $t$ (time of request issuance) and $t_a$ (time of request achievement). Based on this, the agent uses an automated planner to produce a plan that achieves the requested goal in the shortest possible time. Its policy $\pi^\alpha$ removes the first action from this plan and executes it; this is repeated until the following action $a_t^\alpha$ is known to be inadmissible based on a state observation $o_t^\alpha$. We now describe a situation that may warrant the agent to consider goal change.

Soon after it begins acting to achieve the goal, the agent discovers it cannot continue baking because there is no cake flour in the kitchen. The agent considers adoption of a new goal *acquire(cake-flour)*, and creates a plan: go to the grocery store, purchase cake flour, and return. However, the plan to accomplish the new goal would significantly increase the time required to fulfill the supervisor's request. Knowing that the supervisor is hungry and wants chocolate cake, the chef agent decides to instead switch to a goal to make chocolate chip pancakes, which seems like a reasonable substitute. When the supervisor comes home, the agent provides him with an explanation:

$\langle\{$*exists(chocolate-cake), on(chocolate-cake, table)*$\}$,
$\{$*available(cake-flour)*$\}$,
$\{$*exists(pancakes), on(pancakes, table)*$\}$,
$\{$*acquire(pancake-mix), acquire(chocolate-chips), bake(pancakes, pancake-mix, chocolate-chips), serve(pancakes)*$\}\rangle$.

This explanation serves to communicate why the agent changed its goal, and what it did instead. If the context of the supervisor's request had been a birthday party, the agent $\alpha$ might have reasoned that the subgoal of going to the grocery store was warranted.

In general, the issue of how much information must be exchanged between teammates is unresolved. In this example, we assume sufficient knowledge to minimize the need for communication; for example, the agent knows that the supervisor's desires would be met to some degree by chocolate chip pancakes. Future work on goal reasoning agents will need to consider this question.

## 4 Evaluating Explainable Goal Reasoning Agents

We expect that typical evaluations will consider a specific problem and assumptions, and show results on a primary performance metric in a subset of domains. Results should be directly comparable with other agents that make the same assumptions, use the same domain, and use a similar set of teammates. For this reason, sharing domains as well as appropriate automated teammates (i.e., other software agents that are part of the team) should promote comparison.

When discussing the four versions of the human-agent teaming problem, we briefly described the various metrics that can be used to measure whether the goal reasoning agent is an effective member of the team. However, in addition to agent performance there is also the issue of how well the agent interacts with its human teammates. In these cases, evaluations should consider whether the provided explanations are appropriate for aiding human collaboration. We consider metrics for explanation as falling into four categories: *tests of explanation quality*, *tests of user satisfaction*, *tests of user comprehension*, and *tests of user or user-system team performance*. These are based directly on Hoffman, Klein and Mueller's (2017) work on evaluating explanations. Two agents need not use the same explanation representation (e.g., natural language, internal state variables) to be compared.

**Tests of Explanation Quality:** Experiments that measure explanation quality can be conducted without humans in the loop, but often still require a human to assess the results. These can be compared against explanations generated by another system or by a human. Some measures of explanation quality are surveyed in Table 1.

**Tests of User Satisfaction:** These should solicit a user's subjective satisfaction with an agent's performance, typically using Likert scale questions.

**Tests of User Comprehension:** These gauge how well explanations generated by an agent improve the accuracy of a user's mental model of an agent's behavior. For explainable autonomous agents, experiments could include questions about the system's policy to measure user understanding.

**Tests of User or User-System Team Performance:** These measure how explanation affects the user's ability to accomplish some task, often an interactive task involving the explaining agent. A scenario-specific performance metric can be used to evaluate the team's performance for this purpose. To provide a comparison, the same evaluation should be applied with and without agent-provided explanations, and, if possible, against a human-only team.

Table 1: Abstract Measures of Explanation Quality

| | |
|---|---|
| **Soundness** | Plausibility, internal consistency |
| **Appropriate Detail** | Amount of detail and its focus points |
| **Veridicality** | Does not contradict the ideal model (although there are times when inaccurate explanations work better for some users and some purposes) |
| **Usefulness** | Fidelity to the designer's or user's goal for system use |
| **Clarity** | Understandability |
| **Completeness** | Relative to an ideal model |
| **Observability** | Explains an agent mechanism |
| **Dimensions of Variation** | Reveals boundary conditions |

## 5 Conclusions and Future Work

We have presented new formal models and problem variations for human-agent teaming, in hopes of promoting comparisons, competitions, and sharing of evaluation code among goal reasoning researchers. We have made the case that explanation is an important and attainable capability for goal reasoning agents. Finally, we have described useful evaluations to be used to provide evidence of how well both goal reasoning agents and human-agent teams, perform.

In future work, we will produce refined models based on community feedback; furthermore, we will provide concrete problem instances and representations for use in benchmarking and comparison.

## 6 Acknowledgements

This research was developed with funding from DARPA. The views, opinions, and/or findings expressed are those of the author and should not be interpreted as representing the official views or policies of the DoD or the U.S. Government. Table 1 is used with permission from the Institute for Human and Machine Cognition and was created by IHMC, MacroCognition LLC, and Michigan Technical University, with the support of the DARPA XAI Program DARPA-BAA-16-53. All rights are reserved by IHMC.

## References

Bonnano, D.; Roberts, M.; Smith, L.; and Aha, D. 2016. Selecting subgoals using deep learning in Minecraft: A preliminary report. In *Working Notes of the IJCAI-16 Workshop on Deep Learning and Artificial Intelligence*.

Briggs, G.; McConnell, I.; and Scheutz, M. 2015. When robots object: Evidence for the utility of verbal, but not necessarily spoken protest. In *International Conference on Social Robotics*, 83–92. Springer.

Coman, A.; Gillespie, K.; and Muñoz-Avila, H. 2015. Case-based local and global percept processing for rebel agents. In *Case-Based Agents: Papers from the ICCBR 2015 Workshops*, 23–32.

Cox, M. T. 2013. Goal-driven autonomy and question-based problem recognition. In *Proceedings of the Second Annual Conference on Advances in Cognitive Systems, Poster Collection*, 29–45.

Dannenhauer, D., and Muñoz-Avila, H. 2015. Goal-driven autonomy with semantically-annotated hierarchical cases.

In *Proceedings of the 23rd International Conference on Case-Based Reasoning*, 88–103.

Floyd, M. W.; Karneeb, J.; Moore, P.; and Aha, D. W. 2017. A goal reasoning agent for controlling UAVs in beyond-visual-range air combat. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, 4714–4721. AAAI Press.

Geib, C.; Weerasinghe, J.; Matskevich, S.; Kantharaju, P.; Craenen, B.; and Petrick, R. P. 2016. Building helpful virtual agents using plan recognition and planning. In *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*.

Gillespie, K.; Molineaux, M.; Floyd, M. W.; Vattam, S. S.; and Aha, D. W. 2015. Goal reasoning for an autonomous squad member. In *Goal Reasoning: Papers from the ACS 2015 Workshops*, 52–67.

Golpayegani, F., and Clarke, S. 2016. Goal-based multi-agent collaboration community formation: A conceptual model. In *Workshop on Goal Reasoning at IJCAI-2016*.

Hoffman, R.; Klein, G.; and Mueller, S. 2017. Initial concepts and literature review for DARPA-XAI. Technical report from task area 2 (psychological model of explanation) on DARPA contract DARPA-BAA-16-53. Technical report, Institute for Human and Machine Cognition, Pensacola, FL.

Jaidee, U.; Muñoz-Avila, H.; and Aha, D. W. 2013. Case-based goal-driven coordination of multiple learning agents. In *Proceedings of the 21st International Conference on Case-Based Reasoning*, 164–178.

Klein, G.; Woods, D. D.; Bradshaw, J. M.; Hoffman, R. R.; and Feltovich, P. J. 2004. Ten challenges for making automation a ”team player” in joint human-agent activity. *IEEE Intelligent Systems* 19(6):91–95.

Molineaux, M.; Klenk, M.; and Aha, D. 2010. Goal-driven autonomy in a Navy strategy simulation. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, 1548–1554.

Richards, D., and Stedmon, A. 2017. Designing for human–agent collectives: display considerations. *Cognition, Technology & Work* 19(2-3):251–261.

Roberts, M.; Vattam, S.; Alford, R.; Auslander, B.; Karneeb, J.; Molineaux, M.; Apker, T.; Wilson, M.; McMahon, J.; and Aha, D. W. 2014. Iterative goal refinement for robotics. In *Planning and Robotics: Papers from the ICAPS Workshop*. AAAI Press.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement learning: An introduction*. MIT Press Cambridge.

