Numeric Planning via Abstraction and Policy Guided Search

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

Abstract

The real-world application of planning techniques often requires models with numeric fluents. However, these fluents are not directly supported by most planners and heuristics. We describe a family of planning algorithms that takes a numeric planning problem and produces an abstracted representation that can be solved using any classical planner. The resulting abstract plan is generalized into a policy and then used to guide the search in the original numeric domain. We prove that our approach is sound, and evaluate it on a set of standard benchmarks. Experiments demonstrate competitive performance when compared to other well-known algorithms for numeric planning, and a significant performance improvement in certain domains.

1 Introduction

Automated planning techniques are designed to be used for solving problems that can be modeled as the task of finding a course of actions that will transform a given initial state into a goal state. In classical planning the space of possible states is defined by a finite set of propositional fluents. Although this formalism can be used to express many interesting real-world problems, it assumes a finite number of possible states. This can be a limiting factor when modeling problems in which the current state involves resources (e.g., the amount of remaining fuel in an autonomous vehicle) or physical properties (e.g., the current velocity of a vehicle), which has motivated the development of planning systems that allow numeric fluents as part of the domain description.

These *numeric planning problems* can be solved with techniques similar to those used for classical planning. Indeed, much of the research relating to numeric planning has focused on extending existing systems and well-known heuristics for classical planning to work with numeric fluents. A notable example is the Metric-FF planner [Hoffmann, 2003], which extends the FF delete relaxation heuristic [Hoffmann and Nebel, 2001]. The LPRPG planner [Coles *et al.*, 2008] further extends this using linear programming to reason about the numeric fluents linked in consumer-producer relations. A different type of relaxation, based on the subgoaling ideas used for the h^{max} and h^{add} heuristics [Haslum and Geffner, 2000; Bonet and Geffner, 2001], was recently described by Scala *et al.* (2016). Other work has proposed techniques based on problem reformulation and abstraction [Chrpa *et al.*, 2015; Illanes and McIlraith, 2016a; 2016b].

The work presented in this paper completes and extends the approach used by the ASTER algorithm, described in previous work [Illanes and McIlraith, 2016a; 2016b]. ASTER constructs an abstraction based on the numeric conditions present in the problem at hand. The abstraction results in a classical planning problem, and execution of a partial policy obtained for the abstract problem is attempted on the concrete numeric problem and repaired as needed. However, ASTER has several limitations. First, the abstraction process it uses is based on numeric intervals, and is only applicable on a restricted class of problems. Similarly, the repair procedure used in ASTER is very simplistic and prone to falling into dead-ends. The main contributions of this paper are a more general abstraction method that is capable of dealing with arbitrary numeric expressions, and a novel search algorithm, ARGUS, that makes use of the obtained abstract policy to guide search in a way that is not hampered by the presence of dead-ends. In addition, we present theoretical and empirical results that show our approach is correct and effective in practice.

2 Background

In this section, we define and describe the notation and formalisms used throughout the paper. This includes classical and numeric planning, the use of abstraction in the context of planning, and a basic heuristic search algorithm for planning.

2.1 Classical Planning

We consider **classical planning problems** formalized through a finite-domain representation similar to that of SAS$^+$ tasks [Bäckström and Nebel, 1995]. A particular problem is defined by a tuple $\Pi = \langle \mathcal{V}, s_0, s_\star, \mathcal{O} \rangle$ where:

- $\mathcal{V}$ is a finite set of state variables. Each variable $v \in \mathcal{V}$ has a fixed finite domain $\mathcal{D}_v$. A partial assignment to the variables in $\mathcal{V}$ or **partial classical state** is a function s over $\mathcal{V}$ such that $s(v) \in \mathcal{D}_v \cup \{\bot\}$ for every $v \in \mathcal{V}_s$. Here, $s(v) = \bot$ signifies that v is **undefined** in s. Otherwise, v is **defined** in s. If all variables in $\mathcal{V}$ are defined in s, so that $s(v) \neq \bot$ for every $v \in \mathcal{V}$, s is known as a **classical state** over $\mathcal{V}$.

- The **initial classical state** s_0 is a classical state over $\mathcal{V}$.
- The **goal** $s_\star$ is a partial classical state over $\mathcal{V}$.
- $\mathcal{O}$ is a finite set of **operators** over $\mathcal{V}$. An operator $o \in \mathcal{O}$ is defined by a tuple $\langle l_o, \mathit{pre}_o, \mathit{eff}_o \rangle$. Respectively, pre_o and eff_o correspond to the **preconditions** and **effects** of o and are both partial classical states. l_o is the **label** or name of o.

To formalize the semantics of a planning problem, we use the notion of partial state entailment.

Definition 1 (Entailment)**.** *Given two partial classical states, s and s', we say s entails s' if for every $v \in \mathcal{V}$ that is defined in s', $s'(v) = s(v)$. This is denoted by $s \models s'$.*

An operator $o \in \mathcal{O}$ is **applicable** over s if and only if $s \models \mathit{pre}_o$. The **successor** resulting from applying an applicable operator o over partial classical state s is the partial classical state defined by $\delta_\Pi(s, o)$:

- $\delta_\Pi(s, o)(v) = \mathit{eff}_o(v)$, if v is defined in eff_o.
- $\delta_\Pi(s, o)(v) = s(v)$, if v is undefined in eff_o.

