Numeric Planning via Search Space Abstraction

León Illanes and Sheila A. McIlraith
Department of Computer Science
University of Toronto, Toronto, Canada.
{lillanes,sheila}@cs.toronto.edu

Abstract

Many real-world planning problems are best modeled as infinite search space problems, using numeric fluents. Unfortunately, most planners and planning heuristics do not directly support such fluents. We propose a search space abstraction technique that compiles a planning problem with numeric fluents into a finite state propositional planning problem. To account for the loss of precision resulting from the abstraction, we leverage a policy repair technique used for non-deterministic planning and describe a new algorithm for planning with numeric fluents. We evaluate our approach on a set of benchmarks and compare it to state-of-the-art planners that deal with numeric fluents.

1 Introduction

Classical planning concerns itself with problems that are modelled through a restrictive propositional description. Since real-world applications often require richer modelling, a number of extensions to the formalisms of planning have been developed. For example, there has been interest in modelling problems specifying interaction among actions that can be executed concurrently [Boutilier and Brafman, 2001], problems with durative actions [Fox and Long, 2003], problems in which there are many uncontrollable possible outcomes for any given action [Daniele *et al.*, 1999; Cimatti *et al.*, 2003], problems in which the planner only has partial knowledge of the state [Cimatti *et al.*, 2004; Hoffmann and Brafman, 2006], or problems in which the planner has to keep track of physical state properties or quantifiable resources.

In this work, we are concerned with so-called *numeric planning problems* – problems modeled by the use of numeric fluents in addition to standard propositional fluents. The use of numeric fluents allows for modelling problems with either infinite or with continuous state spaces and has been commonly used to represent problems in which there are multiple interacting quantifiable resources.

Much of the work to date in numeric planning has adapted successful techniques from classical planning, often re-interpreting well known heuristics in this context. Some notable examples are the Metric-FF planner [Hoffmann, 2003], which extends the FF heuristic, and the LPRPG planner [Coles *et al.*, 2008], which augments RPG heuristics with linear numeric programming to better address optimality concerns. Other work has shown that local search techniques can also be effective in this context [Gerevini *et al.*, 2004]. More recent research suggests reformulation and abstraction techniques as other interesting approaches to consider [Chrpa *et al.*, 2015; Aldinger *et al.*, 2015].

Our work follows ideas related to these approaches. Indeed, we propose an abstraction approach in which we produce a classical planning problem that roughly represents the numeric problem with some loss in precision. We note that the loss of precision implies that a single action can have more than one possible outcome, depending on the underlying concrete numeric state. This insight reveals some similarities between planning with abstractions and a form of non-deterministic planning, and we take advantage of some existing techniques and concepts used to deal with non-determinism [Muise *et al.*, 2012] in order to build plans for the numeric domain out of plans for the abstract classical domain.

Our main contributions are the description of an abstraction technique for numeric planning problems, and an algorithm for a restricted class of these problems. We believe the approach described in this paper is just one particular realization of a very general idea regarding the use of abstraction in planning, where an abstract or partial plan can be used to generate a plan for a concrete problem.

2 Background

In this section, we give formal definitions relevant to our work. We focus on both classical planning and planning with numeric fluents, and on abstraction in the context of planning.

2.1 Classical Planning

A **classical planning problem** is a tuple $P = \langle F, O, I, G \rangle$. Here, F is a finite set of propositional fluents, O is a finite set of action operators, $I \subseteq F$ is the set of all true valued fluents in the initial state, and G is a conjunction of (possibly negated) literals over F that defines the goal condition. Every action operator $o \in O$ is defined by two conjunctions of fluent literals, $pre(o)$ and $eff(o)$, which respectively represent the action's preconditions and effects. Note that throughout this paper, we will often treat these conjunctions of literals as sets.

In this formalism, a state s can be represented as the set of fluents in F that correspond to everything that is true in the state. As such, we obtain that the set of all possible states in the problem is given by $S(P) = 2^F$. For each state there is a unique propositional valuation $\sigma(s) : F \rightarrow \{\text{true}, \text{false}\}$ that results from assigning every fluent in s to true and the rest to false. An operator o is applicable in s if the state's valuation is consistent with the action's preconditions: $\sigma(s) \models \mathit{pre}(o)$.