Weber, B. G.; Mateas, M.; and Jhala, A. 2010. Applying goal-driven autonomy to starcraft. In *Proceedings of the Sixth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*.

# Planning Hierarchies and Their Connections to Language

**Nakul Gopalan**
ngopalan@cs.brown.edu
Brown University

**Abstract**

Robots working with humans in real environments need to plan in a large state–action space given a natural language command. Such a problem poses multiple challenges with respect to the size of the state–action space to plan over, the different modalities that natural language can provide to specify the goal condition, and the difficulty of learning a model of such an environment to plan over. In this thesis we would look at using hierarchical methods to learn and plan in these large state–action spaces. Further, we would look the using natural language to guide the construction and learning of hierarchies and reward functions.

## Introduction

In this work we consider the problem of robots working with humans in real world environments, and try to postulate some solutions that are feasible to solve such problems efficiently. There are many challenges that robot interacting with humans, we specify a few that we try to address in this work. The first challenge is to plan under uncertainty in large state–action spaces, which are continuous. The problem is also exacerbated as the number of manipulable objects in the environment increase, as there is a combinatorial explosion in the state–action space with each object the agent can manipulate. In this thesis we will explore hierarchical methods to solve such tasks.

The second challenge is to follow a natural language command to its goal specification. Natural language allows multiple modalities to present commands. Commands can be specified at different orders of granularity, coarse or fine, allowing a range to specify commands like "get to the library" to "take a left turn". Further, commands can be specified with ends or means of the task as the goal. For example, an instruction to "go to the red room" is very different from "go to the red room through the long corridor." In this thesis we will look at methods that ground natural language commands to reward functions hierarchies or plan directly, depending on the modality demanded by the natural language command.

The third challenge involves learning to solve such tasks efficiently. This involves learning hierarchies and spatio–temporal abstractions that construct the hierarchies. We are interested in looking at connections between attribute learning and option learning to construct these hierarchies. Attribute learning previously has been done using trajectories or natural language. We want to combine these ideas to learn hierarchies, which are efficient to plan over.

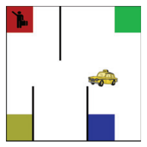
There are other challenges in robotics like partial observability, dialog, vision for robotics, task generalization, etc. which are not the focus of this thesis. In the next sections we would set up the first three challenges in detail along with our proposed solutions.

## The Planning problem

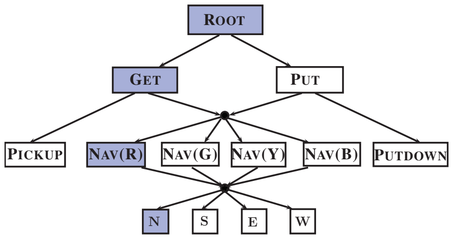
When carrying out tasks in unstructured, multifaceted environments such as factory floors or kitchens, the resulting planning problems are extremely challenging due to the large state and action spaces (Bollini et al. 2012; Knepper et al. 2013). Typical planning methods require the agent to explore the state–action space at its lowest level, resulting in a search for long sequences of actions in a combinatorially large state space. For example, cleaning a room requires arranging objects in their respective places. A naive approach for arranging object might have to search over all possible states by placing all objects in all possible locations, resulting in an intractable inference problem with increasing objects.

One promising approach is to decompose planning problems in such domains into a network of independent subgoals. This approach is appealing because the decision-making problem for each subgoal is typically much simpler than the original problem. There are two ways in which the decision problem can be simplified. First, instead of selecting between actions, the agent can select between subgoals that are recursively solved, decreasing the search depth. Second, the state representation of the world can be compressed to include only information that is relevant to the current decision problem. Importantly, planning algorithms for each subproblem can be custom-tailored, allowing each goal to be solved as efficiently as possible.

We proposed *Abstract Markov Decision Process* (AMDP) hierarchies as a method for reasoning about a network of subgoals (Gopalan et al. 2017 in press), we describe the formalism briefly here. AMDPs offer a model-based hierarchical representation that encapsulates knowledge about abstract tasks at each level of the hierarchy, enabling



(a) Taxi



(b) Taxi AMDP hierarchy

Figure 1: (a) The Taxi problem, where the taxi needs to drop the passenger to their goal; (b) the Taxi AMDP hierarchy, nodes indicate subgoals which are solved using an AMDP or a primitive action. The edges are actions belonging to the parent AMDP. Shaded nodes indicate which subgoals are expanded by AMDPs in a given state. In contrast, bottom-up approaches like MAXQ (Dietterich 2000) expand all nodes in the figure. These savings result in significant total planning computation gains: AMDP planning requires only 3% of the backups that MAXQ requires for the Taxi problem.

much faster, more flexible top–down planning than previous bottom–up methods like MAXQ (Dietterich 2000) or Options (Sutton, Precup, and Singh 1999). An AMDP is an MDP whose states are abstract representations of the states of an underlying *environment* (the ground MDP). The actions of the AMDP are either primitive actions from the environment MDP or subgoals to be solved. An AMDP hierarchy is an acyclic graph in which each node is a primitive action or an AMDP that solves a subgoal defined by its parent. The main advantage of such a hierarchy is that *only* subgoals that help achieve the main task need to be planned for; crucially, plans for irrelevant subgoals are never computed. Another desirable property of AMDPs is that agents can plan in stochastic environments, since each subgoal's decision problem is represented by an MDP. Moreover, each subgoal can be independently solved by any off-the-shelf MDP planner best suited for solving that subgoal.

For example, consider the Taxi problem (Dietterich 2000) shown in Figure 1a and its AMDP hierarchy in Figure 1b. The objective of the task is to deliver the passenger to their goal location out of four locations on the map. The subgoal of Get Passenger, which picks up the passenger from a source location, is represented by an MDP, with lower-level navigation subgoals, Nav(R), and a passenger-pickup subgoal, Pickup. The state space to solve the Get Passenger subgoal need not include certain aspects of the environment such as the Cartesian coordinates of the taxi and passenger. To solve this small MDP when picking up a passenger at the Red location, it is unnecessary to solve for the subpolicy to navigate to the Blue location. Our hierarchy enables a decision about which subgoal to solve without needing to solve the entire environment MDP.

In this top-down methodology, planning is performed by first computing a policy for the root AMDP for the current projected environment state, and then recursively computing the policy for the subgoals the root policy selects. In contrast a bottom up planner like MAXQ or options based planning would compute value functions over the hierarchy by processing the state–action space at the lowest level and backing up values to the abstract subtask nodes. This *bottom-up* process requires full expansion of the state–action space, resulting in large amounts of computation.

Moreover, since the tasks are abstractly defined (for example, "take passenger to blue location"), changing the task description from the "blue" to the "red" location is straightforward, and users do not have to directly manipulate the reward functions at each level of the hierarchy. This abstraction is useful in robotics, as human users can simply change the top-level task description and the required behavior will be achieved by the hierarchy.

Formally, we define an AMDP as a six-tuple ($\tilde{\mathcal{S}}$, $\tilde{\mathcal{A}}$, $\tilde{\mathcal{T}}$, $\tilde{\mathcal{R}}$, $\tilde{\mathcal{E}}, F$). These are the usual MDP components, with the addition of $F : \mathcal{S} \rightarrow \tilde{\mathcal{S}}$, a *state projection* function that maps states from the environment MDP into the AMDP state space $\tilde{\mathcal{S}}$. Additionally, the actions ($\tilde{\mathcal{A}}$) of the AMDP are either primitive actions of the environment MDP, or are associated with subgoals to solve in the environment MDP. The transition function of the AMDP ($\tilde{\mathcal{T}}$) must capture the expected changes in the AMDP state space upon completion of these subgoals. With these action and state semantics, an AMDP, in effect, defines a decision problem over subgoals for the environment MDP.

Naturally, each subgoal for a task must be solved. However, even a single subgoal might be challenging to solve in the environment MDP. Therefore, we introduce the concept of an AMDP hierarchy $H = (V, E)$, which is a directed acyclic graph (DAG) with labeled edges. The vertices of the hierarchy $V$ consist of a set of AMDPs $\mathcal{M}$ and the set of the primitive actions $\mathcal{A}$ of the environment MDP. The edges in the hierarchy link multiple AMDPs together, with the edge label associating the action of an AMDP with either a primitive environment action or a subgoal that is formulated as an AMDP itself. Consequently, an AMDP hierarchy recursively breaks down a problem into a series of small subgoals.