The notions of applicability and operator application can be extended to consider a sequence of operators in an obvious manner. Given a classical planning problem Π, the objective is to find a sequence of operators $\mathbf{o}^\star = [o_0, o_1, \ldots, o_n]$ applicable over s_0, such that $\delta_\Pi(s_0, \mathbf{o}^\star) \models s_\star$. Such a sequence is called a **plan**.

2.2 Numeric Planning

A **numeric planning problem** can be defined by extending the definition of a classical planning problem into a tuple $\Pi = \langle \mathcal{V}, \mathcal{N}, s_0, n_0, s_\star, n_\star, \mathcal{O} \rangle$.

- $\mathcal{V}$, s_0 and $s_\star$ are the same as in the classical formalism.
- $\mathcal{N}$ is a finite set of **numeric variables**. Each numeric variable $w \in \mathcal{N}$ has domain $\mathbb{R}$. A partial assignment to the numeric variables in $\mathcal{N}$ or **numeric state** is a function $n : \mathcal{N} \mapsto \mathbb{R}_\perp = \mathbb{R} \cup \{\perp\}$. As before, $n(w) = \perp$ signifies w is undefined in n. A pair $S = \langle s, n \rangle$ consisting of a classical state and a numeric state is called a **state**.
- n_0 is the **initial numeric state**, a numeric state over $\mathcal{N}$.
- $n_\star$ is the **numeric goal**.
- $\mathcal{O}$ is, again, a set of operators. Each $o \in \mathcal{O}$ is defined by a tuple $\langle l_o, \mathit{pre}_o, \mathit{eff}_o, \mathit{pre}^N_o, \mathit{eff}^N_o \rangle$. Here, l_o is, again, the label of the operator. pre_o and eff_o correspond to the **classical preconditions** and **classical effects**, respectively. As in the classical case, these are partial classical states. pre^N_o and eff^N_o are the **numeric preconditions** and **numeric effects** of o.

To formally define the numeric goal, numeric preconditions and numeric effects, we need to define some auxiliary concepts. The numeric goal and numeric preconditions are both defined as sets of **numeric conditions**. Numeric effects are sets of **numeric assignments**.

Definition 2 (Numeric Condition)**.** *A numeric condition c is defined by a comparator $\mathbf{C} \in \{<, \leqslant, =, \geqslant, >\}$ and two arithmetic expressions over $\mathcal{N}$ and $\mathbb{R}$, **lhs** and **rhs**. Given a numeric state n, c can be evaluated over n by substituting in **lhs** and **rhs** every instance of every numeric variable $w \in \mathcal{N}$ by the value $n(w)$. c evaluates to `true` if and only if every w appearing in **lhs** or **rhs** is defined in n and the comparison **lhs** $\mathbf{C}$ **rhs** holds. Otherwise, c evaluates to `false`. Abusing notation, we will refer to the evaluation of c over n as $n(c)$.*

Definition 3 (Numeric Entailment)**.** *Given a numeric state, n, and a set of numeric conditions, n', we say n entails n' if for every $c \in n'$, $n(c) =$ `true`. We overload the notation for the classical entailment, and denote this as $n \models n'$.*

Definition 4 (Numeric Assignment)**.** *A numeric assignment a is defined by a numeric variable $w_a \in \mathcal{N}$ and an arithmetic expression **exp** over $\mathcal{N}$ and $\mathbb{R}$. a can be applied over a numeric state n by substituting in **exp** every instance of every numeric variable $w \in \mathcal{N}$ by the value of $n(w)$. The resulting arithmetic expression over $\mathbb{R} \cup \{\perp\}$ can be evaluated into a single value $r \in \mathbb{R} \cup \{\perp\}$. The result of the application is a numeric state n_a such that $n_a(w_a) = r$ and $n_a(w) = \perp$ for every $w \neq w_a$.*

Now, an operator $o = \langle l_o, \mathit{pre}_o, \mathit{eff}_o, \mathit{pre}^N_o, \mathit{eff}^N_o \rangle$ is applicable over state $S = \langle s, n \rangle$ if and only if $s \models \mathit{pre}_o$ and $n \models \mathit{pre}^N_o$. The result of applying an applicable operator o over S is $\delta_\Pi(S, o) = \langle \delta_\Pi(s, o), \delta_\Pi(n, o) \rangle$. Here $\delta_\Pi(s, o)$ corresponds to the result of applying the classical part of o over s. $\delta_\Pi(n, o)$ is given by:

- $\delta_\Pi(n, o)(w) = n_a(w)$, if there is some numeric assignment $a \in \mathit{eff}^N_o$ such that $w_a = w$[1].
- $\delta_\Pi(n, o)(w) = n(w)$, otherwise.

Extending the notions of applicability and operator application to deal with sequences of operators can be done in the same way as for the classical case. A plan for a numeric planning problem Π is a sequence of operators $\mathbf{o}^\star$ that is applicable in $\langle s_0, n_0 \rangle$, such that $\delta_\Pi(s_0, \mathbf{o}^\star) \models s_\star$ and $\delta_\Pi(n_0, \mathbf{o}^\star) \models n_\star$.