Given a state $s \in 2^F$ and an action operator $o \in O$ such that o is applicable in s, we can compute the successor state that results from applying o over s as $\delta(s, o) = (s \setminus \mathit{Del}) \cup \mathit{Add}$, where $\mathit{Add} = \{f \mid f \in \mathit{eff}(o)\}$ and $\mathit{Del} = \{f \mid \neg f \in \mathit{eff}(o)\}$.

2.2 Planning with Numeric Fluents

We define a **planning problem with numeric fluents**, henceforth a numeric planning problem, by extending the definition for classical planning into a tuple $P = \langle F, N, O, I, G \rangle$, introducing a set N of *numeric fluents* and associated numeric conditions and effects.

For this formalism, a state s is represented as a tuple $\langle s_P, s_N \rangle$ where s_P is the set of fluents in F that are true in the state and $s_N \in \mathbb{R}^{|N|}$ is a vector of real numbers that corresponds to the values assigned to the numeric fluents in N. Now, the set of all possible states is defined to be $S(P) = 2^F \times \mathbb{R}^{|N|}$.

Numeric conditions are defined to be inequalities of arithmetic expressions over $N \cup \mathbb{R}$. As such, the goal condition can be defined as a pair of sets $G = \langle G_P, G_N \rangle$, where G_P is a set of propositional fluent literals and G_N is a set of numeric conditions. An equivalent formulation can be used for the preconditions of action operator $o \in O$: $\mathit{pre}(o) = \langle \mathit{pre}_P(o), \mathit{pre}_N(o) \rangle$.

A numeric effect can be formalized as the assignment of the evaluation of an arithmetic expression over $N \cup \mathbb{R}$ to a particular numeric fluent. In this way, the effects of an operator $o \in O$ can be defined as a pair of sets $\mathit{eff}(o) = \langle \mathit{eff}_P(o), \mathit{eff}_N(o) \rangle$ where $\mathit{eff}_P(o)$ is a set of propositional fluent literals and $\mathit{eff}_N(o)$ is a set of numeric effects.

With this, we can define the successor state $\delta(o, s)$ resulting from applying an applicable operator $o \in O$ over some state $s \in S$. As expected, the propositional part is identical to the definition used for classical planning problems. The numeric fluents are computed by evaluating all the assigning expressions in $\mathit{eff}_N(o)$ with respect to the numeric values from s and subsequently assigning to $\delta(o, s)$. Any numeric fluents that are not assigned via o are assigned the value from s directly.

Restricted Case

For part of this paper, we will consider a very restricted form of numeric planning in which numeric expressions in conditions and effects are specially simple. In particular, we will assume numeric conditions are inequalities of the form $n \geq c$, for $n \in N$ and $c \in \mathbb{R}$. Numeric effects will be of the form $n \leftarrow n + c$, for $n \in N$ and $c \in \mathbb{R}$. These restrictions allow for a clearer description of our approach, although they represent a serious limitation in expressivity. Nonetheless, extending our work to more expressive cases is possible and further discussed in Section 6.

2.3 Transition Systems

Formally, a **labelled transition system** is defined as a tuple $\mathcal{T} = \langle S, L, T, s_0, S_G \rangle$. Here, S is the set of all possible states in the system, L is a set of transition labels, $T \subseteq S \times L \times S$ is a set of labelled transitions, $s_0 \in S$ corresponds to the initial state, and $S_G \subseteq S$ is the set of goal states.

A classical or numeric planning problem P induces a specific labelled transition system where the set of states is $S = S(P)$. The operators directly correspond to the labels and the set of transitions is $T = \{\langle s, o, \delta(s, o) \rangle \mid s \in S, o \in \mathit{app}(s)\}$, where $\mathit{app}(s) \subseteq O$ is the set of operators that are applicable in s. In both cases, $s_0 = I$ and S_G is straightforwardly derived from G.

A valid trace over $\mathcal{T}$ is any finite sequence $\langle s_0, o_1, s_1, o_2, \ldots, o_n, s_n \rangle$ where $\langle s_{i-1}, o_i, s_i \rangle \in T$ for all $i \in \{1, \ldots, n\}$. A trace is, then, an interleaved sequence of states and operators that represents a possible path within the transition system. We will call a trace successful when $s_n \in S_G$. For a successful trace, we have that the sequence of operators $\langle o_1, o_2, \ldots, o_n \rangle$ corresponds to a plan for the problem P.