We now describe planning with a hierarchy $H$ of AMDPs. The critical property of our planning approach is to make decisions online in a top-down fashion by exploiting the transition and reward function defined for each AMDP. In this top-down methodology, planning is performed by first computing a policy for the root AMDP for the current projected environment state, and then recursively computing the policy for the subgoals the root policy selects. Consequently, the agent never has to determine how to solve subgoals that are not useful subgoals to satisfy, resulting in significant performance gains compared to bottom-up solution methods. This top-down approach does require that the transition model and reward function for each AMDP are available.

If each AMDP's transition dynamics accurately models the subgoal outcomes, then an optimal solution for each AMDP produces a recursively optimal solution for the whole problem; if the transition dynamics are not accurate, then the error associated with the overall solution can still be bounded as shown in our previous work (Gopalan et al. 2017 in press). Further, as each sub-goal has a local model, we can ground any sub-goal in the DAG depending on the

**Algorithm 1** Online Hierarchical AMDP Planning

```
function SOLVE($H$)
    GROUND($H$, ROOT($H$))
function GROUND($H, i$)
    if $i$ is primitive then                  ▷ recursive base case
        EXECUTE($i$)
    else
        $s_i \leftarrow F_i(s)$               ▷ project the environment state $s$
        $\pi \leftarrow$ PLAN($s_i, i$)
        while $s_i \notin \mathcal{E}_i$ do    ▷ execute until local termination
            $a \leftarrow \pi(s_i)$
            $j \leftarrow$ LINK($H, i, a$)     ▷ $a$ links to node $j$
            GROUND($H, j$)
            $s_i \leftarrow F_i(s)$
```

task specification as shown in the next section.

Pseudocode for online hierarchical AMDP planning is shown in Algorithm 1. Planning begins by calling the recursive *ground* function from the root of $H$. If node $i$ passed to the ground function is a primitive action in the environment MDP, then it is executed in the environment. Otherwise, the node is an AMDP that requires solving. Before solving it, the current environment state $s$ is first projected into AMDP $i$'s state space with AMDP $i$'s projection function $F_i$. Next, any off-the-shelf MDP planning algorithm associated with AMDP $i$ is used to compute a policy. The policy is then followed until a terminal state of the AMDP is reached. Following actions selected by the policy for AMDP $i$ involves finding the node the actions links to in hierarchy $H$, and then calling the ground function on that node. Note that after the ground function returns, at least one primitive action in the environment should have been executed. Therefore, after ground is called, the current state for the AMDP is updated by projecting the current state of the environment with $F_i$.

Planning with AMDPs shows significant improvements in planning times when compared with traditional bottom-up planners or flat planners when tested across different domains as shows in the results of (Gopalan et al. 2017 in press). We also showed a real time planning application for task and motion planning in robotics. In this demo a Turtlebot moved a block to from one room to the goal room in presence of environmental disturbances as shown in our video[1]. This is a hard planning problem with a continuous state–action space, and stochasticity in the environment. The agent shows reactive control to retrieve the block in the video as soon as it is snatched, to move the block to the goal room. For more details please refer (Gopalan et al. 2017 in press).

Hence AMDPs show significant improvements in planning times across multiple domains, even with continuous state–action spaces. Now that we have a tool to plan in large domains, we look next at natural language as an input and the different modalities of inputs, some of which would find the use of AMDP hierarchies useful.

[1]https://youtu.be/Bp3VEO66WSg

## Goal specification with Natural Language

Natural language provides an easy interface for an untrained public to work with robots. Such robots that understand natural language commands must at the very least understand goal based commands that ask the robot to achieve a certain goal configuration. Abstraction is important for achieving such goal conditions because it is much harder to map natural language to a sequence of robot control signals. Instead existing approaches map natural language commands to a formal representation at some fixed level of abstraction (Chen and Mooney 2011; Matuszek et al. 2012b; Tellex et al. 2011). While effective at directing robots to complete predefined tasks, mapping to fixed sequences of robot actions is unreliable when faced with a changing or stochastic environment. Accordingly, (MacGlashan et al. 2015) decouple the problem and use a statistical language model to map between language and robot goals, expressed as reward functions in a Markov Decision Process (MDP). Then, an arbitrary planner solves the MDP, resolving any environment-specific challenges. As a result, the learned language model can transfer to other robots with different action sets so long as there is consistency in the task representation (*i.e.*, reward functions). However natural language problem specification has different different kinds of requirements: granularity, means and ends of task solving, and temporal specification of goals.

First is the aspect of granularity. For example, a brief transcript from an expert human forklift operator instructing a human trainee has very abstract commands such as "Grab a pallet," mid-level commands such as "Make sure your forks are centered," and very fine-grained commands such as "Tilt back a little bit" all within thirty seconds of dialog. Humans use these varied granularities to specify and reason about a large variety of tasks with a wide range of difficulties. Furthermore, these abstractions in language map to subgoals that are useful when interpreting and executing a task. Moreover, MDPs for complex, real-world environments face an inherent tradeoff between including low-level task representations and increasing the time needed to plan in the presence of both low- and high-level reward functions (Gopalan et al. 2017 in press).

To address this problems, we developed an approach for mapping natural language commands of varying complexities to reward functions at different levels of abstraction within a hierarchical planning framework. This approach enables the system to quickly and accurately interpret both abstract and fine-grained commands. Our system uses a deep neural network language model that learns how to map natural language commands to the appropriate level of an AMDP planning hierarchy. By coupling abstraction level inference with the overall grounding problem, we fully exploit the subsequent AMDP hierarchy to efficiently execute the grounded tasks. To our knowledge, we are the first to contribute a system for grounding language at multiple levels of abstraction, as well as the first to contribute a deep learning system for improved robotic language understanding. The results show faster average planning times at all levels of the hierarchy when compared to a base level planner. A demo of the system can be seen here[2]. The system can accept low level commands like "go north" and high level commands like "take the block to the red room."

Next we would briefly describe other natural language grounding problems that interest us. First is problem of the means and ends of task solving, where a user might specify how to solve a task. For example the trajectory for "go to red room through the blue room" is very different from the trajectory for "go to the red room." This problem can be solved by a language model that recognizes when the means of solving a task are more important and would then plan for the task with different sets of planners. Second is the problem of temporal specification of rewards, where a command might be "go to the red room and then go to the blue room." Here, we can parse the language with Linear Temporal Logic (LTL) and create a non-Markovian reward function, where the reward functions switch as a subtask is complete. This formulation would be important to solve temporally extended tasks with multiple subgoal specifications given by the human user. Abstraction would be important in these LTL specification as solving these behavioral problems as the lowest level of abstraction might be computationally intractable. Next we look at how we might learn these abstractions.

## Learning AMDP Hierarchies

The hierarchies that we looked at until now were hand designed, however an agent has to be capable of creating these hierarchical abstractions on its own in the real world. We postulate that natural language provides some clues about the levels of abstraction that a human agent might care about when working with such robots. We have two goals in this section; firstly we need to learn the local models for AMDP hierarchies; secondly a more important goal is to learn an AMDP hierarchy with language and trajectories.

To solve the first part we can use R-max (Brafman and Tennenholtz 2002) on every local model of an AMDP hierarchy. This approach will learn the level 1 models by collecting samples from the environment, but models at every higher level can be learned exactly by sampling from the models learned at level 1. This method would be sample efficient and would enlighten the trade-offs of having a precise, expensive to learn hierarchical model versus a cheap erroneous hierarchical model.

The second and more important goal is to learn an AMDP hierarchy. Konidaris 2016 uses options or temporally extended actions to learn symbols from initiation and termination sets, to create state abstractions and a higher level in the hierarchy. We believe that an important method to learn symbols can be via natural language. Matuszek et al. 2012a learned attributes of objects present in a state to model language and perception together. Borrowing ideas of attribute learning from existing literature, we can create methods to learn symbols and associated abstract states directly from demonstrations, and plan for them using AMDP hierarchies. A simpler idea to test attribute learning might be to learn object parameterized options, akin to parametrized skills, where we learn object attributes with natural language.

[2]https://youtu.be/9bU2oE5RtvU

This learning method would satisfy most of the goals with respect to an agent in the real world that learns from natural language and example trajectories; plans in real time given a natural language command at varying degrees of granularity and temporal specification.

## Conclusion

In this work we look at the problems of understanding natural language groundings, learning efficient hierarchies and planning efficiently to have a robot perform tasks real time in stochastic and large state–action spaces. Our initial results show that the planning problem can be made easier with AMDP hierarchies. We have made some inroads in the natural language grounding problems, where we can specify problems at different levels of granularity to an agent. However, we still have to make large amounts of progress in the problem of learning of a hierarchy. We believe our methods would lead to faster learning of hierarchies and shorter planning times when compared to traditional methods.