2.3 Planning as Search

A successful and well-known approach to automated planning is to use **state-space search**. Algorithm 1 outlines a very basic template for a numeric planning algorithm, where the state space is traversed in different ways depending on the strategy employed to remove states from OPEN. A common approach uses heuristic estimates of the distance to go combined with action costs or path lengths in order to determine which state to expand next.

2.4 Abstraction

Abstraction techniques have often been used to aid the search process in automated planning. Early work focused on constructing abstraction hierarchies that decompose a large planning problem into several smaller ones. Examples of this approach include ABSTRIPS [Sacerdoti, 1974] and ALPINE [Knoblock, 1994]. Recent focus has been on state aggregation to produce sufficiently small abstract spaces that can be represented explicitly. Distances in these abstract

[1]Note that this results in an ambiguity if there are two or more numeric assignments in eff^N_o that affect the same numeric variable. As such, the formalism requires that this does not happen.

Algorithm 1: Best-first search for numeric planning

```
1  Algorithm Search(Π)
       Input: Π = ⟨V, N, s_0, n_0, s_⋆, n_⋆, O⟩, a numeric planning
              problem
       Output: A plan π for Π (or false if no plan exists)
2      OPEN ← {⟨s_0, n_0⟩}
3      CLOSED ← ∅
4      PARENT[⟨s_0, n_0⟩] ← NULL
5      while OPEN ≠ ∅ do
6          Remove some state S from OPEN.
7          G ← Expand(S)
8          if G ≠ NULL then
9              return Extract(G)
10     return false
11 Procedure Expand(S = ⟨s, n⟩)
12     if s ⊨ s_⋆ and n ⊨ n_⋆ then
           /* S is a goal state                    */
13         return S
14     CLOSED ← CLOSED ∪ {S}
15     for o ∈ O such that o is applicable in S do
16         S' ← δ_Π(S, o)
17         if S' ∉ OPEN and S' ∉ CLOSED then
18             PARENT[S'] ← ⟨S, o⟩
19             OPEN ← OPEN ∪ {S'}
20     return NULL
21 Procedure Extract(S)
22     π ← []
23     while S ≠ NULL do
24         ⟨S', o⟩ ← PARENT[S]
25         Append o at the beginning of π.
26         S ← S'
27     return π
```

spaces can be used as admissible heuristics for the concrete space. Many leading approaches to optimal planning use such abstractions [Edelkamp, 2001; Sievers *et al.*, 2012; Seipp and Helmert, 2013; 2014; Helmert *et al.*, 2014].

In addition, abstraction has been a key aspect of model-checking and software verification techniques for many decades. The formalization of Abstract Interpretation [Cousot and Cousot, 1977] has been an important tool for static analysis of software. A special form of Abstract Interpretation, Predicate Abstraction [Graf and Saïdi, 1997], has been widely used in model-checking and has many similarities with the abstraction approach we use in our work.

Indeed, we are interested in state-aggregation abstractions. Unlike most planning applications of abstraction we do not use the abstractions to build heuristics and we do not need to represent the resulting state spaces explicitly. The type of abstractions that we consider will take a planning problem and produce another planning problem.

A **state-aggregation abstraction** is defined as a function $\alpha : \mathcal{S} \mapsto \mathcal{S}^\alpha$ that maps states from one state space to another. The domain of α corresponds to the **concrete state space** and its range is the **abstract state space**. Given an abstraction α, we define its concretization function $\gamma : \mathcal{S}^\alpha \mapsto 2^{\mathcal{S}}$ as:

$$\gamma(S^\alpha) = \{S \mid \alpha(S) = S^\alpha\}$$

For a numeric planning problem $\Pi = \langle \mathcal{V}, \mathcal{N}, s_0, n_0, s_\star, n_\star, \mathcal{O} \rangle$ as defined above, we are interested in abstractions for the state space defined by its classical and numeric variables: $\mathcal{S} = \times_{v \in \mathcal{V}} \mathcal{D}_v \times \mathbb{R}_\perp^{|\mathcal{N}|}$. Furthermore, given an abstraction function for this space, it induces an **abstract transition function** $\delta_\alpha : \mathcal{S}^\alpha \times \mathcal{O} \mapsto 2^{\mathcal{S}^\alpha}$ defined as follows:

$$\delta_\alpha(S^\alpha, o) = \{\alpha(S) \mid \exists_{S^\gamma \in \gamma(S^\alpha)} \delta_\Pi(S^\gamma, o) = S\}$$

The semantic intention behind this is that given an existing transition in the concrete state space, there must exist a corresponding transition in the abstract state space.

2.5 Plan Regression and Partial Policies

A possible generalization of planning problems involves finding more than a sequential plan. A **policy** for a problem Π is a function that maps states to operators. A **partial policy** is a policy that is defined only over a subset of the states in the problem.

In classical planning, **operator regression** [Waldinger, 1977] of an operator o over a partial state s is a process that allows to determine the necessary and sufficient conditions that must hold in a partial state s' such that $\delta(s', o) = s$. Repeatedly applying regression from the goal of a planning problem, working backwards through a plan, identifies the relevant properties of intermediate states that allow the plan to work [Fritz and McIlraith, 2007], effectively generalizing the plan into a partial policy [Muise *et al.*, 2012].

Given a partial policy $\mathcal{P}$ and a classical state s, we will denote the operator indicated by the policy for the state as $\mathcal{P}(s)$. If $\mathcal{P}$ is not defined for s, we will say $\mathcal{P}(s) = \text{NULL}$.

