Qualitative Numeric Planning

Siddharth Srivastava and **Shlomo Zilberstein** and **Neil Immerman**
Department of Computer Science
University of Massachusetts, Amherst, MA 01003
{siddharth, shlomo, immerman}@cs.umass.edu

Hector Geffner
ICREA & Universitat Pompeu Fabra
Barcelona, SPAIN
hector.geffner@upf.edu

Abstract

We consider a new class of planning problems involving a set of non-negative real variables, and a set of non-deterministic actions that increase or decrease the values of these variables by some arbitrary amount. The formulas specifying the initial state, goal state, or action preconditions can only assert whether certain variables are equal to zero or not. Assuming that the state of the variables is fully observable, we obtain two results. First, the solution to the problem can be expressed as a *policy* mapping *qualitative states* into actions, where a qualitative state includes a Boolean variable for each original variable, indicating whether its value is zero or not. Second, testing whether any such policy, that may express nested loops of actions, is a solution to the problem, can be determined in time that is polynomial in the qualitative state space, which is much smaller than the original infinite state space. We also report experimental results using a simple generate-and-test planner to illustrate these findings.

1. Introduction

The problem of planning with loops (Levesque 2005), has received increasing attention in recent years (Srivastava et al. 2008; Bonet et al. 2009; Hu and Levesque 2010; Srivastava et al. 2010), as situations where actions or plans have to be repeated until a certain condition is achieved are quite common. For example, the plan for chopping a tree discussed by Levesque involves getting an axe, moving to the tree, and chopping the tree until it falls down. The plan thus includes the action chop, which must be repeated a finite, but unknown number of times. In this work, we aim to understand the conditions under which plans with loops of various forms may be required, and likewise, the conditions under which execution can be guaranteed to lead to the goal in finitely many steps.

We consider a new class of planning problems involving a set of non-negative real variables $x_1 \ldots, x_n$, and a set of actions that increase or decrease the values of these variables by an arbitrary *random* positive amount. These effects, denoted as $inc(x_i)$ (resp. $dec(x_i)$), change the value of x_i to $x_i + \delta$ (resp. $x_i - \delta$), where δ is some arbitrary positive value that could vary over different instantiations of an action. The only restrictions are that (1) decreases never make a variable negative, i.e., δ cannot be greater than x_i when x_i is decreased, and (2) a sequence of decreases will eventually make any variable x equal to 0.

For the chopping tree example, for instance, we may assume that the action "chop" decreases the probability of the tree not falling by some random amount δ each time, so that if this probability x is initially $x = 1$, it will be $x = 1-\delta \geq 0$ after a single "chop" action, and it will be $x = 0$ ("tree falls down") after a finite sequence of actions whose length cannot be predicted. Other examples could involve refueling a vehicle, shoveling snow, or loading a truck. Our goal is to develop a *general* framework for planning with actions that could affect multiple variables simultaneously.

Plans in such non-deterministic settings naturally require loops, and in many cases, *nested* or *complex loops*. For example, suppose that initially $x = 10$ and $y = 5$, and the goal is $x = 0$. The available actions are a with precondition $y = 0$ and effects $dec(x)$ and $inc(y)$, and b with no preconditions and effect $dec(y)$. A plan for this problem involves a loop in which the action b is repeated (in a nested loop) until $y = 0$ and then the action a is performed. The main loop repeats until $x = 0$.

We consider the simplest setting where the need for such loops arises, restricting the literals in action preconditions, initial and goal states to be of the form $x = 0$ or $x \neq 0$. The effects of all the actions are of the form $inc(x)$ (random increases) or $dec(x)$ (random decreases). For the sake of simplicity, the representation does not include Boolean propositions, but such an extension is straightforward.

We assume initially that the states (variable values) are *fully observable*. One of the key results is that, given the nature of the literals in action preconditions, initial and goal states, full observability provides no more useful information than *partial observability*, which only indicates whether each variable is equal to zero or not. We refer to such abstract states as *qualitative states*, and show that solutions to planning problems considered in this paper can be expressed as *functions* or *policies* that map *qualitative states* into actions. For example, the above problem with a nested loop,

$$\textbf{repeat}\ \{\textbf{repeat}\ \{b\}\ \textbf{until}\ (y = 0);\ a;\}\ \textbf{until}\ (x = 0)$$

corresponds to the following qualitative policy π:

$$\pi(qs) = a \text{ if } y = 0 \text{ in } qs\ ;\ \pi(qs) = b \text{ if } x \neq 0 \text{ and } y \neq 0$$

Copyright © 2011, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

where qs ranges over qualitative states.

Furthermore, we develop a sound and complete algorithm for testing whether a plan with loops expressed by a policy π solves the problem; meaning, that it terminates in a goal state in a finite number of steps. This algorithms runs in time that is polynomial in the size of the qualitative space, which is exponential in the number of variables, but still much smaller than the original state space, which is infinite. Experiments with a simple generate-and-test planner are reported, illustrating the potential utility of these results.