2.4 Abstraction in Planning

In the context of planning, abstraction techniques are used to build smaller transition systems out of given planning problems by aggregating states together. Formally, an abstraction is a function $\alpha : S \rightarrow S^\alpha$ that maps concrete states from S to abstract states in S^α, with $|S^\alpha| \leq |S|$. This produces an abstract transition system $\mathcal{T}^\alpha = \langle S^\alpha, L, T^\alpha, s_0^\alpha, S_G^\alpha \rangle$ where $s_0^\alpha = \alpha(s_0)$, $S_G^\alpha = \{\alpha(s_G) \mid s_G \in S_G\}$, and $\langle s^\alpha, o, s^{\alpha\prime} \rangle \in T^\alpha$ if and only if there is at least one transition $\langle s, o, s' \rangle \in T$ such that $\alpha(s) = s^\alpha$ and $\alpha(s') = s^{\alpha\prime}$.

A common application of this idea is to automatically derive a sufficiently small abstract transition system that can be represented explicitly. Distances in this abstract space can be used as admissible heuristics for the concrete problem. Many leading approaches to optimal planning use abstractions in this manner (e.g., [Sievers *et al.*, 2012; Seipp and Helmert, 2013; 2014; Helmert *et al.*, 2014]).

2.5 Goal and Affordance Preserving Abstractions

State aggregation will result in some loss of information as whatever distinguishes two states s and s' is evidently lost if $\alpha(s) = \alpha(s')$. We can informally define a notion of perfect abstraction as an abstraction that does not lose any relevant information. Such an abstraction would be such that an abstract plan always can be converted into a plan for the concrete problem[1]. We can define many other properties to categorize abstractions based on what information they preserve

[1]A number of other important details regarding what makes an abstraction perfect go beyond the scope of this paper and are perhaps application specific. Should all concrete plans be representable in the perfect abstraction? Should optimality be a factor?

or lose. Two properties germaine to the work presented here are formally defined below and refer to preserving information that distinguishes goal states from non-goal states and information that distinguishes states in which different actions are applicable.

Definition 1. *A* ***goal preserving abstraction*** α *is an abstraction for transition system* $\mathcal{T} = \langle S, L, T, s_0, S_G \rangle$ *such that* $\alpha(s) \in S_G^\alpha$ *if and only if* $s \in S_G$.

Definition 2. *An* ***affordance preserving abstraction*** α *is an abstraction for transition system* $\mathcal{T} = \langle S, L, T, s_0, S_G \rangle$ *such that for every pair of states* $s, s' \in S$, $\alpha(s) = \alpha(s')$ *only if* $app(s) = app(s')$.

3 Abstraction for Numeric Planning

In this section we describe an abstraction for planning problems with numeric fluents as described in Section 2.2. The method takes a planning problem with numeric fluents and produces an abstract classical planning problem. In the next section, we show how we can use a plan for the abstract problem to find a plan for the original problem.

The basic mechanism behind our abstraction is to replace all numeric fluents by newly introduced propositional fluents that represent the specific numeric conditions that are needed to distinguish when actions are or are not applicable and when the goal has been reached. To achieve this we simply add a propositional fluent for each unique action precondition and goal condition in the problem. In this way, we produce a goal preserving and affordance preserving abstraction for the planning problem which maintains most of the dynamics of the domain while removing all numeric fluents.

As an example, consider a robot that can pick and load objects onto itself to carry them around. Assume the capabilities of this robot have been modelled into a numeric planning problem in which battery levels and carrying capacity are described through numeric fluents. Suppose the robot can only load an object if the object's weight does not exceed the robot's remaining capacity. Our abstraction would then introduce propositional fluents that represent directly whether or not each object in the model has a weight that exceeds the robot's current capacity. Furthermore, suppose the robot can only move from one place to another if its current battery levels surpass some value that is a function of its current load. The abstraction we are interested would incorporate a single propositional fluent that represents whether or not that condition is met. If this were the only condition for performing the `move` action, we would essentially include a fluent that directly indicates if the action is applicable.