## Acknowledgements

This material is based upon work supported by the National Science Foundation under grant number IIS-1637614 and the National Aeronautics and Space Administration under grant number NNX16AR61G.

## References

Bollini, M.; Tellex, S.; Thompson, T.; Roy, N.; and Rus, D. 2012. Interpreting and executing recipes with a cooking robot. In *International Symposium on Experimental Robotics*.

Brafman, R. I., and Tennenholtz, M. 2002. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research* 3(Oct):213–231.

Chen, D. L., and Mooney, R. J. 2011. Learning to interpret natural language navigation instructions from observations. In *AAAI Conference on Artificial Intelligence*.

Dietterich, T. 2000. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research* 13:227–303.

Gopalan, N.; desJardins, M.; Littman, M. L.; MacGlashan, J.; Squire, S.; Tellex, S.; Winder, J.; and Wong, L. L. 2017 in press. Planning with abstract markov decision processes. In *International Conference on Automated Planning and Scheduling*.

Knepper, R.; Tellex, S.; Li, A.; Roy, N.; and Rus, D. 2013. Single assembly robot in search of human partner: Versatile grounded language generation. In *ACM/IEEE International Conference on Human-Robot Interaction Workshop on Collaborative Manipulation*.

Konidaris, G. 2016. Constructing abstraction hierarchies using a skill-symbol loop. In *International Joint Conference on Artificial Intelligence*.

MacGlashan, J.; Babeş-Vroman, M.; desJardins, M.; Littman, M. L.; Muresan, S.; Squire, S.; Tellex, S.; Arumugam, D.; and Yang, L. 2015. Grounding English commands to reward functions. In *Robotics: Science and Systems*.

Matuszek, C.; FitzGerald, N.; Zettlemoyer, L.; Bo, L.; and Fox, D. 2012a. A joint model of language and perception for grounded attribute learning. *arXiv preprint arXiv:1206.6423*.

Matuszek, C.; Herbst, E.; Zettlemoyer, L.; and Fox, D. 2012b. Learning to parse natural language commands to a robot control system. In *International Symposium on Experimental Robotics*.

Sutton, R.; Precup, D.; and Singh, S. 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence* 112(1):181–211.

Tellex, S.; Kollar, T.; Dickerson, S.; Walter, M. R.; Banerjee, A. G.; Teller, S.; and Roy, N. 2011. Understanding natural language commands for robotic navigation and mobile manipulation. In *AAAI Conference on Artificial Intelligence*.

# Learning Planning Operators from Episodic Traces

**David Ménager**
Electrical Engineering & Computer Science
University of Kansas
Lawrence, KS 66045 USA
dhmenager@ku.edu

**Dongkyu Choi**
Aerospace Engineering
University of Kansas
Lawrence, KS 66045 USA
dongkyuc@ku.edu

**Mark Roberts, David W. Aha**
Naval Research Laboratory, Code 5514
Washington, DC, 20375 USA
mark.roberts@nrl.navy.mil
david.aha@nrl.navy.mil

**Abstract**

Learning is an important aspect of human intelligence. People learn from various aspects of their experience over time. We present an episodic infrastructure for learning in the context of a cognitive architecture, ICARUS. After a review of this architecture, we formally define the architectural extensions for episodic capabilities. We then demonstrate the extended system's capability to learn planning operators using the episodic traces from two Minecraft-like scenarios.

## 1 Introduction

Learning is of central importance to intelligent agents. From the beginning of artificial intelligence back in 1950's, researchers have recognized that the learning process is intimately tied to the nature of intelligence (Simon 1980). In order to adapt to dynamic environments, intelligent agents must possess mechanisms that allow them to acquire a broad repertoire of relevant behaviors. For this reason, there has been a significant amount of research on learning domain models in a variety of manners. But we rarely find any theories that provide a complete account of how experiences are gathered and how knowledge is derived from such experiences over time.

Our research aims to provide an infrastructure for organizing and processing collected experience, which then establishes a foundation for an experiential learning in intelligent agents. We model human *episodic* capabilities (Tulving 1983) in the context of a cognitive architecture, ICARUS (Langley and Choi 2006), and attempt to bridge these capabilities with other learning modalities. In this paper, we begin our study with the experiential learning of planning operators including action and event models. This will produce agents capable of learning throughout their lives to develop low-level expertise and adapt to dynamic environments. Such agents will also be able to recover from incorrect or incomplete knowledge over time. Additionally, because ICARUS learns structured models, agents retain the advantage of explainability.

Our work is motivated by situated agents that learn in changing, dynamic environments. Certainly, robots are one kind of such agent, but this paper focuses on a simulated domain described in the next section. After a description of this illustrative domain, we review the ICARUS architecture by providing necessary definitions that contextualize the episodic extensions we describe next. Then we present some preliminary results in the domain. Finally, we will discuss related work before we conclude.

## 2 Illustrative Domain

To motivate our research on episodic agents and evaluate our system's capabilities, we use a simplified version of a popular open-world game, Minecraft (Johnson et al. 2016), where players attempt to survive in a continuous, dynamic world by collecting resources, forging tools, building structures, and fighting enemies. Consider a novice agent learning from an expert player who starts at the lower left corner of a room. There are resources scattered around the room and a craft desk nearby the player. The player should gather the resources to make a sword for protection, but there are zombies in this room that guard the resources. The player must be careful because she will lose health if a zombie attacks her.

The expert player starts by selecting a resource and moving north toward it. Once she is on the same row as the resource, the player moves east toward it until she is on the same column. Now the player is standing by the resource and picks up the resource to hold it. But there was a zombie in the same location, so the player's health was reduced while the player was standing there. Then she moves south and then west to the craft desk. When the player arrives there, she puts down the resource on the desk. After repeating this process several times, the expert player would have gathered all the resources necessary to build a sword and achieve its mission by crafting one. The novice observer stores in its mind all the situations the expert has encoun-

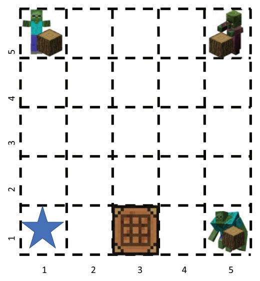


Figure 1: A 5x5 notional plot of Minicraft.

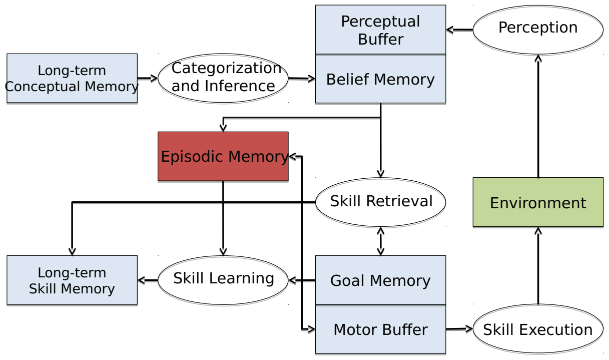


Figure 2: ICARUS cycle prior to episodic memory extension.

tered, and learns action and event models from them that it will be able to use to play the game.

We began our work by creating a grid world, *Minicraft*, that is inspired by the original Minecraft. Although simplified, this game captures enough dynamism to demonstrate the learning ability of our system. Figure 1 shows a notional view of Minicraft, which consists of four entities: resource, craftdesk, zombie, and the agent. The only entities with dynamic properties are the zombie and the agent. The agent begins at the star and moves one grid at a time while picking up or dropping resources and crafting items. Zombies, once placed on the map, are stationary, but provide dynamism to the world by decreasing the agent's health by one for every moment that the agent resides in the same grid as the zombie. All world dynamics, such as the effects of movement and action are unknown to the observer.

## 3 ICARUS Review

As a cognitive architecture, ICARUS provides a framework for modeling human cognition and programming intelligent agents. The architecture makes commitments to its representation of knowledge and structures, the memories that store these contents, and the processes that work over them. ICARUS shares some of these commitments with other architectures like Soar (Laird 2012) and ACT-R (Anderson and Lebiere 1998), but it also has distinct characteristics like the architectural commitment to hierarchical knowledge structures, teleoreactive execution, and goal reasoning capabilities (Choi 2011). Section 3.1 describes the key knowledge and memory structures of ICARUS, while Section 3.2 outlines how processes operate on these memories as part of a cognitive cycle.

ICARUS learns in the context of propositional states and action event models. Given a finite set of first order propositions $P$ we define a propositional language $\mathcal{L}(P)$, and a finite set of labeled procedures, called *actions*, $\mathcal{A}$ such that $\mathcal{L}(P) \cap \mathcal{A} = \emptyset$.