2. The Planning Problem

We begin by defining the fully-observable non-deterministic *quantitative* planning problem, or simply the planning problem to be discussed in this paper. Throughout the paper, we will make the assumption that whenever an action (qualitative or quantitative) has a decrease effect on x, the action preconditions include $x \neq 0$. Given a set of variables X, let $\mathcal{L}_X$ denote the class of all consistent sets of literals of the form $x = 0$ and $x \neq 0$, for $x \in X$.

Definition 1. *A quantitative planning problem $P = \langle X, I, G, O\rangle$ consists of X, a set of non-negative numeric variables; I, a set of initially true literals of the form $x = c$, where c is a non-negative real, for each $x \in X$; $G \in \mathcal{L}_X$, a set of goal literals; and O, a set of action operators. Every $a_i \in O$ has a set of preconditions $pre(a_i) \in \mathcal{L}_X$, and a set* eff$(a_i)$ *of effects of the form* inc(x) *or* dec(x) *for $x \in X$.*

This formulation is related to the general numeric planning problem (Helmert 2002), but includes non-deterministic actions and uses limited forms of preconditions and goals.

A setting of the variables in X constitutes a state of the quantitative planning problem. Unless otherwise specified, a "state" in this paper refers to such a state. When needed for clarity, we will refer to these states as *quantitative states*. Note that these states are fully obvervable. We use s_I to represent the initial state of the problem, corresponding to the assignment I. Solutions to quantitative planning problems can be expressed in the form of *policies*, or functions from the set of states of P to the set of actions in O. Without loss of generality, we will assume that all policies discussed in this paper are partial. In particular, states satisfying the goal condition are not mapped to any action.

A sequence of states, $s_0, s_1, \ldots$ is a *trajectory given a policy* π iff $s_{i+1} \in a_i(s_i)$, where $a_i = \pi(s_i)$. Intuitively, a policy π solves P if every trajectory given π beginning with s_I leads to a state satisfying the goal condition in a finite number of steps. To make this definition formal, we first define ϵ-bounded transitions and trajectories:

Definition 2. *The pair (s_1, s_2) is an ϵ-*bounded transition *if there exists $a_i \in O$ such that for every variable x, if* inc(x) $\in$ eff(a_i)*, then the value of x in s_2 (denoted $x(s_2)$) is $x(s_1) + \delta$, $\delta \in [\epsilon, \infty)$ and for every variable x such that* dec(x) $\in$ eff(a_i)*, $x(s_2) = x(s_1) - \delta$, where $\delta \in [\min(\epsilon, x), x]$.*

*An ϵ-*bounded trajectory *is one in which every consecutive pair of states is an ϵ-bounded transition.*

In ϵ-bounded trajectories, no variable can be decreased an infinite number of times without also being increased an infinite number of times. Recall that a variable being positive is a precondition of every action that decreases the variable. We can now define when a policy solves a given quantitative planning problem:

Definition 3. *A policy π* solves *$P = \langle X, I, G, O\rangle$ if for any $\epsilon > 0$, every ϵ-bounded trajectory given π starting with s_I is finite and ends in a state satisfying G.*

While each ϵ-bounded trajectory of a solution policy must be of some finite length, the policy itself must cover an infinite set of reachable states. Consequently, explicit representations of such policies in the form of state-action mappings will be infinite, making explicit representations infeasible, and the search for solutions difficult. In order to deal with this problem, we use a particularly succinct policy representation that abstracts out most of the information present in a quantitative state, and yet is sufficient to solve the problem.

2.1 Qualitative Formulation

We consider an abstraction of the quantitative planning problem defined above, where *qualitative* states (*qstates*) are elements of $\mathcal{L}_X$, for a set of variables X. We first define the effect of qualitative increase and decrease operations on qstates, and then show their correspondence with the original planning problem.

The qualitative version of $inc(x)$ always results in a state where $x \neq 0$. The qualitative version of $dec(x)$, when applied to a qstate where $x \neq 0$ results in two qstates, one with $x \neq 0$ and the other with $x = 0$. For notational convenience we will use the same terms (*inc* and *dec*) to represent qualitative effects and clarify the usage when it is not clear from the context. The outcome of a qualitative action with $dec()$ effects on k different non-zero variables includes 2^k qstates representing all possible combinations of each qualitative effect. While large, this is better than the infinite set of quantitative states that could arise from a single action that adds or subtracts any possible δ.

Definition 4. *A qualitative planning problem $\tilde{P} = \langle \tilde{X}, \tilde{I}, G, \tilde{O}\rangle$ for a set of variables X, consists of $\tilde{X}$, the set of literals $x = 0$ and $x \neq 0$ for $x \in X$; $\tilde{I} \in \mathcal{L}_X$, a set of initially true literals from $\tilde{X}$; $G \in \mathcal{L}_X$ the set of goal literals; and $\tilde{O}$, a set of qualitative action operators. Every $a_i \in \tilde{O}$ has a set of preconditions $pre(a_i) \in \mathcal{L}_X$ and a set* eff(a_i) *of qualitative effects of the form $inc(x)$ or $dec(x)$, for $x \in X$.*

Any quantitative planning problem P can be *abstracted* into a qualitative planning problem by replacing each assignment in I with the literal of the form $x = 0$ or $x \neq 0$ that it satisfies, and replacing inc and dec effects in actions with their qualitative counterparts. Similarly, every state s over a set of variables X corresponds to the unique qstate $\tilde{s}$ obtained by replacing each variable assignment $x = c$ in s with $x = 0$ if $c = 0$ and $x \neq 0$ otherwise. We denote the set of states corresponding to a qstate $\tilde{s}$ as $\gamma(\tilde{s})$.

