Research Workshop of the
Israel Science Foundation

Proceedings of the 7th Workshop on

Heuristics and Search for Domain-independent Planning (HSDIP)

Edited By:

Ron Alford, J. Benton, Erez Karpas, Michael Katz, Nir Lipovetzky, Gabriele Röger and Jordan Thayer

Jerusalem, Israel 8/6/2015

Organizing Committee

Ron Alford
U.S. Naval Research Lab, USA

J. Benton
SIFT, USA

Erez Karpas
Massachusetts Institute of Technology, USA

Michael Katz
IBM Research, Israel

Nir Lipovetzky
University of Melbourne, Australia

Gabriele Röger
University of Basel, Switzerland

Jordan Thayer
SIFT, USA

Table of Contents

Complexity Issues of Interval Relaxed Numeric Planning

Johannes Aldinger and **Robert Mattmüller** and **Moritz Göbelbecker**
Albert-Ludwigs-Universität, Institut für Informatik
79110 Freiburg, Germany
{aldinger,mattmuel,goebelbe}@informatik.uni-freiburg.de

Abstract

Automated planning is a hard problem even in its most basic form as STRIPS planning. We are interested in numeric planning tasks with instantaneous actions, a problem which is not even decidable in general. Relaxation is an approach to simplifying complex problems in order to obtain guidance in the original problem. We present a relaxation approach with intervals for numeric planning and discuss the arising complexity issues.

Introduction

Relaxation is a predominant approach to simplifying planning problems. Solutions of the relaxed planning problem can be used to guide search in the original planning task. The forward propagation heuristic h_{add} (Bonet, Loerincs, and Geffner 1997; Bonet and Geffner 2001) was used in the heuristic search planner that won the first International Planning Competition (IPC 1998) and $h_{\max}$ (Bonet and Geffner 1999) is its admissible counterpart. The underlying assumption of a delete relaxation is that propositions which are achieved once during planning can not be invalidated. More recent planning systems are usually not restricted to propositional state variables of the planning problem. Instead they use the SAS^+ formalism (Bäckström and Nebel 1993) which allows for (finite-domain) multi-valued variables. Unlike propositional STRIPS (Fikes and Nilsson 1971), a "delete relaxation" corresponds to variables that can attain a *set* of values at the same time. Extending this concept for numeric planning relaxes the set representation even further. Numeric variables can have infinitely many values which makes it impossible to store all of them. A memory efficient approach is to consider the enclosing interval of all possible values for each numeric variable. The methods to deal with intervals have been subject to the field of interval arithmetic (Young 1931) which has been used in mathematics for decades (Moore, Kearfott, and Cloud 2009) and enables us to deal with intervals in terms of basic numeric operations.

Numeric planning tasks can require actions to be applied multiple times, as setting a numeric variable to a target value can require multiple steps even in relaxed problems. In this paper we provide the foundations for interval relaxed numeric planning.

Related Work

Extending the concept of classical planning heuristics to numeric problems has been done before, albeit only for a subset of numeric tasks. In many relevant real world problems, numeric variables can only be manipulated in a restricted way. The Metric-FF planning system (Hoffmann 2003) tries to convert the planning task into a *linear numeric* task which ensures that variables can "grow" in only one direction. By introducing inverted auxiliary variables for decreasing numeric variables, the concept of a delete relaxation translates into a relaxation where *decrease* effects are considered harmful and higher values of a variable are always beneficial to fulfill the preconditions of actions.

More recently, Coles et al. (2008) investigated an approach based on linear programs. In many relevant real world applications, numeric variables are used to model resources. Delete relaxation heuristics fail to offer guidance on such problems if a *cyclic resource transfer* is possible. As delete relaxations make the assumption that sub-goals stay achieved, a resource transfer can "produce" resources without decreasing them at their original destination. Coles et al. analyze the planning problem for consumers and producers of resources and build a linear program to ensure that resources are not more often consumed than produced or initially available to obtain an informative heuristic.