Note that given a planning problem and an abstraction α as defined above, a partial policy $\mathcal{P}^\alpha$ for the resulting abstract problem can be easily adapted into a policy for the concrete problem. We extend the notion of concretization onto partial policies so that given a partial policy $\mathcal{P}^\alpha$ for the abstract problem we define the corresponding partial policy for the concrete problem $\mathcal{P} = \gamma(\mathcal{P}^\alpha)$ according to the formula $\mathcal{P}(s) = \mathcal{P}^\alpha(\alpha(s))$.

3 The ARGUS Algorithm

In this section we give a high-level overview of our numeric planner. ARGUS (Abstraction and Regression to Guide Search) takes advantage of a given abstraction procedure that takes a numeric planning problem and produces an abstracted version in the form of a classical planning problem. It then uses a classical planner to find a solution for this abstract problem. Our abstraction procedure is described in Section 4.

The algorithm works in four stages, as outlined in Algorithm 2. First, it generates an abstracted version of the problem at hand by using the given `Abstract` procedure. It then solves the abstract problem through the use of the classical planner. It subsequently generalizes the obtained plan into a partial policy by regressing from the goal over the obtained plan. Note that regression of the *abstract* plan π^α actually

Algorithm 2: Overview of the ARGUS algorithm

```
1  Algorithm ARGUS(Π)
       Input: Π = ⟨V, N, s_0, n_0, s_⋆, n_⋆, O⟩, a numeric planning
              problem
       Output: A plan π for Π
2      Π^α ← Abstract(Π)
3      π^α ← Plan(Π^α)
4      P ← γ(Regress(π^α))
5      return PolicyGuidedSearch(Π, P)
```

produces an *abstract* policy that is subsequently concretized into the *concrete* policy $\mathcal{P}$. Finally, it uses a policy-guided search algorithm to search in the numeric planning problem space until reaching a goal state.

The POLICYGUIDEDSEARCH algorithm is based on the basic SEARCH algorithm described in Algorithm 1, but takes advantage of the existing policy through a modification of the expansion procedure. Effectively, the only change made to Algorithm 1 is to substitute the call to EXPAND for a call to POLICYGUIDEDEXPAND. The modified expansion procedure is shown in Algorithm 3.

Algorithm 3: Policy Guided Expansion Procedure

```
1  Procedure PolicyGuidedExpand(S = ⟨s, n⟩, P)
2      if s ⊨ s_⋆ and n(c) = true for every c ∈ n_⋆ then
           /* S is a goal state                    */
3          return S
4      CLOSED ← CLOSED ∪ {S}
5      for o ∈ O such that o is applicable in S do
6          S' ← δ_Π(S, o)
7          if S' ∉ OPEN and S' ∉ CLOSED then
8              PARENT[S'] ← ⟨S, o⟩
9              if P(S) = o then
10                 G = PolicyGuidedExpand(S', P)
11                 if G ≠ NULL then
12                     return G
13             else
14                 OPEN ← OPEN ∪ {S'}
15     return NULL
```

Intuitively, the idea behind policy guided search is to accelerate the search process by greedily following advice given by the policy. Whenever a state is expanded we verify if the policy dictates an applicable action should be used in it. If this is true, we proceed to recursively expand the resulting state. Effectively, this means that the search algorithm quickly explores parts of the search space that are covered by the policy.

It is easy to see that the policy guided expansion procedure used in policy guided search does not allow traversal of any impossible transition. Therefore, any expanded state –be it a direct expansion or a recursive one– is reachable from its `Parent`, through the corresponding action. This justifies the following observation.

Observation 1. *Given a valid state-aggregation abstraction,* ARGUS *is sound. Whenever a goal state is about to be expanded, a sound plan is extracted and returned by the algorithm.*

However, note that the algorithm as described is not *complete*. Although the abstract policy obtained by regressing a plan for the abstract problem cannot exhibit any loops, its application over the concrete problem may result in infinite loops in the abstract space. Consider the case in which the abstraction is extremely coarse and can only distinguish between goal and non-goal states. If the initial state is not a goal state, an abstract plan will consist of a single action that transitions from a non-goal state to a goal state. If this action is always applicable (e.g., by having no preconditions) and repeated application of it will never revisit a state (e.g., by always incrementing some numeric variable), then ARGUS will continue to recursively apply this action and expand the reached state.

We believe that this specific issue will not be seen in practice, if more elaborate abstractions are used. In fact, we argue that the loop-like behavior is desirable, as it allows for repeatedly applying an action that incrementally brings the concrete state closer to satisfying an important condition. When that condition is satisfied, the abstract state should also change. As an example, consider a simple problem in which there is only one numeric variable that represents some resource that can be produced or consumed. The goal can only be reached by applying an action that consumes a large amount of the resource, but the resource can only be produced in small quantities. If we assume the abstraction can distinguish between states in which the consuming action can be applied and those in which it cannot, we will obtain a policy that recommends applying the production action, followed by the consumption action. Applied directly, the policy would fail, but ARGUS will repeatedly apply the production action until transitioning to a state in which consumption is possible.