### 3.1 Representation and Memories

ICARUS distinguishes two main types of knowledge: concepts and skills which represent semantic and procedural knowledge, respectively. Both have parameterized (i.e., lifted) variants that are grounded when variables are assigned to objects. Figure 2 shows the long-term and short-term memories of ICARUS, in which concepts and skills are stored. Paramaterized concept and skill definitions are stored in conceptual and procedural long-term memories, respectively. Instances of these definitions are stored in their respective conceptual or procedural short-term memories.

*Concepts* describe certain aspects of a situation in the environment. They resemble horn clauses (Horn 1951), complete with a predicate as the head, perceptual matching conditions, tests against matched variables, and references to any sub-relations.

**Definition 1 (Concepts ($C$))** *A* ***primitive*** *concept is defined over $P$ as $c_i = \langle \lambda, \epsilon \rangle$ where $\lambda \in P$ known as the concept head, $\epsilon$ denoting elements to pattern match in the world state $S$, where $S$ is a subset of $P$. Let $C_p$ be the set of primitive concepts. A* ***non-primitive*** *concept is defined over $P \cup C_p$ as $c_j = \langle \lambda, \epsilon, \gamma \rangle$ where $\gamma$ denotes $c_j$'s subrelations. We can further define non-primitive concepts over $P \cup C_p \cup C_n$, where $C_n$ is the set of non-primitive concepts.*

Figure 3 shows example concepts for Minicraft. The first, `north-of`, is a primitive concept that describes the situation where a zombie is to the north of the agent, using perceptual matching and test conditions for *self* and *zombie*. The second, `on-horizontal-axis`, depicts a non-primitive concept where a zombie is on the same horizontal line as the agent. The third, `standing-by`, describes an even more abstract non-primitive concept where the zombie is standing right next to the agent.

*Skills* describe procedures to achieve certain concept instances in the environment. These are hierarchical versions of STRIPS operators (Fikes and Nilsson 1971) with a named head, perceptual matching conditions, preconditions that need to be true to execute, direct actions to perform in the

```
((north-of ?o1 ?self)
   :elements ((self ?self y ?y) (zombie ?o1 y ?y1))
   :tests ((> ?y1 ?y)))
((on-horizontal-axis ?o1 ?self)
   :elements ((self ?self) (zombie ?o1))
   :conditions ((not (north-of ?o1 ?self))
                (not (south-of ?o1 ?self))))
((standing-by ?self ?o1)
   :elements ((self ?self) (zombie ?o1))
   :conditions ((on-horizontal-axis ?o1 ?self)
                (on-vertical-axis ?o1 ?self)))
```

Figure 3: Three ICARUS concepts in the Minicraft domain.

```
((gather-resource ?o1)
   :elements ((self ?self) (resource ?o1))
   :conditions ((not (carrying ?any))
                (standing-by ?self ?o1))
   :effects ((carrying ?o1))
   :actions ((*pick-up-resource ?o1)))
((go-to ?o1)
   :elements ((self ?self))
   :conditions ((north-of ?o1 ?self))
   :subskills ((go-up-to ?o1))
   :effects ((standing-by ?self ?o1)))
((gather-resource ?o1)
   :elements ((self ?self) (resource ?o1))
   :conditions ((not (carrying ?any)))
   :subskills ((go-to ?o1) (gather-resource ?o1))
   :effects ((carrying ?o1)))
```

Figure 4: Three ICARUS skills in the Minicraft domain.

world or any sub-skills, and the intended effects of the execution.

**Definition 2 (Skills ($K$))** *Given the finite set of actions $\mathcal{A}$, a skill defined over $C \cup S$ where $C$ is the set of concepts and $S$ is a propositional state, is a* primitive *skill if $k_i = \langle \epsilon, \gamma, \alpha, \sigma, \eta \rangle$, where pattern match conditions $\epsilon \subseteq S$, preconditions $\gamma \subseteq \{\lambda | \langle \lambda, \cdot \rangle \in C\}$, actions $\alpha \subseteq \mathcal{A}$, sub-skills $\sigma = \emptyset$, and effects $\eta \subseteq \{\lambda | \langle \lambda, \cdot \rangle \in C\}$. Let $K_p$ be the set of primitive skills.*

*A skill defined over $C \cup S \cup K_p$ is a* non-primitive *skill if $k_j = \langle \epsilon, \gamma, \alpha, \sigma, \eta \rangle$, where $\epsilon \subseteq S, \gamma \subseteq \{\lambda | \langle \lambda, \cdot \rangle \in C\}, \alpha = \emptyset, \sigma \subseteq K_h$, and $\eta \subseteq \{\lambda | \langle \lambda, \cdot \rangle \in C\}$. $K_h$ is the set of non-primitive skills.*

Figure 4 shows example skills for Minicraft. The first, `gather-resource`, is a primitive skill that describes a procedure to collect a resource that is executable when the agent is not carrying anything and is standing next to the resource. This skill uses a direct action to pick up the resource and its intended effect is carrying the resource. The bottom two are non-primitive skills that use sub-skills: `go-to` uses a sub-skill `go-up-to` to achieve the goal of standing near the object, while `gather-resource` uses the two sub-skills above it to collect a resource.

## 3.2 The ICARUS Cognitive Cycle

The ICARUS architecture operates in a cognitive cycle repeating two steps: conceptual inference and skill execution. *Conceptual inference* is the process of creating concept instances (i.e., beliefs). At the beginning of each cycle, the system receives sensory input from the environment as a list of objects with their attribute-value pairs; this can be thought of as the world state and is represented as propositions. Based on this information, the architecture infers the concept instances (i.e., beliefs) that are true in the current state by matching its concept definitions to perceived objects and other concept instances in a bottom-up fashion.

In summary, Figure 2 shows concept definitions housed in the conceptual long-term memory are used to infer the beliefs of the system from the world state and are stored as concept instances in the conceptual short-term memory.

**Definition 3 (Beliefs (B))** *Let $C$ be the set of concepts. $\forall c = \langle \lambda, \epsilon, \gamma, \tau \rangle \in C$, $\exists$* belief *$b = \langle \lambda, \epsilon, \gamma, \tau, \beta \rangle$ where $\beta$ represents bindings that ground $b$ on the perceptual elements, $\epsilon$. Let $B$ be the set of all possible beliefs, and let $\mathcal{B} = 2^B$ be the set of all* belief states. *A* belief state *$s \in \mathcal{B}$.*

*Skill execution* proceeds after conceptual inference whereby ICARUS finds all the relevant skill definitions for the current goal(s) that are executable based on the current beliefs. ICARUS chooses a skill and sets it as its *intention* and executes it in the world.

**Definition 4 (Intentions ($\iota$))** *Let $K$ be the set of skills. $\forall k = \langle \epsilon, \gamma, \alpha, \sigma, \eta \rangle \in K$, there exists* intention *$\iota = \langle \epsilon, \gamma, \alpha, \sigma, \eta, \beta \rangle$ where $\beta$ represents bindings that ground $\iota$ in the belief state.*

Each cycle may introduce changes in the environment, which may modify the sensory input for the next cycle, resulting in new beliefs and intentions. The architecture iterates in this manner until all of its goals are achieved or its operations are terminated for any other reasons.

## 4 Constructing Episodes

We now shift our attention to extending ICARUS with an episodic memory. In particular, we highlight the core data structures of ICARUS's Episodic Memory (Section 4.1), how it encodes episodes within that memory through a process called event segmentation (Section 4.2), and how it generalizes episodes over time (Section 4.3).

### 4.1 The Episodic Memory

The episodic memory in ICARUS is a long-term, cue-based memory that the agent uses to deliberately encode and retrieve episodes. The architecture organizes its episodic memory $E = \langle \rho, \mathcal{F}, \mathcal{T} \rangle$ in a compound structure composed of an episodic beliefs-action cache $\rho$, a concept frequency forest $\mathcal{F}$, and the episodic generalization tree $\mathcal{T}$.

Figure 5 shows how information is processed within the episodic memory and is discussed through this section. $\rho$ acts as a storage for the agent's unprocessed history. We assume that the agent has sufficient memory to store the complete beliefs-action sequence. $\mathcal{F}$ records counts for the number of times concepts and their instantiations as beliefs have occurred during the execution of the agent. $\mathcal{T}$ is the main data structure that organizes and stores episodes; the contents of $\mathcal{T}$ are used in the process of learning new skills. The elements $\rho$ and $\mathcal{F}$ (Definitions 5 and 6), discussed next, facilitate the workings of the event segmentation and episodic encoding (Section 4.2). Generalization with $\mathcal{T}$ is discussed in Section 4.3.