As mentioned in previous sections, abstractions often induce some loss of information. Although we specifically produce goal preserving and affordance preserving abstractions, there is an important loss of information with regards to the effects of actions. This is easily illustrated through an example. Consider a simple setting in which an automated vehicle traverses through different locations using a unit amount of fuel each time. If the amount of fuel in the vehicle's tank is modelled as a numeric fluent, the abstraction process we have outlined would result in the introduction of a single propositional fluent to represent whether the amount of fuel is at least one unit. It is unclear if the proposition should become false or not after execution of the action. Indeed, both cases should be possible and the abstraction cannot distinguish between them. A possible solution to this issue would be to model the multiple possible effects as non-deterministic effects. Figures 1 and 2 show a more detailed example of this situation as it applies to the `move-ship` action of the Settlers domain [Long and Fox, 2003]. In this domain, there are a number of different resources that can be produced and consumed, and different actions consume different quantities. The `move-ship` action consumes two units of the `coal` resource, whereas a different action (`move-train`) consumes one unit. As such, our abstraction includes two new propositional fluents and effectively represents three intervals in which the assignment for the numeric fluent `(available coal ?v)` can be. Reducing the amount by two units can have three possible effects in which both, one, or none of the propositional fluents are deleted.

```
(:action move-ship
  :parameters
    (?v - vehicle
     ?p1 - place
     ?p2 - place)
  :precondition (and
    (is-ship ?v)
    (connected-by-sea ?p1 ?p2)
    (is-at ?v ?p1)
    (>= (available coal ?v) 2))
  :effect (and
    (not (is-at ?v ?p1))
    (is-at ?v ?p2)
    (decrease (available coal ?v) 2)
    (increase (pollution) 2)))
```

Figure 1: The `move-ship` action schema from the numeric Settlers domain.

In general, identifying all the possible effects of an action is a hard problem that merits further investigation. In this work, we limit formal analysis of this topic to the restricted case of numeric planning described in Section 2.2. Nonetheless, a simple approach for the general case might be to use some symbolic solver to test for each action and numeric condition whether or not the application of the action can (or will) make the condition true or false.

A final important point to make is that a correct policy for the resulting non-deterministic problem would be effective as a solution for the original problem. However, such a policy may not exist or be too hard to find. That said, we can use techniques similar to the ones used by non-deterministic planners to find solutions. In particular, we use the all-outcomes determinization [Muise *et al.*, 2012] to obtain a classical planning problem from the non-deterministic one. This point will be revisited in Section 4.

3.1 Interval Abstraction for the Restricted Case

For numeric planning problems that belong to the restricted case that we have described above, all numeric conditions are

```
(:action move-ship
  :parameters
    (?v - vehicle
     ?p1 - place
     ?p2 - place)
  :precondition (and
    (is-ship ?v)
    (connected-by-sea ?p1 ?p2)
    (is-at ?v ?p1)
    (available-gte2 coal ?v))
  :effect (and
    (not (is-at ?v ?p1))
    (is-at ?v ?p2)
    (oneof
      (and
        (not (available-gte1 coal ?v))
        (not (available-gte2 coal ?v)))
      (and
        (not (available-gte2 coal ?v)))
      (and))))
```

Figure 2: The `move-ship` action schema from the Settlers domain after abstraction. The only numeric precondition has been replaced by a condition on a new propositional fluent. The numeric effect on the `(pollution)` fluent is removed since it does not affect the dynamics of the problem. The numeric effect for the `(available coal ?v)` fluent has been replaced by a set of different possible effects over some of the new propositional fluents

of the form $n \geq c$, where n is a numeric fluent and c is a constant number. For any given problem, we might find a set of conditions $\{n \geq c_0, n \geq c_1, \ldots, n \geq c_m\}$. In the corresponding abstract problem we will have a set of propositional fluents respectively representing each of these conditions: $n^\alpha = \{p_{c_0}, p_{c_1}, \ldots, p_{c_m}\}$.

These propositional fluents correspond to an interval abstraction for the numeric fluent n. Indeed, if we assume $c_0 < c_1 < \ldots < c_m$, we can easily see how an assignment to the propositional fluents can be mapped to an interval. For example, if all the fluents in n^α are false, then the value must be in the interval $(-\infty, c_0)$. If only p_{c_0} is true, then the value must be in the interval $[c_0, c_1)$, and if p_{c_0} and p_{c_1} are the only true fluents the value must be in $[c_1, c_2)$. Note that to be consistent with the intended semantics the assignments have to be so that all fluents below a particular level are set to true, and all those above it are set to false. The point at which this phase change occurs corresponds to the specific interval defined by the assignment. If all the fluents are false or all are true then the numeric value has to be in one of the edge intervals.