Example 1. *For the example used in the introduction, $X = \{x, y\}$; $I = \{x = 10, y = 5\}$; $G = \{x = 0\}$; $O = \{a, b\}$, where $a = \langle$pre:$\{y = 0\}$; eff:$(dec(x), inc(y))\rangle$ and $b = \langle$pre:$\{\}$; eff:$(dec(y))\rangle$.*

The qualitative version of this problem defined over X has the same G, with $\tilde{X} = \{x = 0, x \neq 0, y = 0, y \neq 0\}$;$\tilde{I} = \{x \neq 0, y \neq 0\}$ and $\tilde{O}$ has the same operators as O but uses the qualitative forms of inc *and* dec *effects.*

The qstate $\tilde{s_I} = \{x \neq 0, y \neq 0\}$. $\gamma(\tilde{s_I})$ includes s_I, and infinitely many additional states where x and y are non-zero.

Notation In the rest of this paper, we will refer to the quantitative planning problems that can be abstracted into a qualitative planning problem $\tilde{P}$ as the *quantitative instances* of $\tilde{P}$. We will use the notation $\tilde{a}$ and $\tilde{s}$ to denote the qualitative versions of an action a and state s, respectively. Since the class of states represented by any qstate is always non-empty, any qstate can be represented as $\tilde{s}$ for some state s.

The following result establishes a close relationship between the quantitative and qualitative formulations:

Theorem 1. (Soundness and completeness of qualitative action application) $\tilde{s_2} \in \tilde{a}(\tilde{s_1})$ *iff there exists* $t \in \gamma(\tilde{s_2})$ *such that* $t \in a(s_1)$.

Proof. If $t \in a(s_1)$, then it is easy to show the desired consequence (soundness) because the qualitative inc and dec operators capture all possible qualitative states resulting from inc and dec.

In the other direction, if $\tilde{s_2} \in \tilde{a}(\tilde{s_1})$, the desired element t can be constructed as follows. If a and $\tilde{a}$ decrease a variable x, then we must have $x \neq 0$ in s_1 and either $x \neq 0$ or $x = 0$ in $\tilde{s_2}$. In either case, we can use a suitable δ for the decrease to arrive at a value of x satisfying the condition. If a increases a variable, then we can use any δ for $inc(x)$, because $\tilde{s_2}$ must have $x \neq 0$. In this way we can choose δ's for each inc/dec operation in a to arrive at a setting of the variables corresponding for $t \in a(s_1) \cap \gamma(\tilde{s_2})$. □

2.2 Qualitative Policies

A qualitative policy for a qualitative problem $\tilde{P}$ is a mapping from the qstates of $\tilde{P}$ to its actions. A terminating qualitative policy is one whose *instantiated*, ϵ-bounded trajectories are finite. Formally,

Definition 5. *Let $\tilde{\pi}$ be a qualitative policy. An* instantiated trajectory *of $\tilde{\pi}$* started at qstate *$\tilde{t}$ is a sequence of* quantitative *states $s_0, s_1, \ldots$, such that $s_0 \in \gamma(\tilde{t})$ and $s_{i+1} \in a_i(s_i)$, where a_i is the quantitative action corresponding to the action $\tilde{a_i} = \tilde{\pi}(\tilde{s_i})$.*

Definition 6. *A qualitative policy is said to* terminate *when started at $\tilde{s_i}$ iff all its instantiated, ϵ-bounded trajectories started at $\tilde{s_i}$ are finite.*

A qualitative policy $\tilde{\pi}$ is said to **solve** *a qualitative planning problem $Q = \langle X, \tilde{I}, G, \tilde{O}\rangle$ iff: (1) $\tilde{\pi}$ terminates when started at the initial state $\tilde{s_I}$ and (2) every instantiated ϵ-bounded trajectory of $\tilde{\pi}$ started at $\tilde{s_I}$ ends at a state satisfying the goal condition G.*

If $\tilde{\pi}$ is a qualitative policy for a qualitative planning problem, $\tilde{P}$, then $\tilde{\pi}$ defines a corresponding policy π for any quantitative instance P of $\tilde{P}$: $\pi(s)$ is the quantitative version of the action $\tilde{\pi}(\tilde{s})$. However, the converse is not true: a quantitative policy may map different states represented by the *same* qstate to different actions. In such cases, it is not clear which action this qstate must be mapped to under a qualitative policy. To distinguish the class of quantitative policies that do yield a natural qualitative policy, we use the following notions:

Definition 7. *Let $P = \langle X, I, G, O\rangle$ be a quantitative planning problem. Two states s_1, s_2 of P are* qualitatively similar *iff $\forall x \in X, (x\!=\!0\ in\ s_1) \Leftrightarrow (x\!=\!0\ in\ s_2)$. Otherwise, they are* qualitatively different.