Since episodes are built on top of sequences of beliefs, we introduce first the beliefs-action cache, which stores the moment-by-moment changes in belief, inferred from the world state, as well as the actions that were taken based on those beliefs.

**Definition 5 (Beliefs-action cache ($\rho$))** *The* beliefs-action cache $\rho$, *is an ordered sequence of belief-action pairs. This cache stores a complete, detailed history of what the agent observed. Figure 5 shows that the contents of the belief memory are inputs to the beliefs-action cache.*

Once these traces are collected, they must be processed for *interesting* events, which are tracked in the concept frequency forest.

**Definition 6 (Concept frequency forest ($\mathcal{F}$))** *Let $X$ be a set of location predicates, and let $Y = \{x.first | x \in S\}$ be the set of object types.* A concept frequency tree *is a tree whose the root $\mu$ is a location predicate from $X$. The children of $\mu$ are all the concepts the agent has observed in that location. For each child concept, c, of $\mu$, $\exists$ a set of types from $Y$, to specify concept disjunctions. Under each disjunction, $j$, there exists concept instances. Each node in the tree has a count field, denoting the number of times this node has been observed.* A concept frequency forest *is a collection of concept frequency trees.*

ICARUS uses $\mathcal{F}$ to model *expectation violation*. The agent sets two thresholds: one for positive expectations and one for negated expectations. Any belief with a conditional probability, given the location, is greater than the positive threshold is said to be *expected*. Any belief with a conditional probability, given the location, is less than the negated threshold is *not expected* to be in the state. A belief that violates an expectation is a *significant belief*, which prompt the system to create an episode. This is a primitive method for novelty detection that only uses spatial information, but we can further extend the novelty detection method to include the temporal domain as well.

The episode structure defined in Definition 7 represents the agent's experiences in the architecture. Once they are stored in memory, episodes are processed to abstract general rules that allow the agent to predict environmental dynamics.

**Definition 7 (Episode ($\varepsilon$))** *An episode is a tuple $\langle B_s, B_e, \Sigma, \psi \rangle$, where $B_s$ is the start state of the episode, $B_e$ is the end state of the episode, $\Sigma$ is the set of significant beliefs in $B_e$, and $\psi$ is a count for the number of times the episode has occurred.*

During episodic encoding, the start and final states are taken from the $\rho$ (i.e., the beliefs-action cache). In the current implementation, $B_s$ and $B_e$ are consecutive belief states, but our work does not require this. Our rationale is psychologically inspired. When humans perform low-level actions, kicking a soccer ball for instance, humans know that the effect is not always observed in their next cognitive cycle. The ball travels in time before it reaches the goal. This dynamic is readily understood by most humans. Modeling actions with temporally delayed effects is part of our future work.

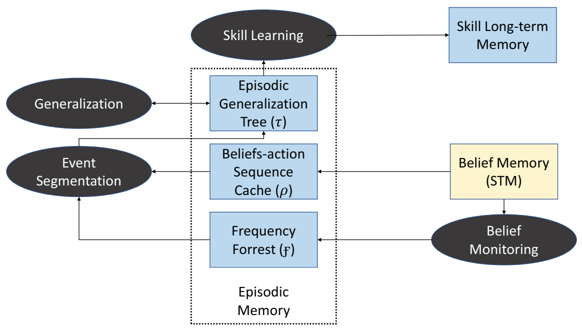


Figure 5: Block diagram depicting episodic memory components and information flow starting from the belief memory.

## 4.2 Episodic Encoding

Episodic Encoding in ICARUS is a two-step process. First, ICARUS operates on the $\rho$ to returns a new episode $\varepsilon$. This is referred to as "Event Segmentation" in Figure 5. Once the episode exists, the second process places it into the episodic generalization tree. Algorithm 1 shows that encoding is triggered by the presence of one or more significant beliefs in belief state.

Algorithm 2 traces how episodes are inserted into the episodic generalization tree. Suppose the generalization tree contains several episodes. $\Gamma$ is a list of sibling episodes under parent $\varrho \in \mathcal{T}$ If $\forall \varepsilon_i \in \Gamma, (\varepsilon_i, \varepsilon) \notin E$ then $(\varrho, \varepsilon) \in E$. That is $\varepsilon$ becomes a child of $\varrho$. A new episode has successfully been encoded into the episodic memory. If $\exists \varepsilon_j \ni \varepsilon_j = \varepsilon$, then the counter for $\varepsilon_j$ increments by one and $\varepsilon$ is not inserted.

On every cycle, ICARUS records the belief state and executed actions into the episodic cache and updates $\mathcal{F}$. When the agent infers one or more significant beliefs, it encodes

**Algorithm 1** CREATEEPISODE($\rho, loc, B_c$)

```
1: $\rho$ is beliefs-action cache
2: $loc$ is current location
3: $B_c$ is current belief state
4: $B_{prev} \leftarrow$ last state in $\rho$
5: $\rho \leftarrow \rho.add(B_c, a)$
6: $sigs \leftarrow$ GETSIGNIFICANTBELIEFS($B_c, loc$)
7: if not NULL($sigs$) then
8:     $\varepsilon \leftarrow$ MAKEEPISODE($sigs, B_c, B_{prev}$)
9:     $\mathcal{T} \leftarrow$ INSERT($\varepsilon, \mathcal{T}$)
```

**Algorithm 2** INSERTEPISODE($\varepsilon$, $\mathcal{T}$)

```
1:  $queue \leftarrow \emptyset$
2:  $temp \leftarrow$ root of $\mathcal{T}$
3:  $match \leftarrow \emptyset$
4:  $p \leftarrow \emptyset$
5:  while not NULL($temp$) do
6:      $match \leftarrow$ STRUCTURALEQ?($temp$, $\varepsilon$)
7:      if $match$ is exact match then
8:          $temp.count \leftarrow temp.count + 1$
9:          Try to learn from $temp$ if count high enough
10:         BREAK
11:     else if $match$ is bc of unification then
12:         $temp.count \leftarrow temp.count + 1$
13:         $queue \leftarrow \emptyset$
14:         $queue \leftarrow temp$'s children
15:         Try to learn from $temp$ if count high enough
16:         $p \leftarrow temp$
17:     $temp \leftarrow queue$.FIRST
18:     $queue \leftarrow queue$.POP
19: if null($temp$) and $match$ not exact then
20:     $p \leftarrow p$.ADDCHILD($\varepsilon$)
21:     $\mathcal{T} \leftarrow$ GENERALIZE($p$, $\varepsilon$)
```

a new episode. The root node of the generalization tree is the most general episode and is allowed to have an arbitrary number of children. Under the root, episodes are grouped according to *structural similarity*. Two episodes $e_1, e_2$ are *structurally similar* if their significant beliefs unify. By "unify" we mean that there must exist a binding set that transforms the significant beliefs of $e_1$ to those of $e_2$ and vise versa. This is a rigid generalization scheme that needs more consideration in future work. Each child is a *k*-ary tree where $k \in \mathbb{N}$. Episodes become more specific at each decreasing level of the tree according to structural similarity. At the leaf nodes exist fully instantiated episodes.

### 4.3 Episodic Generalization

ICARUS supports generalization of the episodic tree during encoding of episode, $\varepsilon_i$. Definition 8 shows that an episode hierarchy is induced by structural similarity. Two sibling episodes $\varepsilon_i, \varepsilon_j$ generalize iff $\exists$ episode $\varepsilon_g$ such that $(\varepsilon_g, \varepsilon_i) \in E$ and $(\varepsilon_g, \varepsilon_j) \in E$, but $(\varepsilon_i, \varepsilon_g) \notin E$ and $(\varepsilon_j, \varepsilon_g) \notin E$. This means that $\varepsilon_g$ unifies with its children, $\varepsilon_i, \varepsilon_j$, but its children cannot unify with it because they contain more specified bindings. If $\varepsilon_g$ exists, ICARUS tests to see if it is still more specific than the parent of $\varepsilon_i$. If so, then $\varepsilon_g$'s parent becomes $\varepsilon_i$'s parent and $\varepsilon_g$'s children become $\varepsilon_i, \varepsilon_j$. The count for a generalized episode is the summation of the count of its children.