The advantage of using an interval abstraction is that to understand all the possible effects of actions in this context, we only need to do basic interval algebra. Consider for instance a numeric effect that increases a numeric fluent n by some constant $k > 0$. If the numeric value for n was originally in the interval $[c_i, c_j)$, then the possible resulting intervals after the effect are all those that have a non empty intersection with the interval $[c_i + k, c_j + k)$.

4 Planning with the Interval Abstraction

In this section, we will discuss how we can exploit the type of abstraction described in Section 3 to obtain plans for planning problems with numeric fluents. A very general overview of the approach, the ASTER algorithm (AbSTract, Execute and Repair), is shown in Algorithm 1.

Algorithm 1: ASTER

Input: $P = \langle F, N, O, I, G \rangle$, a planning problem with numeric fluents
Output: π, a plan for P

1 $P^\alpha \leftarrow$ ABSTRACT(P)
2 $\pi^\alpha \leftarrow$ PLAN(P^α)
3 $\Pi^\alpha \leftarrow$ REGRESS(π^α)
4 $\pi \leftarrow \langle\rangle$
5 $s \leftarrow I$
6 **while** *s is not consistent with G* **do**
7   **if** *s^α is handled by Π^α* **then**
8   Append operator $\Pi^\alpha[s^\alpha]$ to π
9         $s \leftarrow$ apply operator $\Pi^\alpha[s^\alpha]$ over s
10   **else**
11   REPAIR(P, s, Π^α)
12   **if** REPAIR *was successful* **then**
13   Append path from REPAIR to π
14             $s \leftarrow$ state reached by REPAIR
15   **else**
16   Backtrack and REPAIR
17 **return** π

The algorithm works in four stages. First, in line 1 we call a procedure ABSTRACT that produces a classical planning by abstracting the numeric problem into a non-deterministic one as described in the previous sections and then returning the all-outcomes determinization. In line 2, we use any classical planner to obtain a plan in the abstract space. In line 3 we use a procedure REGRESS over the plan. This procedure repeatedly applies operation regression [Waldinger, 1977] from the goal over the operators in the plan, which results in a sequence of pairs of partial states and operators. We can interpret this as a partial policy that maps any state consistent with one of the partial states to the operator paired with that partial state[2]. We will say that an abstract state s^α is handled by the policy if the policy contains some partial state that is consistent with s^α. We will refer to the operator paired with the partial state as the operator selected by the policy for that state.

In what remains of the algorithm, we attempt to execute the policy over the original problem, repairing whenever we reach a state that cannot be handled.

Since the abstract space is a classical planning problem, we can use any classical planner to find a plan. This allows us to take advantage of any and all the advancements in the well developed field of classical planning. Regressing the obtained plan can be done efficiently and the result

[2]If a state is consistent with more than one partial state, we use the pair that is closest to the end of the sequence.

is a mapping of partial states to actions such that it effectively corresponds to a partial policy for the abstract problem. This approach is based on one used in PRP, a state-of-the-art planner for fully observable non-deterministic planning problems [Muise *et al.*, 2012]. Indeed, as mentioned before, there are a number of parallels between our abstract planning space and non-deterministic planning. In particular, our abstraction considers multiple possible outcomes for operators which can be interpreted as non-determinism. However, the underlying dynamics of our problem are completely deterministic, which violates some important assumptions often used for non-deterministic planning.

Given a policy Π^α for the classical problem and a state s^α, we let $\Pi^\alpha[s^\alpha]$ refer to the operator selected by the policy for the state s^α. We extend this notation in the obvious way so that for a state s from the original numeric planning problem $\Pi^\alpha[s]$ refers to the operator selected by the policy for the corresponding state s^α.

4.1 Simulating and Repairing

Simulating the execution of the abstract policy over the concrete numeric problem space is a simple process. We just need to keep track of a concrete numeric state and the corresponding abstract state. Since our abstraction guarantees that all concrete states matching a particular abstract state have the same applicable operators, we know that if the partial policy handles the abstract state then the specific selected operator will be applicable on the concrete state. Similarly, we know that if we reach the goal state in the abstract space we will also have reached the goal in the concrete space. As such, the only interesting consideration is the case in which execution of an operator led to a concrete state that maps to some abstract state that is not handled by the policy. It is precisely in this case where we must perform some sort of repair.

Here we propose one of the simplest possible approaches to the repair process, as outlined in Algorithms 2 and 3. The approach works by doing blind, breadth-first search in the concrete numeric space around the reached state until reaching some other state that is handled by the policy. At this point we verify whether continuing to follow the policy in the abstract space will lead to the goal or would produce a loop. In the first case, the repair is done. In the second case, the blind search continues. If the repair fails, the planner must backtrack, effectively jumping back to the previous abstract state, and then attempt to repair from there.