A policy π for P is essentially qualitative *iff $\pi(s) = \pi(s')$ for every pair of qualitatively similar states s, s'.*

Note that qualitative similarity is an equivalence relation over the set of quantitative states. Essentially qualitative policies can be naturally translated into qualitative policies. In order to show that any problem P must have a qualitative solution if it has a quantitative solution, we first define a method for translating a state trajectory into the qstate space:

Definition 8. *An* abstracted trajectory *given a quantitative policy π starting at state s_0 is a sequence of qstates $\tilde{s_0}, \tilde{s_1}, \ldots$, corresponding to an ϵ-bounded trajectory given π, $s_0, s_1, \ldots$, where each s_i is qualitatively different from s_{i+1}.*

We now present the main result of this section. Proofs of the lemmas used to prove this result are presented in Appendix A.

Theorem 2. *If P has a solution policy then it also has a solution policy that is essentially qualitative.*

Proof. Suppose the conclusion is not true, and every solution policy for P requires that two qualitatively similar states s_1 and s_2 are mapped to different actions. Let π be such a solution policy with $\pi(s_1) = a_1$ and $\pi(s_2) = a_2$. By our assumption, $a_1(s_2)$ must be an unsolvable state.

Let π' be a policy such that π' is the same as π, except that $\pi'(s_2) = a_1$. Let the set of abstracted trajectories for π and s_1 be AT_1, and AT_2, those for π' and s_2. By Lemma 1 (see Appendix A for proofs of lemmas), since s_1 and s_2 are qualitatively similar, we know that π and π' have the same set of abstracted trajectories. As π solves P, by Lemma 2 π' also solves P. Thus, s_1 and s_2 *can* be mapped to the same action and we have a contradiction. □

This result gives us an effective method to search for solutions to the original planning problem whose policies need to map an entire space of real-valued vectors to actions: we only need to consider policies over the space of qualitative (essentially boolean) states. Further, we only need to solve the qualitative version of a problem to find a solution policy: a qualitative policy $\tilde{\pi}$ solves $\tilde{P}$ iff the natural quantitative translation π of $\tilde{\pi}$ solves P (Cor. 1, Appendix A).

3. Identifying Qualitative Solution Policies

We now present a set of results and algorithms for determining when a qualitative policy solves a given qualitative problem. These methods abstract away the need for testing all ϵ-bounded trajectories. Since Theorem 2 shows that the only policies we need to consider are qualitative policies, in the rest of this paper "policies" refer to qualitative policies, unless otherwise noted. We use the following observation as the central idea behind the methods in this section:

Fact 1. *In any ϵ-bounded trajectory, no variable can be decreased infinitely often without an intermediate increase.*

Let $ts(\pi, \tilde{s_I})$ be the transition system induced by a policy π with $\tilde{s_I}$ as the initial qstate. Thus, $ts(\pi, \tilde{s_I})$ includes all qstates that can be reached from $\tilde{s_I}$ using π. In order to ensure that π solves $\tilde{P}$, we check $ts(\pi, \tilde{s_I})$ for two independent criteria: (a) if an execution of π terminates, it terminates in a goal state; and (b) that every execution of π terminates. We call a state in a transition system terminal if there is no outgoing transition from that state. Test (a) above is accomplished simply by checking for the following condition:

Definition 9. (Goal-closed policy) *A policy π for a qualitative planning problem $\tilde{P} = \langle X, \tilde{I}, G, \tilde{O} \rangle$ is goal-closed if the only terminal states in $ts(\pi, \tilde{s_I})$ are goal states.*

If a policy π solves a qualitative planning problem $\tilde{P}$ then it must be goal-closed (Lemma 3, Appendix A). Testing that a policy is goal-closed takes time linear in the number of states in $ts(\pi, \tilde{s_I})$. It is easy to see that if a policy terminates then it must do so at a terminal node in $ts(\pi, \tilde{s_I})$. Consequently, if we have a policy π that is goal-closed and can also be shown to terminate (as per Def. 6), then π must be a solution policy for the given qualitative planning problem.

Terminating qualitative policies are a special form of strong cyclic policies (Cimatti et al. 2003). A strong cyclic policy π for a qualitative problem $\tilde{P}$ is a function mapping qstates into actions such that if a state s is reachable by following π from the initial state, then the goal is also reachable from s by following π. The execution of a strong cyclic policy does not guarantee that the goal will be eventually reached, however, as the policy may cycle. Indeed, the policies that solve $\tilde{P}$ can be shown to be terminating strong cyclic policies for $\tilde{P}$:

Theorem 3. *π is a policy that solves $\tilde{P}$ iff π is a strong cyclic policy for $\tilde{P}$ that terminates.*

Proof. Suppose π is a policy that solves $\tilde{P}$ and has a cyclic transition graph. By Def. 6, π must terminate at a goal state. Suppose s is a qstate reachable from s_I under π. If there is no path from s to the goal, then no instantiated trajectory from s will end at a goal state, contradicting the second condition in Def. 6. Thus, π must be strong cyclic. The converse is straightforward. □

The set of terminating qualitative policies is a strict subclass of the set of strong cyclic policies, which in turn is a strict subclass of goal-closed policies. If we can determine whether or not a policy terminates, then we have two methods for determining if it solves a given problem: by proving either that the policy is goal-closed or that it is strong cyclic.