**Definition 8 (Generalization tree ($\mathcal{T}$))** *An* episodic generalization tree *is a tree (V, E) where V is a set of episodes, and E is a set of edges. For any $\varepsilon_i, \varepsilon_j \in V, (v_i, v_j) \in E$ if they are structurally similar. An episode is said to be generalized or* partially instantiated *if the bindings contain one or more unbound variables.*

The generalization tree naturally lends itself to the learning process as a result of generalization. For example, if person $x$ drops a glass on the ground and it breaks, and person $y$ drops a glass on the ground and it breaks as well, ICARUS forms a generalized episode that implies if anyone drops a glass on the ground, it will break. The ability to gain knowledge in this way is central to general intelligence. As the tree adds more episodes, they are sorted into increasingly sensible taxonomies. The resulting tree after insertion is ICARUS' best estimate of the ideal generalization tree. This organizational structure was inspired by the incremental concept formation literature (Gennari, Langley, and Fisher 1989). As episodes become more general, the skills ICARUS learns from those episodes are equivalently general. So, generalizing skills is performed within the episodic generalization tree, not the skill learning algorithm.

## 5 Skill Learning using the Episodic Memory

In previous work, ICARUS supported learning by observing problem solving traces that include goals, conditions, and the skills used (Nejati 2011). The system relied on the explanations it generated based on the given trace, and this process required, at the very least, primitive skills in ICARUS' memory. In the current work, we start with only the concepts that are sufficient to describe situations in the world but the agent does not have any skills in its knowledge base.

ICARUS starts as an observer and records the history of belief states and ground actions in its episodic memory. As its experience accumulates, the agent will insert an episode whose count surpasses a predefined threshold for model learning. At that moment, the system uses the actions from $B_s \rightarrow B_e$ as a search cue for collecting other episodes where that ordering of actions took place. This trace of episodes is then used in the rule induction algorithm, MLEM2 (Grzymala-Busse and Rzasa 2010). Although we are using MLEM2, this need not be the case. Any rule learning algorithm may be used as long as there is a transformation from ICARUS's representation of experience to the representation that the learning algorithm requires. After learning, the agent can seamlessly utilize the learned skills during problem solving.

### 5.1 Learning Action and Event Models

In order to learn models of the world, ICARUS must first retrieve experiences via a retrieval cue. The system generates an observation, as defined in Definition 9 for each episode that matches the cue. For the case of model learning, the retrieval cue is some subset of actions $a_i$ from $\mathcal{A}$. As the episodes are examined, matches are collected into an episodic trace of evidence related to $a_i$.

**Definition 9 (Observations (O))** *Let $o = \langle s_i, a_i, s_f \rangle$ be an* observation *from $\rho$, the beliefs-action cache, where $s_i, s_f \in \varsigma$ are respectively initial and final belief states, and $a_i \subseteq \Lambda$ be the set of actions that transformed $s_i$ to $s_f$ An* episodic trace*, O is a collection of observations.*

MLEM2 learns rules from data tables, therefore, once the episodic trace is obtained it needs to be transform $O$ into a table. The x-axis for this table is an enumeration of all the

| Belief | $\ell_b$ | $\ell_b \cap \{1,3,4\}$ |
|---|---|---|
| (holding sword1) | {1,2,3,4,5} | {1,3,4} |
| (holding nothing) | {6} | $\emptyset$ |
| (holding food1) | {7,8} | $\emptyset$ |
| (next-to ?zombie) | {1,3,4,7} | {1,3,4} |
| (next-to tree1) | {5,2, 6,8} | $\emptyset$ |
| (health good) | {1,2,3,4,5,6,7,8} | {1,3,4} |

Table 1: Sample attribute and decision blocks.

unique beliefs in $O$, and the y-axis numbers each observation in $O$. Each belief, $b$ on the x-axis has an associated list , $block_b = \{i | \langle s_j, a_j, s_k \rangle \in O[i], b \in s_j\}$ of the observation indices it appeared in. The last column of the data table is the list of the effects, $fx$ for each associated observation. Table 1 summarizes the data table in a way that clearly shows each belief's block list. For example, the middle column states for the first row, that the (holding sword1) belief was present in observations 1 through 5.

For each effect, $f$ in $fx$, the algorithm computes a list, $block_{fx} = \{i | \langle s_j, a_j, s_k \rangle \in O[i], f \in s_k\}$ of observation indices that it appeared in as well. MLEM2 tries to find, for each effect, conditions whose associated blocks cover the effect block. These coverings are what are the learned action and event models.

In this example, assume $a_i = ((*attack))$, and $fx = \{((\textit{zombie-dead ?zombie}), \{1,3,4\}), ((\textit{wood wood1}), \{2\})\}$.[1] MLEM2 attempts to find local coverings of $fx$ from the list of belief conditions. MLEM2 tries the pair $(b, block_b)$ whose listing, $block_b$ intersected with an uncovered effect $block^0_{fx} = \{1,3,4\}$ is the largest. If $block_b \leq block^0_{fx}$, then that condition becomes a rule that covers that effect. If $block_c \nleq block^0_{fx}$ then other conditions need to be added to cover it. Once a rule has been found that covers all the cases of for an effect, the same process repeats for the uncovered effects in $fx$. In the example, the system learns the following rule: (next-to ?zombie) $\cap$ (holding sword) $\rightarrow$ (zombie-dead ?zombie).

In the ICARUS context, MLEM2 results are converted to action and event models, which are primitive skills. The left hand side of the rules become the preconditions, the right hand side would be the effects of the skill. The action information would capture what work needs to be done to realize the effects.

## 6 Experimental Setup

The goal with this research was to create an agent that could learn unknown domain dynamics from experience. Furthermore, we want a system that is flexible and continues learning over the course of its life to reflect the changes in the world's changing dynamics. We assume that the world is fully observable, and that the agent has a vocabulary that distinguishes belief states perfectly. Also, we assume effects come immediately after actions, and that the environment is not stochastic.

We tested on two scenarios. Each scenario has one expert with perfect concept and skill knowledge, and one observer with full observability of the state, perfect concept knowledge, but no skill knowledge (i.e., no knowledge of the domain dynamics). We are primarily interested in what action and event models the agent learns and know how they change in response to new evidence. In the first scenario, we place the expert at (1,1), and zombies and resources are at the other three corners. At (5, 1) there exists a craftdesk. The expert is tasked with collecting resources and placing them on the craftdesk. For the case of the expert, this problem is easily solved, but for the novice, we are interested in how well it learns the dynamics of the world. An example of an event model would be knowing that being next to a zombie reduces the agent's health, and an example of an action model would be learning about what happens to the state when the agent moves.

The second scenario extends the first with the zombies and resources have been randomly re-assigned to different corners. This makes for two different, but structurally identical scenarios. By doing this, we ensure that the agent constructs episodes that will generalize with the other episodes in its memory.

```
(achieve-bottom-horizontal-axis-and-more)
   :conditions ((at minicraft) (north-of r1 me)
               (north-of r3 me) (east-of r2 me)
               (east-of r3 me) (east-of craftdesk1 me)
               (north-of zombie2 me) (north-of zombie3 me)
               (east-of zombie1 me) (east-of zombie2 me)
               (good-health me) (on-ground r1)
               (on-ground r2) (on-ground r3)
               (on-vertical-axis r1 me)
               (on-vertical-axis zombie3 me)
               (on-horizontal-axis zombie1 me)
               (on-horizontal-axis craftdesk1 me)
               (on-horizontal-axis r2 me))
   :actions ((*move-up))
   :effects ((south-of ?r3 me) (south-of craftdesk1 me)
               (south-of ?zombie3 me)
               (bottom-of-horizontal ?zombie3)
               (bottom-of-horizontal ?r3)
               (bottom-of-horizontal craftdesk1))

(achieve-bottom-horizontal-axis-and-more)
   :conditions ((on-horizontal-axis ?r3 me)
                  (on-horizontal-axis craftdesk1 me)
                  (on-horizontal-axis ?zombie3 me))
   :actions ((*move-up))
   :effects ((south-of ?r3 me) (south-of craftdesk1 me)
               (south-of ?zombie3 me)
               (bottom-of-horizontal ?zombie3)
               (bottom-of-horizontal ?r3)
               (bottom-of-horizontal craftdesk1))
```

Figure 6: Learned action models for the `*move-up` action before (top) and after (bottom) generalization.

## 7 Results

We demonstrate that the agent is able to learn goal-directed, specific or generalized action and event models from experience. Because of the episodic memory, ICARUS agents have a mechanism for experiential learning which allows them to learn world dynamics in the form of ICARUS skills. The learned skills are continually revised according to evidence.

Figure 6 demonstrates how the action model for moving

```
(achieve-fair-health-and-more)
   :conditions ((good-health me) (on-ground r1)
                (healthy-standing-by zombie3))
   :actions (nil)
   :effects ((fair-health me) (slouching-by me zombie3))
```

```
(achieve-fair-health-and-more)
   :conditions ((good-health me)
                (healthy-standing-by ?zombie2))
   :actions (nil)
   :effects ((fair-health me) (slouching-by me ?zombie2))
```

Figure 7: Action models for the event model before learning (top) and after learning (bottom).

up changes with experience. The initial action model in Figure 6 (top) contains many irrelevant conditions, while the final version (bottom) contains no irrelevant conditions; not shown are intermediate versions. The same is true for the event model the agent learns for achieving (fair health). Figure 7 shows that the irrelevant condition is removed from the event model by the last refinement, where the event model also successfully generalizes the initial version (top) to the final version (bottom).