Algorithm 2: REPAIR

Input: P, a planning problem with numeric fluents; Π, a partial policy for P; and s, a state not handled by Π

while *it is possible to continue the search* **do**
  Start or continue BFS around s until reaching some state t handled by Π
  if SAFE(t, Π) **then**
    return *success*
return *failure*

Algorithm 3: SAFE

Input: s, a state from a planning problem; Π a partial policy for P

Let s^α be the corresponding state for s in the abstract problem P^α
$c \leftarrow s^\alpha$
Loop
  $c \leftarrow$ the result of applying operator $\Pi[c]$ over state c
  if *c is consistent with G^α* **then**
    return ***true***
  else if $c = s^\alpha$ **then**
    return ***false***

5 Experimental Evaluation

In this section we give some brief details regarding the implementation of our approach in practice, and show some basic experimental results that validate the feasibility and effectiveness of the method.

Our implementation was built on top of the implementation of PRP [Muise *et al.*, 2012], which is itself an augmentation of the Fast Downward planning system [Helmert, 2006] to effectively deal with non-deterministic actions. Since we use many similar techniques, we can reuse some of the machinery in place for finding and regressing the initial plan to generate a policy. We further the augmentation by allowing the planner to read and automatically abstract a numeric planning problem in the way described in the previous sections. We use the FF heuristic [Hoffmann and Nebel, 2001] and a greedy search algorithm to generate the initial plan.

We compare our approach, the ASTER algorithm, to the Metric-FF algorithm [Hoffmann, 2003] and the LPRPG algorithm [Coles *et al.*, 2008]. For all experiments we use the Settlers domain from the 3rd International Planning Competition (IPC) [Long and Fox, 2003]. This is an interesting domain that exhibits complex interaction between different numeric fluents. The problems model a setting with a number of different locations that require collecting resources and building infrastructure and transportation. Resources are consumed when building, and while some resources can be produced directly at certain locations (e.g.: timber at a woodlands location), other resources can only be produced by refining existing resources (e.g.: refined wood is produced from timber).

In our experiments we consider the same problem instances used for the IPC[3]. However, since our approach does not handle optimization, we ignore the optimization requirements in all cases. We ran all experiments on a Linux machine with a 2.2GHz Intel Xeon E5 CPU, limiting the running time to a maximum of 30 minutes. Resulting times and plan lengths obtained by all three algorithms in each problem instance are summarized in Table 1.

For the problems from the Settlers domain without optimization, our approach seems to be more effective than either

[3]We omit problem 8, which is actually unsolvable. All three algorithms considered in this section are immediately able to recognize the problem as unsolvable.

Prob.	Time (s)			Plan length		
	M-FF	LPRPG	ASTER	M-FF	LPRPG	ASTER
ipc-1	3.98	**0.29**	17.59	**52**	59	64
ipc-2	**0.01**	0.10	15.75	**25**	26	26
ipc-3	24.15	38.25	**17.14**	**101**	113	105
ipc-4	93.41	**0.41**	20.42	76	**70**	80
ipc-5	0.28	**0.23**	21.86	76	76	**72**
ipc-6	**10.45**	43.90	20.94	**77**	117	89
ipc-7	T/O	T/O	**33.69**	—	—	**184**
ipc-9	T/O	T/O	T/O	—	—	—
ipc-10	T/O	607.52	**63.74**	—	245	**167**
ipc-11	1798.61	892.11	**42.66**	**195**	262	200
ipc-12	539.36	T/O	**38.63**	**147**	—	162
ipc-13	1105.52	T/O	**70.98**	**230**	—	241
ipc-14	T/O	T/O	T/O	—	—	—
ipc-15	1114.18	T/O	**84.39**	**235**	—	240
ipc-16	T/O	T/O	**129.95**	—	—	**340**
ipc-17	T/O	T/O	T/O	—	—	—
ipc-18	T/O	T/O	T/O	—	—	—
ipc-19	T/O	T/O	T/O	—	—	—
ipc-20	T/O	T/O	**280.10**	—	—	**389**
solved	10	8	**14**			

Table 1: Results on the Settlers domain, where ASTER finds more solutions than Metric-FF and LPRPG combined. ASTER often finds a solution more than an order of magnitude faster than the other planners. T/O is time out.