3.1 Identifying Terminating Qualitative Policies

We now address the problem of determining when a qualitative policy for a given problem will terminate. Execution of a policy can be naturally viewed as a flow of control through the transition graph of the policy. Therefore, to determine if π terminates, we check if every strongly connected component (SCC) in $ts(\pi, \tilde{s_I})$ will be exited after a finite number of steps in any execution.

Algorithm 1: Sieve Algorithm

Input: $g = ts(\pi, \tilde{s_I})$
1 **if** *g is not strongly connected* **then**
2   Goto Step 11
end
3 **repeat**
4   Pick an edge e of g that decreases a variable that no edge in g increases
5   Remove e from g
until *Step 4 finds no such element*
6 **if** *g is acyclic* **then**
7   Return Terminating
end
8 **if** *no edge was removed* **then**
9   Return "Non-terminating"
end
10 **if** *edge(s) were removed* **then**
11   **for** *each* $g' \in$ SCCs-of(g) **do**
12   **if** *Sieve(g')= "Non-terminating"* **then**
13   Return "Non-terminating"
        end
    end
14   Return "Terminating"
end

The Sieve algorithm (Alg. 1) is a sound and complete method for performing this test. The main sections of the algorithm apply on the SCCs of the input g. Lines 1 & 2 (see Alg. 1) redirect the algorithm to be run on each strongly connected component in the input (lines 11-13). The main algorithm successively identifies and removes edges in an SCC that decrease a variable that no edge increases. Once all such edges are removed in lines 4 & 5, the algorithm's behavior depends on whether or not the resulting graph is cyclic. If it isn't, then the algorithm identifies this graph as "Terminating" (line 5). Otherwise, its behavior depends on whether or not edges were removed in lines (4 & 5). If they weren't, then this graph is identified to be non-terminating. If edges were removed, the algorithm recurses on each SCC in the graph produced after edge removal (10-13). The result "Non-terminating" is returned iff any of these SCCs is found to be non-terminating (line 13).

Theorem 4. (Soundness of the Sieve Algorithm) *If the sieve algorithm returns "Terminating" for a transition system $g = ts(\pi, \tilde{s_I})$, then π terminates when started at $\tilde{s_I}$.*

Proof. By induction on the recursion depth. For the base case, consider an invocation of the sieve algorithm that returns "Terminating" through line 6 after removing some edges in lines 4-5. Fact 1 implies that in any execution of g, the removed edges cannot be executed infinitely often because they decrease a variable that no edge in the SCC increases. The removal of these edge corresponds to the fact that no path in this SCC using these edges can be executed infinitely often. Since g becomes acyclic on the removal of these edges, after finitely many iterations of the removed edges flow of control must take an edge that leaves g.

For the inductive case, suppose the algorithm works for recursive depths $\leq k$ and returns "Terminating" when called on an SCC that requires a recursion of depth $k+1$. The argument follows the argument above, except that after the removal of edges in lines 4-5, the graph is not acyclic and Sieve algorithm is called on each resulting SCC. In this case, by arguments similar to those above, we can get a non-terminating execution only if there is a non-terminating execution in one of the SCCs in the reduced graph ($g' \in$ *SCCs-of(g)*), because once the flow of control leaves an SCC in the reduced graph it can never return to it, by definition of SCCs. We only need to consider the reduced graph here because edges that are absent in this reduced graph can only be executed finitely many times.

However, the algorithm returned "Terminating", so each of the calls in line 12 must have returned the same result. This implies, by the inductive hypothesis, that execution cannot enter a non-terminating loop in any of the SCCs g', and therefore, must terminate. □

Theorem 5. (Completeness of the sieve algorithm) *Suppose the sieve algorithm returns "Non-terminating" for a certain transition system* $g = \text{ts}(\pi, \tilde{s_I})$. *Then* π *is non-terminating when started at* $\tilde{s_I}$.

Proof. Suppose the sieve algorithm returns "Non-terminating". Consider the deepest level of recursion where this result was generated in line 9. We show that the SCC g_0 for which this result was returned actually permits a non-terminating ϵ-bounded trajectory given π, for any ϵ. Let $\tilde{s_0}$ be a state in g_0 through which g_0 can be entered. Let the total number of edges in g_0 that decrease any variable be D, and let the number variables that undergo a decrease be V.