Furthermore, the use of a standard best-first search algorithm as a base for ARGUS ensures that the algorithm can successfully recover from reaching a dead-end. Indeed, maintaining an OPEN list of reachable but unexpanded nodes guarantees that, before reaching the goal, we will always be able to expand some node. This means that the only reason why the algorithm can be incomplete is the aforementioned case in which it attempts to completely explore some infinite region that does not contain the goal. That said, implementing a solution for this particular problem can be done by defining a limit on the number or depth of the extra expansions, although we have not experimented with this.

4 Abstraction for Numeric Domains

In this section we describe the specific approach we take when constructing abstractions for numeric planning problems. The general idea is to consider the set of all numeric conditions present in the problem description and to introduce new classical variables to act as proxies for these conditions.

Algorithm 4 describes this process. Initially (lines 1–5) we build the set of all numeric conditions present in the problem and we copy the set of classical variables, the classical initial state, and the classical goal. The loop in lines 6–9 creates new variables that will act as proxies for the corresponding

Algorithm 4: Abstraction of Numeric Planning Problem into Classical Planning Problem

1 **Algorithm** `Abstract` (Π)
  Input: $\Pi = \langle \mathcal{V}, \mathcal{N}, s_0, n_0, s_\star, n_\star, \mathcal{O} \rangle$, a numeric planning problem
  Output: Π^α, an abstracted version of Π, without numeric variables
2 NumConds $\leftarrow n_\star \cup \bigcup_{o \in \mathcal{O}} pre_o^N$
3 $\mathcal{V}^\alpha \leftarrow \mathcal{V}$
4 $s_0^\alpha \leftarrow s_0$
5 $s_\star^\alpha \leftarrow s_\star$
6 **for** $c \in$ NumConds **do**
7 Create a new variable v_c with domain $\mathcal{D}_v = \{\texttt{true}, \texttt{false}\}$
8 $\mathcal{V}^\alpha = \mathcal{V}^\alpha \cup \{v_c\}$
9 Update s_0^α so that $s_0^\alpha(v_c) = n_0(c)$
10 **for** $c \in n_\star$ **do**
11 Update $s_\star^\alpha$ so that $s_\star^\alpha(v_c) = n_\star(c)$
12 $\mathcal{O}^\alpha = \varnothing$
13 **for** $o \in \mathcal{O}$ **do**
14 $pre \leftarrow pre_o$; $eff \leftarrow eff_o$
15 $may^+ \leftarrow \varnothing$; $may^- \leftarrow \varnothing$
16 **for** $c \in pre_o^N$ **do**
17 Update pre so that $pre(v_c) = \texttt{true}$
18 **for** $c \in$ NumConds **do**
19 Update eff so that $eff(v_c) = \delta_\Pi(pre_o^N, o)(c)$
20 **if** $eff(v_c)$ *is still undefined* **then**
21 **if** *there exists a numeric state* n *such that* $n \models pre_o^N$, $n(c) = \texttt{false}$ *and* $\delta_\Pi(n, o)(c) = \texttt{true}$ **then**
22 $may^+ \leftarrow may^+ \cup v_c$
23 **if** *there exists a numeric state* n *such that* $n \models pre_o^N$, $n(c) = \texttt{true}$ *and* $\delta_\Pi(n, o)(c) = \texttt{false}$ **then**
24 $may^- \leftarrow may^- \cup v_c$
25 **for** $add \in 2^{may^+}$ **do**
26 $eff' \leftarrow eff$
27 Update eff' so that $eff'(v_c) = \texttt{true}$ for every $v_c \in add$
28 **for** $del \in 2^{may^-}$ **do**
29 Update eff' so that $eff'(v_c) = \texttt{false}$ for every $v_c \in del \setminus add$
30 $\mathcal{O}^\alpha = \mathcal{O}^\alpha \cup \{\langle l_o, pre, eff' \rangle\}$
31 **return** $\Pi^\alpha = \langle \mathcal{V}^\alpha, s_0^\alpha, s_\star^\alpha, \mathcal{O}^\alpha \rangle$

numeric conditions. The initial state is updated to reflect the correct values the proxies should have (line 9). The loop in lines 10–11 updates the goal condition to consider all the numeric conditions present in the numeric goal.

After that, the loop in lines 13–30 builds the set of abstract operators. This is accomplished by building a set of abstract operators for each original operator. All of the abstract operators that correspond to a given concrete operator share the same preconditions, that correspond to the preconditions of the original operator augmented with the proxies for the relevant numeric conditions (lines 16–17). The algorithm proceeds by considering every numeric condition and testing whether the numeric effects of the operator will necessarily make them `true` or `false` (line 19). If not, then it continues to test whether or not there exist states in which the operator is applicable and where the numeric effects of the operator would change the result of evaluating the numeric condition (in lines 21–24). Here, the auxiliary variables may^+ and may^- are used to store the sets of all the proxy variables whose truth value can be changed from `true` to `false` or `false` to `true`, respectively. The nested loops in lines 25–30 use the information gathered here to produce the set of all combinations of possible effects. Each such combination corresponds to a new abstract operator. Note, however, that two abstract operators created out of the same concrete operator will have the same label.

The tests described in lines 21 and 23 can be implemented with a solver that can handle the numeric conditions and effects, such as an SMT solver or a theorem prover. Specifically, the tests require finding whether or not there exists a pair of assignments to the numeric variables such that the first satisfies the numeric preconditions of o, the second is consistent with what results from applying the numeric effects of o over the first assignment, and the truth value of the numeric condition c changes from one assignment to the other.

