Foundations and Applications of Generalized Planning

Siddharth Srivastava
University of Massachusetts, Amherst MA 01003

Generalized planning problems capture planning problems with uncertainties in object quantities and properties. While the generalized planning problem is incomputable in general, this thesis presents a new formalization of the problem, a study of the limits of computability of generalized plans and algorithms for solving a broad class of generalized planning problems.

Keywords: Automated planning, plan generalization, handling unbounded state spaces

1. Introduction

Advances in AI are fueling the development of automated agents ranging in scope from intelligent software to automated vacuum cleaners and self-driving cars. As the number and impact of such agents in our society increases, the need for computing desirable behaviors for them ("automated planning") while ensuring safety is becoming crucial. Interaction with real, dynamic worlds poses many computational problems for automated agents, including the key representational problem of expressing and reasoning about situations with unknown numbers of objects. Situations where we would like to use automated agents rarely have a small or predetermined number of objects. A rover sent to a distant planet cannot know in advance the number of rocks of interest that it can expect to encounter. Similarly, it would be infeasible to preprogram a household robot for assisting the elderly, with the precise number and properties of objects that can be found in an average home.

Although research in automated planning has led to vast improvements in the scalability of planning systems, representing and planning in situations with unknown numbers of objects with uncertain properties remains a largely unexplored frontier of research. Consider the simple problem of sorting an unknown number of objects in a room into their respective boxes (clothes, shoes, etc.). An algorithmic, generalized plan for this problem would consist of a simple loop of actions, which incrementally picks up an object, senses its property and stores it in the appropriate box. Although this appears to be a simple plan, automatically computing such plans with safety or correctness guarantees requires reasoning about cyclic control flows. This has been called a "notorious" problem in automated planning due to its equivalence with the halting problem of Turing machines in general. It is not surprising that very few directions of work address the problem of computing or representing such plans.

2. Contributions

In my thesis, I formulated this problem as a generalized planning problem [1]. I identified the factors that make it intractable in general, and identified a useful subclass of problems where reasoning about cyclic control and therefore, automatically computing generalized plans, is possible. I also developed algorithms for computing generalized plans in this class.

Given a possibly infinite set of initial problem states and "lifted" action specifications independent of the actual object quantities, the generalized planning problem is to compute a control structure that solves as many of the initial states as possible. We used the language of first-order logic with transitive closure (FO(TC)) to express both the class of initial states and actions in this formalization. The initial states of a generalized planning problem can come from *different* state spaces with varying object quantities and properties. In this framework, generalized plans are algorithms expressed in the form of finite state automata with actions labeling the control-nodes and conditions labeling the edges. A generalized plan can also include actions with variables in place of arguments (e.g. $put(obj_x, box_y)$) and choice-actions (e.g. *choose* box_y: $type(box_y)$=$type(obj_x)$) specifying the objects to be chosen for those variables during execution. This

AI Communications 0 () 0
ISSN 0921-7126 / $8.00 © , IOS Press

representation allows us to express solutions for problems with unknown object quantities and properties.

As a foundation for my thesis, I developed a set of evaluation criteria for generalized plans. These criteria can be used to evaluate any approach that addresses the generalized planning problem. For example, to be truly applicable, any generalized plan should come with an efficient *applicability test* that determines whether it will solve a given problem instance. This is because even small generalized plans with cyclic control can have disparate behaviors on different "similar" problems, to the extent of efficiently achieving the goal in one, and traversing almost the entire state space without ever achieving in another. Automatically computing the applicability test for a generalized plan amounts to computing the preconditions under which it will solve a problem. Unfortunately, because of the expressiveness of generalized plans, this problem is equivalent to the halting problem of Turing machines and is incomputable in general.

I addressed this challenge by studying the problem of finding preconditions for reaching a desirable state in an *abacus program*. Abacus programs have a finite set of positive, integer-valued registers and actions that can increment or conditionally decrement these registers. This framework is Turing-equivalent and can therefore express any cyclic control flow, including generalized plans. Further, generalized plans for a broad class of planning problems have natural representations as abacus programs. Although it is impossible in general to compute the preconditions for reaching a desirable state in an abacus program, I developed methods for computing them in a useful subclass. In this class, our methods can be used to compute the applicability conditions for a generalized plan, under which it will achieve the goal in a finite number of steps without visiting undesirable states.

Equipped with these theoretical results, I developed algorithms for computing generalized plans with efficient applicability tests for a class of planning problems. These algorithms use the concept of *canonical abstraction* from software model checking as a foundation. Given a concrete state, its canonical abstraction represents all objects that have the same properties with respect to a set of *abstraction predicates* by a *summary* element. Adapting this system to planning representations and using all unary predicates in the domain as abstraction predicates gives us an efficient, finite representation for states with uncertainty in object quantities and properties.

Plan Generalization I used canonical abstraction in an algorithm for plan generalization. This algorithm first applies an input concrete plan on the canonical abstraction of an initial state that the plan solves. This yields a sequence of abstract states where recurring state properties can be easily identified in the form of subsuming states. When one abstract state in the sequence subsumes another, the actions between these two states can be placed in a loop (executability of the loop is guaranteed by properties of the abstraction). However, most such loops will repeatedly iterate over the same concrete states and will never terminate during execution. Therefore, before creating the loop we test whether it will make progress and terminate in every possible execution, using the abacus program approach discussed above. In a broad class of domains, this approach produces generalized plans with loops together with efficient applicability tests. Typically, the output generalized plans solve many more problems than the original concrete plan, which solved only one.

Hybrid Plan Synthesis I also developed an algorithm for "bootstrapping" the generalization process and incrementally growing partial generalized plans. In this algorithm, a classical planner is used to create a concrete plan for an unsolved problem instance. At the beginning, this instance can be any member of the class of initial states. At any point in the overall algorithm, a super-set of problem instances that may not be solved by the algorithm is maintained. The incremental process works by selecting a small instance from this class, invoking a classical planner on it, and generalizing and merging the resulting plan with the existing generalized plan. In this way the algorithm quickly builds an initial generalized plan and then continues to add to its coverage by incorporating automatically obtained, useful plan segments.

These approaches were able to solve a wide range of generalized planning problems. We also used them to solve algorithm-synthesis problems like reversing a singly linked list and sorting a list of elements. By using state-of-the-art planners, we were able to utilize our theoretical results and representations to develop a system for performing goal-directed heuristic search for generalized plans.

Acknowledgements This thesis was supervised by Neil Immerman and Shlomo Zilberstein and was supported in part by NSF grant IIS-0915071.

References

[1] Siddharth Srivastava. *Foundations and Applications of Generalized Planning*. PhD Dissertation, Department of Computer Science, University of Massachusetts Amherst, 2010. http://www.cs.umass.edu/~siddhart/Publications/thesis.pdf