Consider a state $q_0 \in \gamma(\tilde{s_0})$ where every variable that is non-zero in $\tilde{s_0}$ is assigned a value of at least $3DV\epsilon$. For every variable x_i that is decreased, let $(\tilde{s_i}, \tilde{t_i})$ be an edge that increases it. Let p_i be the smallest path $\tilde{s_0} \rightarrow^* \tilde{s_i} \rightarrow \tilde{t_i} \rightarrow^* \tilde{s_0}$ in the SCC g_0. We can then construct a non-terminating execution sequence as follows: for each variable x_i being decreased, follow the path p_i exactly once in succession, using the following δs starting with s_0: the δ for every inc operation is at least $3DV\epsilon$; the δ for each $dec(x)$ operation is either (a) ϵ, if the result state in this path following this operation has $x \neq 0$, or, (b) x, if the result state has $x = 0$.

The execution of any path causes a maximum of $2D$ decs on any variable that is not zero in $\tilde{s_0}$, and this occurs only if the variable was never set to zero, in which case it will be increased again in the path leading to $\tilde{s_0}$, resulting in a higher value. Also, after this round of executing V paths, each non-zero variable must be at least $3DV\epsilon - 2DV\epsilon + 3DV\epsilon$. Thus the final value of every variable $x \neq 0$ in $\tilde{s_0}$ increases after one iteration of this execution sequence, and this sequence can be repeated ad infinitum. The main result follows, because we can construct an $s_i \in \gamma(\tilde{s_I})$ that leads to a value of at least $3DV\epsilon$ for every non-zero variable in $\tilde{s_0}$ along a linear path from $\tilde{s_I}$ to $\tilde{s_0}$. (This application may result in a value greater than $3DV\epsilon$, if ϵ can take on only positive integral values.) □

Note that these results continue to hold in settings where the domains of any of the variables and action effects on those variables are positive integers, under the natural assumption that ϵ, or the minimum change caused by an action on those variables, must be at least 1. The results follow, since Theorem 1 and Fact 1 continue to hold.

4. Experiments

We implemented the algorithms presented above using a simple generate and test planner that is sound and complete: it finds a solution policy for a given problem (which may or may not be qualitative) iff there exists one. The purpose of this implementation is to illustrate the scope and applicability of the proposed methods. The experiments show that these methods can produce compact plans that can handle complex situations that are beyond the scope of existing planners. The following section presents some directions for building an even more scalable planner based on these ideas.

The planner works by incrementally generating policies until one is found that is goal-closed and passes the Sieve algorithm test for termination. The set of possible policies is constructed so that only the actions applicable in a state can be assigned to it, and the test phase proceeds by first checking if the goal state is present in the policy's transition graph. Tarjan's algorithm (Tarjan 1972) is used to find the set of strongly connected components of a graph.

Problems. In addition to the tree-chop and nested loop examples discussed in the introduction, we applied the planner on some new problems that require reasoning about loops of actions that have non-deterministic numeric effects. In the Snow problem, a snow storm has left some amounts of snow, *sd* and *sw*, in a driveway and the walkway leading to it, respectively. The actions are *shovel()*, which decreases *sw*; *mvDW()*, with precondition *sw=0*, which moves a snowblower to the driveway by decreasing the distance to driveway (*dtDW*); *snow-blower()*, with precondition *dtDW*$=0$, which decreases *sd* but also spills snow on the walkway, increasing *sw*. The goal is *sd*=*sw*=0.

In the delivery problem, a truck of a certain capacity *tc* needs to deliver *objs* objects from a source to a destination. The *load()* action increases the objects in truck, *objInT*, while decreasing *tc*; *mvD()* decreases the truck's distance to destination, *dtD()*, while increasing its distance from the source, *dtS()*; *mvS()* does the inverse. Both *mv()* actions also decrease the amount of *fuel*, which is increased by *getFuel()*. Finally, *unload()* with precondition *dtd* $=0$ decreases *objs, objInT* while increasing *tc*. The goal is *objs* $=0$. We also applied the planner on the trash-collecting problem (Bonet et al. 2009) in which a robot must reach the nearest trash object, collect it if its container has space, move to the trash can and empty its container. This is repeated until no trash remains in the room.

Summary of Solutions. Solutions for each of these problems were found in less than a minute on a 1.6GHz Intel Core2 Duo PC with 1.5GB of RAM (Table 1). Almost all the solutions have nested loops in which multiple variables

Tree	NestedVar	Snow	Delivery+Fuel	Trash-Collection
0	0.01	0.02	0.7	30

Table 1: Time taken (sec) for computing a solution policy

are increased and decreased, which makes existing methods for determining correctness infeasible. However our planner guarantees that the computed policies will reach the goal in finitely many steps. Achieving these results is beyond the scope of any existing planning approach. Most of the solutions are too large to be presented, but a schematic representation of one policy for Trash-collecting and two for the Snow problem are presented in Fig. 1. Note that the states corresponding to the two *shovel* actions after *snow-blower* in the policy in the center are distinct: the shovel action in the nested loop is for the state with snow in both the drivewway and the walkway ($sd \neq 0, sw \neq 0$). The nested loop must terminate because *sd* is never increased. On the other hand the shovel action in the non-nested loop is for $sd = 0, sw \neq 0$. If allowed to continue, the planner also finds the policy on the right in under a second.