Metric-FF or LPRPG. Indeed, our approach solves 3 more problems than the other two algorithm combined, and finds plans faster on 6 of the remaining problems. These solutions are found at least one and sometimes two orders of magnitude faster than the other planners.

At the same time, there is a small but somewhat consistent degradation in the quality of the plans found. Nonetheless, a particularly interesting point is that whenever our algorithm terminates, it does so in under 5 minutes of time. In cases where optimization is important, our algorithm could be used as a first attempt to find a solution in a very short amount of time before attempting to find a near optimal solution with some other method. The suboptimal solution discovered by ASTER can then be used as a fallback whenever the optimal algorithm doesn't terminate in time.

6 Extensions and Future Work

As described, our methods are practical for a restricted class of numeric planning problems. In this section we discuss what is required to extend the approach to a more general case, and how the ideas presented in the paper could be applied in other contexts.

At a high level, our algorithm works by taking a numeric planning problem and generating a roughly equivalent classical planning one in which all numeric conditions that are mentioned in the original domain are associated to some new propositional fluent that should hold whenever the numeric condition holds. Identifying the possible outcomes of a given numeric effect implies understanding which of the numeric conditions can change from true to false or vice versa after the numeric transformation represented by the effect is applied. If, as we've required so far, all conditions are of the form $n \geq c$ where n is a fluent and c is a constant, then this process is easy to do. As mentioned before, for this case the resulting abstraction is an interval abstraction and all that is needed to evaluate the effects is a basic use of interval arithmetics.

Extending the abstractions to consider propositional fluents that represent more elaborate conditions is certainly possible. Understanding the possible outcomes of some transformation over the numeric variables requires the use of some solver capable of handling the particular theory over which the conditions are defined. Alongside interval abstractions, more expressive abstractions for numeric variables, such as convex polyhedra, have been extensively studied within the field of Abstract Interpretation for Static Analysis of Software [Cousot and Cousot, 1976; 1977; 1979; Cousot and Halbwachs, 1978]. Techniques for applying the corresponding numeric transformations over the abstractions are well understood, and adapting them to our context is feasible.

Other interesting directions in which the methods proposed in this paper can be extended involve applying the basic idea of abstracting or relaxing a planning problem and then generating a policy for the abstract plan that can be used to subsequently generate a plan for the original task. This approach is not limited only to numeric planning problems. We are interested in investigating abstraction techniques similar to certain reformulation approaches that identify resources encoded into classical planning problems [Riddle *et al.*, 2015; Fuentetaja and de la Rosa, 2016]. In these works, indistinguishable objects from the planning problems are grouped together to reduce symmetries and therefore reduce the complexity of the task.

7 Conclusions

We have given a method for generating interval abstractions for a class of numeric planning problems and described a planning algorithm that exploits such abstractions to generate plans for the original problems. For an interesting benchmark domain, we find that our planner often obtains solution much faster than two other very effective algorithms.

Although the class of problems we can handle is limited, we've give insight into how we could adapt our algorithm to work in more general cases. We believe our approach is easy to extend into general techniques for many interesting problems in planning, and that it offers worthy directions for further research.

Acknowledgements: We gratefully acknowledge funding from the Natural Sciences and Engineering Research Council of Canada (NSERC). We also would like to thank the anonymous reviewers for insightful feedback and helpful comments.

References

[Aldinger *et al.*, 2015] Johannes Aldinger, Robert Mattmüller, and Moritz Göbelbecker. Complexity of interval relaxed numeric planning. In *KI 2015: Advances in Artificial Intelligence*, pages 19–31. Springer, 2015.

[Boutilier and Brafman, 2001] Craig Boutilier and Ronen I Brafman. Partial-order planning with concurrent interacting actions. *Journal of Artificial Intelligence Research*, pages 105–136, 2001.

[Chrpa *et al.*, 2015] Lukáš Chrpa, Enrico Scala, and Mauro Vallati. Towards a reformulation based approach for efficient numeric planning: Numeric outer entanglements. In *Proceedings of the 8th Symposium on Combinatorial Search (SOCS)*, 2015.

[Cimatti *et al.*, 2003] Alessandro Cimatti, Marco Pistore, Marco Roveri, and Paolo Traverso. Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence*, 147(1):35–84, 2003.