The output of Algorithm 4 is a classical planning problem that corresponds to an abstracted version of the numeric planning problem given as input. Indeed, given a state $\langle s, n \rangle$ from the problem $\Pi = \langle \mathcal{V}, \mathcal{N}, s_0, n_0, s_\star, n_\star, \mathcal{O} \rangle$, we can find its corresponding abstract state $\alpha(\langle s, n \rangle)$ by defining:

- $\alpha(\langle s, n \rangle)(v) = s(v)$ for every $v \in \mathcal{V}$.
- $\alpha(\langle s, n \rangle)(v_c) = n(c)$ for every numeric condition $c \in n_\star \cup \bigcup_{o \in \mathcal{O}} pre_o$.

The following theorem establishes the correctness of the abstraction. Every transition possible in the concrete problem has at least one corresponding transition in the abstract space. Furthermore, the operators for corresponding transitions share the same labels.

Theorem 1. *As defined above, α is a state-aggregation abstraction. Given any pair of states S, S' and operator $o \in \mathcal{O}$ such that $\delta_\Pi(S, o) = S'$, there exists some abstract operator $o^\alpha \in \mathcal{O}^\alpha$ such that $\delta_\Pi(\alpha(S), o^\alpha) = \alpha(S')$.*

Proof. Take any $S = \langle s, n \rangle$, $S' = \langle s', n' \rangle$ and o such that $\delta_\Pi(S, o) = S'$. Let $\mathcal{O}^o$ be the set of operators in $\mathcal{O}^\alpha$ that have label l_o. Since $2^{may^+} \neq \varnothing$ and $2^{may^-} \neq \varnothing$, we know there is at least one $o^\alpha \in \mathcal{O}^o$. We also know all such operators affect all variables in $\mathcal{V}$ in exactly the same way as o. In addition, it is easy to see that they are all applicable in $\alpha(S)$. We must prove that at least one $o^\alpha \in \mathcal{O}^o$ has the correct effect over the proxy variables.

Now, for every $c \in$ NumConds where $\delta_\Pi(pre_o^N, o)(c)$ is defined, we know the execution of line 19 in Algorithm 4 ensures that every $o^\alpha \in \mathcal{O}^o$ is such that $eff_{o^\alpha}(v_c) = \delta_\Pi(pre_o^N, o)(c) = \delta_\Pi(n, o)(c) = n'(c)$. For every other $c \in$ NumConds where $n(c) = \texttt{false}$ and $n'(c) = \texttt{true}$, we know $n \models pre_o^N$, so line 22 is executed for this o and c. The case where $n(c) = \texttt{true}$ and $n'(c) = \texttt{false}$ is

analogous. Since may^+ and may^- necessarily contain all the remaining individual effects required to produce n', we are assured that the correct combination is produced. □

As discussed above, the abstraction process adds one new propositional variable for each distinct numeric condition present in the original domain. As such, the number of variables in the abstract domain is linear in the size of the original domain. However, for each operator in the original domain we add a number of new operators that can be up to exponential in the number of numeric conditions affected by the original operator. Nonetheless, most domains have operators that only affect a few domain variables, and will therefore only affect a few conditions, so in practice the final size of the abstract domain's description is not much bigger than that of the original numeric domain.

5 Experimental Evaluation

We implemented ARGUS as an extension to the Fast Downward planning system [Helmert, 2006]. What follows is a brief summary that outlines some key aspects of the implementation. First, the abstraction process works by taking a lifted numeric planning problem specified in PDDL 2.1 [Fox and Long, 2003] and producing an abstracted version with no numeric variables, conditions, or effects. The tests used to determine the possible effects of operators are evaluated using the Z3 Theorem Prover [De Moura and Bjørner, 2008]. The resulting classical planning problem is passed through the preprocessing stages of Fast Downward. Finally, a modified version of the search component of Fast Downward is used to find the abstract plan and to search in the concrete domain with numeric variables.

The abstract search is performed as a greedy best-first search using the FF heuristic [Hoffmann and Nebel, 2001], as this is a standard approach to finding a solution quickly when solution quality is not a priority. The concrete search algorithm is a weighted $A^\star$ (with $w = 5$) modified with the ARGUS expansion procedure. We use weighted $A^\star$ as a compromise between speed and a guarantee of escaping from infinitely large heuristic depressions. The search uses the exact same heuristic used for the abstract search, which is the FF heuristic as computed over the corresponding abstract states. This heuristic is less informative than the Metric-FF heuristic [Hoffmann, 2003], but can be computed for a more general set of problems.

5.1 Evaluation of the Abstraction Process

In addition to the process described in Algorithm 2, we also implemented a naive abstraction method. In this approach, all numeric information in a problem is completely ignored, which results in a much coarser abstraction than the one used by the main approach. This both allows us to analyze the effects of using different types of abstractions, and gives a fallback method that can be used when the abstraction process is prohibitively expensive. In our experiments, we refer to the version of ARGUS that uses the full abstraction as ARGUS-standard and we refer to the version that uses the naive abstraction as ARGUS-naive. Finally, we also implemented a combined version that attempts to compute the full abstraction, but will use the naive abstraction when computing the full one is impractical. We call this version ARGUS-combo.