As expected however, performance of the "generate" part of this implementation can deteriorate rapidly as the number of variables increases. Although problems of the kind described above can be solved easily, in randomly constructed problems and problems with larger variable or action sets, the size of the policy space can be intractable as it is more likely to approach the true bound of $|O|^{2^{|X|}}$.

5. Related Work

The recent revival of interest in planning with loops has highlighted the lack of sufficiently good methods for determining when a plan or a policy with complex loops will work. Existing analysis however has focused on Turing-complete frameworks where proving plan termination is undecidable in general and can only be done for plans with restricted classes of loops (Srivastava et al. 2010) and on problems with single numeric variables (Hu and Levesque 2010). The qualitative framework presented in this paper is very close to the framework of abacus programs studied by Srivastava et al. (2010). If action effects are changed from *inc/dec* to increments and decrements by 1, qualitative policies can be used to represent arbitrary abacus programs. Consequently, the problems of determining termination and reachability of goal states in qualitative policies using increment/decrement actions are undecidable in general. The DISTILL system (Winner and Veloso 2007) produces plans with limited kinds of loops; controller synthesis (Bonet et al. 2009) computes plans or controllers with with loops that are general in structure, but the scope of their applicability and the conditions under which they will terminate or lead to the goal are not determined.

Our work is also related to the framework of strong cyclic policies. Qualitative policies that terminate form a meaningful subclass of strong cyclic policies: namely, those that cannot decrease variables infinitely often without increasing them. Our definition of termination is orthogonal to the definition of strong cyclic policies and exploits the semantics

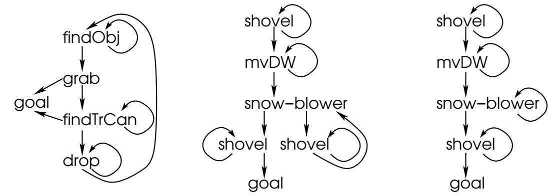

Figure 1: Schematic representations of solution policies for trash-collection (left) and snow problems (center & right).

of numeric variables and actions on those variables in the original quantitative planning problem. In particular, an action a that decreases a variable x is not only mapped to a qualitative action that non-deterministically maps the literal $x \neq 0$ into either $x = 0$ or $x \neq 0$, but is also constrained by the semantics such that the first transition, from $x \neq 0$ to $x \neq 0$, is possible only finitely many times unless x is increased infinitely often by another action.

On the other hand, actions in strong cyclic policies have non-deterministic, *stationary* effects in the sense that all action outcomes are equally possible, regardless of the number of times that an action is applied. Consider a blocks world scenario where the goal is to unstack a tower of blocks and a *tap* action non-deterministically topples the entire tower (assume that the stability of the tower is not disturbed when it is not toppled). Repeatedly applying *tap* is a strong cyclic plan, and also the only solution to this problem, but it is not terminating in the context of this paper. If the domain also includes an action to move the topmost block in a stack to the table, a qualitative policy that maps the qstates with non-zero heights to this unstacking action is terminating because it always decreases the height of the tower. In a domain with both actions, a strong cyclic planner will return a policy with either the tap or the unstacking action, while our approach will select the policy with the unstacking action.

6. Conclusions and Future Work

We presented a sound and complete method for solving a new class of numeric planning problems where actions have non-deterministic numeric effects and numeric variables are partially observable. Our solutions represent plans with loops and are guaranteed to terminate in a finite number of steps. Although other approaches have addressed planning with loops, this is the first framework that permits complete methods for determining termination of plans with any number of numeric variables and any class of loops. We also showed how these methods can be employed during plan generation and validated our theoretical results with a simple generate and test planner. The resulting planning framework is expressive enough to capture many interesting problems that cannot be solved by existing approaches.

Although we only explored applications in planning, these methods can also be applied to problems in qualitative simulation and reasoning (Travé-Massuyès et al. 2004; Kuipers 1994). Our approach can also be extended easily to problems that include numeric as well as boolean variables. These methods can be used to incorporate bounded counting. Scalability of our planner could be improved substantially by using a strong cyclic planning algorithm to enumerate potential solution policies (the testing phase would only require a termination check). A promising direction for future work is to capture situations where variables can be compared to a finite set of landmarks in operator preconditions and goals. Another direction for research is to represent finitely many scales of increase and decrease effects e.g. to represent scenarios such as a robot that needs to make large scale movements when it is far from its destination and finer adjustments as it approaches the goal.

Acknowledgments

We would like to thank the anonymous reviewers for their diligent reviews and helpful comments. Support for the first three authors was provided in part by the National Science Foundation under grants IIS-0915071, CCF-0541018, and CCF-0830174. H. Geffner is partially supported by grants TIN2009-10232, MICINN, Spain, and EC-7PM-SpaceBook.

A Proofs of Auxiliary Results

We present some results to show that abstracted trajectories capture the necessary features of solution policies for the original quantitative planning problem.