In our framework the system learns models based on the agent's interpretation of the ground truth. This is interesting because it clarifies certain properties of inference. Specifically, if an agent is lacking conceptual vocabulary to describe situations, its learned models will show evidence of stochasm. In other words, there will be cases where the same action occurred in identical belief states resulting in different effects.

## 8 Related Work

Earlier research in action recognition and learning aims to teach robots to recognize and perform human gestures (Yang, Xu, and Chen 1997). In that work the researchers used a discrete hidden Markov model to decode human intentions, and to learn the motor actions that controlled making gestures. Along this line, Liu et al. (2017) recently developed a multi-task learning system that hierarchically recognizes human actions. Also, another recent approach attempted to learn control policies for continuous, non-Gaussian stochastic domains (Wang et al. 2017). The work describes a reinforcement learning system that learns an incomplete policy for a discrete controller. Given the policy, a robot executes the action for the nearest state to the current one.

The main distinction from our work and these is that they do not learn action models in the way that we have defined them. The action models these systems learn are often limited to scenario-specific transition functions, and control policies. The semantic meaning of actions, however is still unknown to the agent, so planning with the notion of explicit goals is not possible. Moreover, when these system refer to action models they typically refer to modeling the human motor controls that produce gestures.

In addition to machine learning, researchers are also trying to learn operator descriptions that can be used in performance systems. As Langley and Simon point out, our goal is to understand and characterize the invariants of intelligence. Building systems that help explain how novices become experts in general is key to this endeavor. Wang et al. (1994) created a system built on PRODIGY (Carbonell et al. 1991) that incrementally learned planning operators based on STRIPS (Fikes and Nilsson 1971) via observation and practice. Expert demonstrations allowed the system to estimate initial versions of the operators. The agent refined its knowledge base by attempting to use learned operators to solve problems. The system was able to learn subgoal orderings for the operators, but the system could not learn operator decompositions, so operators were learned and stored in a flat structure. Gil et al. (1994) discussed how imperfections in domain knowledge do not always lead to planning or execution failures. They also presented a system that learns to refine imperfect operators by experimenting. The experimentation process can refine both operator pre and post conditions.

Another system, ALPINE provided methods for inducing abstraction hierarchies over operators (Knoblock 1990). Given a set of low-level operators, the system could induce abstraction hierarchies that reduced the search space.

Another interesting approach learned operators with associated numeric attributes to denote the utility of a particular operator (García-Martínez and Borrajo 2000). In this way the system favored more accurate operators. Walsh and Littman (2008) addressed the problem of efficiently learning STRIPS-like operators via experience. They define their own notion of an episode to be an initial state, $s_0$ goal state, and all state-action pairs following $s_0$ until the problem is solved or marked unsolvable. Their notion of episode, however, is not tied to a larger theory of episodic memory.

Lastly,Molineaux and Aha (2014) describe a surprise-driven method for learning event models. Given a problem, the system returned a plan of actions that would achieve the goal as well as a sequence of expected state changes caused by executing those actions. The system notices surprises when discrepancies exist between actual and expected state transitions. Discrepancies trigger an explanation module, DISCOVERHISTORY to hypothesize the cause of the discrepancies. When explanations fail, the system uses a variant of FOIL to learn an action model that repairs broken explanations.

In our work we addressed the problem of model learning from the vantage point of episodic memory for intelligent agents. Other research has investigated episodic memory. In the work most similar to ours, Nuxoll and Laird (2007) extended the Soar architecture (Laird, Newell, and Rosenbloom 1987) with episodic memory. They present results for action modeling in their work, but details about the learning mechanism are left out. There are also significant theoretical differences between the episodic memory in ICARUS and Soar. ICARUS has strong commitments to hierarchical organization of knowledge throughout the architecture, which helps support our theory for incremental learning. Soar, although it has had many successes, does not have such strict commitments to hierarchy. In their architecture episodes are stored in a flat container for experiences. Moreover, episodes in ICARUS have temporal components, meaning that they contain a sequence of states, whereas Soar's episodes do not have any temporal dimension.

## 9 Conclusion

We presented a new extension to the ICARUS architecture that allows agents to learn goal-directed planning operators from episodic traces. Our results from the Minicraft domain showed that our theory incrementally learns skills in a specific-to-general manner, and also refines skills based on evidence. This evidence is collected from ICARUS episodic memory, a dedicated facility for constructing, storing and organizing experience.

## Acknowledgments

The research presented in this paper was supported in part by the Naval Research Laboratory under contract number N00173-17-P-0837 and the Naval Research Enterprise Internship Program.

## References

Anderson, J. R., and Lebiere, C. 1998. *The atomic components of thought*. Mahwah, NJ: Erlbaum.

Carbonell, J.; Etzioni, O.; Gil, Y.; Joseph, R.; Knoblock, C.; Minton, S.; and Veloso, M. 1991. Prodigy: An integrated architecture for planning and learning. *ACM SIGART Bulletin* 2(4):51–55.

Choi, D. 2011. Reactive goal management in a cognitive architecture. *Cognitive Systems Research* 12:293–308.

Fikes, R., and Nilsson, N. 1971. STRIPS: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2:189–208.

García-Martínez, R., and Borrajo, D. 2000. An integrated approach of learning, planning, and execution. *Journal of Intelligent and Robotic Systems* 29(1):47–78.

Gennari, J. H.; Langley, P.; and Fisher, D. 1989. Models of incremental concept formation. *Artificial intelligence* 40(1-3):11–61.

Gil, Y. 1994. Learning by experimentation: Incremental refinement of incomplete planning domains. In *International Conference on Machine Learning*, 87–95.

Grzymala-Busse, J. W., and Rzasa, W. 2010. A local version of the mlem2 algorithm for rule induction. *Fundamenta Informaticae* 100(1-4):99–116.

Horn, A. 1951. On sentences which are true of direct unions of algebras. *Journal of Symbolic Logic* 16(1):14–21.

Johnson, M.; Hofmann, K.; Hutton, T.; and Bignell, D. 2016. The malmo platform for artificial intelligence experimentation. In *IJCAI*, 4246–4247.

Knoblock, C. A. 1990. Learning abstraction hierarchies for problem solving. In *AAAI*, 923–928.

Laird, J. E.; Newell, A.; and Rosenbloom, P. S. 1987. Soar: An architecture for general intelligence. *Artificial Intelligence* 33(1):1–64.

Laird, J. E. 2012. *The Soar Cognitive Architecture*. Cambridge, MA: MIT Press.

Langley, P., and Choi, D. 2006. A unified cognitive architecture for physical agents. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence*.

Langley, P. 1983. Learning search strategies through discrimination. *International Journal of Man-Machine Studies* 18(6):513–541.

Liu, A.-A.; Su, Y.-T.; Nie, W.-Z.; and Kankanhalli, M. 2017. Hierarchical clustering multi-task learning for joint human action grouping and recognition. *IEEE transactions on pattern analysis and machine intelligence* 39(1):102–114.

Molineaux, M., and Aha, D. W. 2014. Learning unknown event models. In *AAAI*, 395–401.

Nejati, N. 2011. *Analytical Goal-Driven Learning of Procedural Knowledge by Observation*. Ph.D. Dissertation, Stanford University.

Nuxoll, A. M., and Laird, J. E. 2007. Extending cognitive architecture with episodic memory. In *Proceedings of the Twenty-Second National Conference on Artificial Intelligence*, 1560–1565.

Simon, H. A. 1980. Cognitive science: The newest science of the artificial. *Cognitive science* 4(1):33–46.

Tulving, E. 1983. Elements of episodic memory.

Walsh, T. J., and Littman, M. L. 2008. Efficient learning of action schemas and web-service descriptions. In *AAAI*, volume 8, 714–719.

Wang, Z.; Jegelka, S.; Kaelbling, L. P.; and Lozano-Pérez, T. 2017. Focused model-learning and planning for non-gaussian continuous state-action systems. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, 3754–3761. IEEE.

Wang, X. 1994. Learning planning operators by observation and practice. In *AAAI*, 335–340.

Yang, J.; Xu, Y.; and Chen, C. S. 1997. Human action learning via hidden markov model. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans* 27(1):34–44.