5.2 Evaluation of Policy Guided Search

To effectively evaluate the effects of doing policy guided search, we implemented a baseline planner that searches directly in the concrete space. This planner uses the abstraction only to compute the heuristic values, and is then implemented as a weighted $A^\star$ algorithm with the same parameters and heuristic used by our planner.

Finally, we also implemented a search algorithm similar to the repair process we used in ASTER. Here, the policy obtained from the abstract search is followed blindly until an unhandled state is reached. At this point, we do a breadth-first search from the reached state until reaching some state that is handled by the policy. We then repeat the process starting from this state. Note that since the abstraction process is different from the one used in ASTER, the resulting algorithm is not equivalent, and can be used in the wider range of domains ARGUS covers. In our experiments, we refer to this re-implementation of ASTER as ASTER'.

5.3 Results

Experimental results in five standard benchmark domains from the International Planning Competition [Long and Fox, 2003] are displayed in Tables 1 and 2. In addition, we include reformulated versions of some of the domains (Depots-NL and Rover-NL) where all (linear) numeric constraints have been replaced by equivalent quadratic constraints. These domains serve to highlight how our approach is a natural method for adapting existing techniques and heuristics for classical planning to numeric planning with non-linear numeric expressions. As mentioned before, we compare our approach using the abstraction described in Algorithm 4 (ARGUS-standard), the approach using the naive abstraction (ARGUS-naive), a simple combination of both (ARGUS-combo), our implementation of the ASTER repair procedure (ASTER'), and the baseline planner. We also compare with Metric-FF, as this is a well-known standard algorithm for numeric planning that exhibits state-of-the-art performance, and with ENHSP, which implements the $\hat{h}^{add}_{hbd+}$ heuristic described by Scala *et al.* (2016) and can handle more expressive domains than Metric-FF. For our experiments, both planners were configured as satisficing planners, ignoring all the optimization aspects of the benchmark domains.

As mentioned, the combination of ARGUS and the naive approach is simply a fallback scheme in which we first try to use the elaborate abstraction process of Algorithm 4 and use the naive abstraction if that cannot be done. This is useful in problems where the *lifted* numeric conditions depend on object variables that can take a large number of possible ground values. In this case, the number of proxy variables that must be created is too high to attempt verifying if a given action can or will modify the value of each of them. In practice, our implementation always attempts to compute the full abstraction and falls back to the naive approach when the standard process reaches a predefined memory limit of 4GB.

Domain	ARGUS-standard	ARGUS-naive	ARGUS-combo	Metric-FF	ENHSP	ASTER'	Baseline
Depots (22)	19	19	19	20	3	9	11
Driverlog (20)	20	20	20	17	11	20	18
Rovers (20)	11	8	11	10	8	10	10
Settlers (20)	5	1	5	8	–	2	2
Zenotravel (20)	1	18	18	20	15	0	18
Depots-NL (22)	19	19	19	–	3	9	11
Rovers-NL (20)	12	8	12	–	6	6	8
Total (102)	86	92	103	75	46	56	78

Table 1: Number of problems solved by each planner. For each instance, every planner was limited to a 30 minute runtime using at most 4GB of memory. These limits are applied to all preprocessing stages and search. ARGUS-standard and ARGUS-naive correspond to our algorithm using the two types of abstraction described. ARGUS-combo attempts to use the richer abstraction and falls back to the naive one if computing the former requires too much memory. ASTER' is the implementation of the ASTER repair procedure over our abstraction. A dash (–) signifies that the corresponding planner cannot solve problems from that specific domain. The first column shows the domain names and, in parentheses, the total number of problem instances in that domain.

	Metric-FF		ENHSP		Actual Search
Domain	Length	Time	Length	Time	Time
Depots	0.94	0.11	1.00	29.51	0.97
Driverlog	1.04	0.19	1.12	196.14	0.79
Rovers	1.00	0.00	1.06	1.31	0.95
Settlers	0.98	0.011	N/A	N/A	0.95
Zenotravel	0.99	0.0016	1.04	10.34	0.70
Depots-NL	N/A	N/A	1.00	25.51	0.95
Rovers-NL	N/A	N/A	0.98	184.80	0.99
Overall	0.99	0.08	1.05	78.20	0.95

Table 2: Proportional averages of plan length and times to solution when comparing the algorithms to ARGUS-combo. For both length and time, a value close to 1 implies that the specified algorithm has similar performance to ARGUS-combo on the subset of problems that both were able to solve. Values above 1 mean ARGUS-combo has better performance. We also show the proportion of time spent by ARGUS-combo doing actual search, as opposed to computing the abstraction.

6 Conclusions and Future Work

The experimental results show that out approach is competitive when compared to Metric-FF. When considering only the standard benchmarks, we can see that the time needed by Metric-FF to find solutions significantly outperforms ARGUS, although this does not reflect into a significant difference in coverage of solved problems. Moreover, the results for the non-linear reformulations suggest that our approach is an effective method for dealing with problems that require more expressivity. The comparison with ENHSP further confirms this, and also highlights how the use of existing technology for classical planning, such as the highly optimized Fast Downward, can be beneficial for numeric planning. Nonetheless, since our algorithms do not consider any optimization metrics and our abstraction process loses information regarding the numeric dynamics of the problems, we expect ENHSP would outperform ARGUS if optimization was important, or if domains involved a greater number of (complex) numeric constraints, relative to propositional aspects. That said, we believe our approach can be combined with the ideas used in ENHSP and other recent advances for numeric planning, as ARGUS could certainly benefit from using more informed heuristics in the concrete search space.