Lemma 1. *Any two qualitatively similar states have the same set of abstracted trajectories.*

Proof. If two states s_1 and t_1 are qualitatively similar, then $\tilde{a}(\tilde{s_1}) = \tilde{a}(\tilde{t_1})$. Then, Theorem 1 implies that the possible results $s_2 \in a(s_1)$ and $t_2 \in a(t_2)$ must also come from the same set of qualitative states. The lemma follows by an inductive application of this fact over the length of each abstracted trajectory. □

Lemma 2. *A policy π solves a quantitative planning problem $P = \langle X, I, G, 0\rangle$ iff every abstracted trajectory for π starting at s_I is finite, and ends at a state satisfying the goal condition.*

Proof. The forward direction is straightforward, since if π has an infinite abstracted trajectory, it must correspond to an infinite ϵ-bounded trajectory for π.

In the other direction, suppose all abstracted trajectories for π are finite. The only way an ϵ-bounded trajectory can be non-terminating is if it remains in a single qstate for an unbounded number of steps. In other words, the qstate where this happens must be such that $\tilde{s} \in \tilde{a}(\tilde{s})$, where a is the action that is being repeated. If there is no other state in $\tilde{a}(\tilde{s})$, and $\tilde{s}$ is not a goal state, then this abstracted trajectory, and all the ϵ-bounded trajectories corresponding to it can only end at the non-goal state $\tilde{s}$. This is given to be false. Thus, there must be another qstate $\tilde{s_1} \in \tilde{a}(\tilde{s})$. This implies that $\tilde{a}$ must have a qualitative dec operation, because this is the only operation in the qualitative formulation that leads to multiple results. Finally, if there is a qualitative $dec(x)$ operation in $\tilde{a}$, then any ϵ-bounded trajectory can only execute a and stay in $\tilde{s}$ a finite number of times before it makes x zero. This shows that any ϵ-bounded instantiated trajectory of this trajectory must terminate after a finite number of steps. □

Corollary 1. *Let P be a quantitative planning problem and $\tilde{P}$ its qualitative abstraction. A qualitative policy $\tilde{\pi}$ solves $\tilde{P}$ iff the natural quantitative translation π of $\tilde{\pi}$ solves P.*

Proof. If $\tilde{\pi}$ is a solution to $\tilde{P}$, then π solves every quantitative instance P by Def. 6.

On the other hand, if any essentially qualitative policy π solves a problem P, then every abstracted trajectory for π, starting at s_I must be finite and terminate at a state satisfying the goal condition (by Lemma 2). By Lemma 1, we get that the abstracted trajectories for π starting from any s_I' that is qualitatively similar to s_I, must be finite and terminate at a goal state. Using the converse in Lemma 2, we get that the policy π in fact solves any problem instance with a state s_I' that is qualitatively similar to s_I, and therefore, any quantitative instance of $\tilde{P}$. Thus, $\tilde{\pi}$ is a solution for $\tilde{P}$. □

Lemma 3. *If a policy π solves a qualitative planning problem $\tilde{P}$ then it must be goal-closed.*

Proof. If a policy is not goal-closed, then there must be a non-goal terminal qstate $\tilde{t} \in ts(\pi, \tilde{s_I})$. This implies, by an inductive application of the completeness of action application (Theorem 1) over the finite path from $\tilde{s_I}$ to $\tilde{t}$, that it is possible to execute π and terminate at a non-goal $t_0 \in \gamma(\tilde{t})$ from any member $s_i \in \gamma(\tilde{s_I})$. □

References

Bonet, B.; Palacios, H.; and Geffner, H. 2009. Automatic derivation of memoryless policies and finite-state controllers using classical planners. In *Proc. of the 19th Intl. Conf. on Automated Planning and Scheduling*, 34–41.

Cimatti, A.; Pistore, M.; Roveri, M.; and Traverso, P. 2003. Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence* 147(1-2):35–84.

Helmert, M. 2002. Decidability and undecidability results for planning with numerical state variables. In *Proc. of the 6th Intl. Conf. on Automated Planning and Scheduling*, 303–312.

Hu, Y., and Levesque, H. J. 2010. A correctness result for reasoning about one-dimensional planning problems. In *Proc. of The 12th Intl. Conf. on Knowledge Representation and Reasoning*.

Kuipers, B. 1994. *Qualitative reasoning: modeling and simulation with incomplete knowledge*. MIT Press.

Levesque, H. J. 2005. Planning with loops. In *Proc. of the 19th International Joint Conference on Artificial Intelligence*, 509–515.

Srivastava, S.; Immerman, N.; and Zilberstein, S. 2008. Learning generalized plans using abstract counting. In *Proc. of the 23rd Conf. on AI*, 991–997.

Srivastava, S.; Immerman, N.; and Zilberstein, S. 2010. Computing applicability conditions for plans with loops. In *Proc. of the 20th Intl. Conf. on Automated Planning and Scheduling*, 161–168.

Tarjan, R. E. 1972. Depth-first search and linear graph algorithms. *SIAM Journal of Computing* 1(2):146–160.

Travé-Massuyès, L.; Ironi, L.; and Dague, P. 2004. Mathematical foundations of qualitative reasoning. *AI Magazine* 24:91–106.

Winner, E., and Veloso, M. 2007. LoopDISTILL: Learning domain-specific planners from example plans. In *Workshop on AI Planning and Learning, in conjunction with ICAPS*.