[Cimatti *et al.*, 2004] Alessandro Cimatti, Marco Roveri, and Piergiorgio Bertoli. Conformant planning via symbolic model checking and heuristic search. *Artificial Intelligence*, 159(1):127–206, 2004.

[Coles *et al.*, 2008] Andrew Coles, Maria Fox, Derek Long, and Amanda Smith. A Hybrid Relaxed Planning Graph-LP Heuristic for Numeric Planning Domains. In *Proceedings of the 18th International Conference on Automated Planning and Sched. (ICAPS)*, pages 52–59, 2008.

[Cousot and Cousot, 1976] Patrick Cousot and Radhia Cousot. Static determination of dynamic properties of programs. In *Proceedings of the 2nd International Symposium on Programming*, pages 106–130, 1976.

[Cousot and Cousot, 1977] Patrick Cousot and Radhia Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 238–252. ACM, 1977.

[Cousot and Cousot, 1979] Patrick Cousot and Radhia Cousot. Systematic design of program analysis frameworks. In *Proceedings of the 6th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 269–282. ACM, 1979.

[Cousot and Halbwachs, 1978] Patrick Cousot and Nicolas Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Proceedings of the 5th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 84–96. ACM, 1978.

[Daniele *et al.*, 1999] Marco Daniele, Paolo Traverso, and Moshe Y Vardi. Strong cyclic planning revisited. In *Recent Advances in AI Planning*, pages 35–48. Springer, 1999.

[Fox and Long, 2003] Maria Fox and Derek Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20:61–124, 2003.

[Fuentetaja and de la Rosa, 2016] Raquel Fuentetaja and Tomás de la Rosa. Compiling irrelevant objects to counters. Special case of creation planning. *AI Communications*, 29(3):435–467, 2016.

[Gerevini *et al.*, 2004] Alfonso Gerevini, Alessandro Saetti, and Ivan Serina. Planning with numerical expressions in LPG. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI)*. Citeseer, 2004.

[Helmert *et al.*, 2014] Malte Helmert, Patrik Haslum, Jörg Hoffmann, and Raz Nissim. Merge-and-shrink abstraction: A method for generating lower bounds in factored state spaces. *Journal of the ACM (JACM)*, 61(3):16, 2014.

[Helmert, 2006] Malte Helmert. The Fast Downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

[Hoffmann and Brafman, 2006] Jörg Hoffmann and Ronen I Brafman. Conformant planning via heuristic forward search: A new approach. *Artificial Intelligence*, 170(6):507–541, 2006.

[Hoffmann and Nebel, 2001] Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

[Hoffmann, 2003] Jörg Hoffmann. The Metric-FF planning system: Translating "ignoring delete lists" to numeric state variables. *Journal of Artificial Intelligence Research*, 20:291–341, 2003.

[Long and Fox, 2003] Derek Long and Maria Fox. The 3rd international planning competition: Results and analysis. *Journal of Artificial Intelligence Research*, 20:1–59, 2003.

[Muise *et al.*, 2012] Christian Muise, Sheila A. McIlraith, and J. Christopher Beck. Improved Non-deterministic Planning by Exploiting State Relevance. In *Proceedings of the 22nd International Conference on Automated Planning and Sched. (ICAPS)*, 2012.

[Riddle *et al.*, 2015] Patricia J Riddle, Michael W Barley, Santiago Franco, and Jordan Douglas. Automated transformation of pddl representations. In *Proceedings of the 8th Symposium on Combinatorial Search (SOCS)*, 2015.

[Seipp and Helmert, 2013] Jendrik Seipp and Malte Helmert. Counterexample-guided cartesian abstraction refinement. In *Proceedings of the 23rd International Conference on Automated Planning and Sched. (ICAPS)*, 2013.

[Seipp and Helmert, 2014] Jendrik Seipp and Malte Helmert. Diverse and additive cartesian abstraction heuristics. In *Proceedings of the 24th International Conference on Automated Planning and Sched. (ICAPS)*, 2014.

[Sievers *et al.*, 2012] Silvan Sievers, Manuela Ortlieb, and Malte Helmert. Efficient implementation of pattern database heuristics for classical planning. In *Proceedings of the 5th Symposium on Combinatorial Search (SOCS)*, 2012.

[Waldinger, 1977] Richard Waldinger. Achieving several goals simultaneously. In *Machine Intelligence 8*, pages 94–136. Ellis Horwood, Edinburgh, Scotland, 1977.