Indeed, the results also serve to highlight some of the current limitations of our approach. First, Metric-FF severely outperforms ARGUS when measuring runtime over the problems solved by both algorithms. This is due to the fact that the heuristic used by our algorithm is significantly less informative than the Metric-FF heuristic. In addition, we can see that ARGUS is not very effective in some domains, such as Zenotravel. This is because the number of numeric conditions affected by some of the actions in the domain is too high. Since the abstraction process is exponential in that number, it cannot be completed in any reasonable amount of time or without using too much memory.

In the future, we are interested in studying how similar techniques can be applied to other problems with very large search spaces. We believe the approach is a general way of doing a form of refinement over a heuristic by exploiting advances in classical planning. The approach is not limited to numeric planning, as reasonable abstractions can be computed for many other classes of problems. We are particularly interested in exploring how our approach can be applied for generalized planning problems, where the goal is to find a plan or policy that can solve multiple problem instances [Levesque, 2005; Hu and De Giacomo, 2011]. In this context, the notion of having plans that loop is clearly useful, and we believe our approach can be applied to this specific objective and can be related to existing literature in that field (e.g., Srivastava *et al.*; Srivastava *et al.* (2011; 2012)).

Acknowledgements

The authors gratefully acknowledge funding from the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

[Bäckström and Nebel, 1995] Christer Bäckström and Bernhard Nebel. Complexity results for SAS+ planning. *Computational Intelligence*, 11:625–656, 1995.

[Bonet and Geffner, 2001] Blai Bonet and Hector Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1-2):5–33, 2001.

[Chrpa *et al.*, 2015] Lukáš Chrpa, Enrico Scala, and Mauro Vallati. Towards a reformulation based approach for efficient nu-

meric planning: Numeric outer entanglements. In *Proceedings of the 8th Symposium on Combinatorial Search (SoCS)*, pages 166–170, 2015.

[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 Scheduling (ICAPS)*, pages 52–59, 2008.

[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 (POPL)*, pages 238–252, 1977.

[De Moura and Bjørner, 2008] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In *Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 337–340, 2008.

[Edelkamp, 2001] Stefan Edelkamp. Planning with pattern databases. In *Sixth European Conference on Planning (ECP)*, pages 13–24, 2001.

[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.

[Fritz and McIlraith, 2007] Christian Fritz and Sheila A. McIlraith. Monitoring plan optimality during execution. In *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 144–151, 2007.

[Graf and Saïdi, 1997] Susanne Graf and Hassen Saïdi. Construction of abstract state graphs with PVS. In *Proceedings of the 9th International Conference on Computer Aided Verification (CAV)*, pages 72–83, 1997.

[Haslum and Geffner, 2000] Patrik Haslum and Hector Geffner. Admissible heuristics for optimal planning. In *Proceedings of the 5th International Conference on Artificial Intelligence Planning Systems (AIPS)*, pages 140–149, 2000.

[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*, 61(3):16, 2014.

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

[Hu and De Giacomo, 2011] Yuxiao Hu and Giuseppe De Giacomo. Generalized planning: Synthesizing plans that work for multiple environments. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI)*, pages 918–923, 2011.

[Illanes and McIlraith, 2016a] León Illanes and Sheila A. McIlraith. Numeric planning via search space abstraction. In *Proceedings of the 9th Symposium on Combinatorial Search (SoCS)*, pages 133–134, 2016.

[Illanes and McIlraith, 2016b] León Illanes and Sheila A. McIlraith. Numeric planning via search space abstraction. In *Proceedings of the Workshop on Knowledge-based Techniques for Problem Solving and Reasoning (KnowProS)*, 2016.

[Knoblock, 1994] Craig A. Knoblock. Automatically generating abstractions for planning. *Artificial intelligence*, 68(2):243–302, 1994.

[Levesque, 2005] Hector J. Levesque. Planning with loops. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 509–515, 2005.

[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 Scheduling (ICAPS)*, pages 172–180, 2012.

[Sacerdoti, 1974] Earl Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5:115–135, 1974.

[Scala *et al.*, 2016] Enrico Scala, Patrik Haslum, and Sylvie Thiébaux. Heuristics for numeric planning via subgoaling. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 3228–3234, 2016.

[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 Scheduling (ICAPS)*, pages 347–351, 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 Scheduling (ICAPS)*, pages 289–297, 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)*, pages 105–111, 2012.

[Srivastava *et al.*, 2011] Siddharth Srivastava, Shlomo Zilberstein, Neil Immerman, and Hector Geffner. Qualitative numeric planning. In *Proceedings of the 25th AAAI Conference on Artificial Intelligence (AAAI)*, pages 1010–1016, 2011.

[Srivastava *et al.*, 2012] Siddharth Srivastava, Neil Immerman, and Shlomo Zilberstein. Applicability conditions for plans with loops: Computability results and algorithms. *Artificial Intelligence*, 191-192:1–19, 2012.

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