Basics

In this section we outline numeric planning with instantaneous actions which is expressible in PDDL2.1, layer 2 (Fox and Long 2003). We present an overview over interval arithmetic, the technique we use to extend delete relaxation heuristics to numeric planning. The section closes with a short complexity discussion.

Numeric Planning with Instantaneous Actions

Given a set of variables $\mathcal{V}$ with domains $\text{dom}(v)$ for all $v \in \mathcal{V}$, a *state* s is a mapping of variables v to their respective domains. Throughout the paper, we denote the value of a variable v in a state s by $s(v)$.

A numeric planning task $\Pi = \langle \mathcal{V}_P, \mathcal{V}_N, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$ is a 5-tuple where $\mathcal{V}_P$ is a set of propositional variables $\mathsf{v_p}$ with domain $\{\mathit{true}, \mathit{false}\}$. $\mathcal{V}_N$ is a set of numeric variables $\mathsf{v_n}$ with domain $\mathbb{Q}^\infty$ where we abbreviate $\mathbb{Q}^\infty$ for

$\mathbb{Q} \cup \{-\infty, \infty\}$ throughout the paper. $\mathcal{O}$ is a set of operators, $\mathcal{I}$ the initial state and $\mathcal{G}$ is the goal condition. A *numeric expression* $\mathsf{e}_1 \circ \mathsf{e}_2$ is an arithmetic expression with operators $\circ \in \{+, -, \times, \div\}$ and expressions e_1 and e_2 recursively defined over variables $\mathcal{V}_N$ and constants from $\mathbb{Q}$. A *numeric constraint* $\mathrm{con} = (\mathsf{e}_1 \bowtie \mathsf{e}_2)$ compares numeric expressions e_1 and e_2 with $\bowtie \in \{\leq, <, =, \neq\}$. A *condition* is a conjunction of propositions and numeric constraints. A *numeric effect* is a triple $(\mathsf{v_n} \circ\!= \mathsf{e})$ where $\mathsf{v_n} \in \mathcal{V}_N$, $\circ\!= \in \{:=, +=, -=, \times=, \div=\}$ and e is a numeric expression. Operators $o \in \mathcal{O}$ are of the form $\langle \mathrm{pre} \rightarrow \mathrm{eff} \rangle$ and consist of a condition pre and a set of effects eff which contains at most one numeric effect for each numeric variable $\mathsf{v_n}$ and at most one truth assignment for each propositional variable $\mathsf{v_p}$.

The semantic of a numeric planning task is straightforward. For constants $c \in \mathbb{Q}, s(c) = c$ by abuse of notation. Numeric expressions $(\mathsf{e}_1 \circ \mathsf{e}_2)$ for $\circ \in \{+, -, \times, \div\}$ are recursively evaluated in state s: $s(\mathsf{e}_1 \circ \mathsf{e}_2) = s(\mathsf{e}_1) \circ s(\mathsf{e}_2)$. A state satisfies a condition $s \vDash \mathsf{v_p}$ iff $s(\mathsf{v_p}) = \mathit{true}$, where $\mathsf{v_p} \in \mathcal{V}_P$. For numeric constraints, $s \vDash (\mathsf{e}_1 \bowtie \mathsf{e}_2)$ iff $s(\mathsf{e}_1) \bowtie s(\mathsf{e}_2)$ where $\bowtie \in \{\leq, <, =, \neq\}$, and e_1 and e_2 are expressions. A state satisfies a conjunctive condition $s \vDash k_1 \wedge k_2$ iff $s \vDash k_1$ and $s \vDash k_2$.

An operator $o = \langle \mathrm{pre} \rightarrow \mathrm{eff} \rangle$ is applicable in s iff $s \vDash \mathrm{pre}$. The successor state $\mathrm{app}_o(s) = s'$ resulting from an application of o is defined as follows, where $\mathrm{eff} = \{\mathrm{eff}_1, \ldots, \mathrm{eff}_n\}$: if eff_i is a numeric effect $\mathsf{v_n} \circ\!= \mathsf{e}$ with $\circ\!= \in \{+=, -=, \times=, \div=\}$, $s'(\mathsf{v_n}) = s(\mathsf{v_n}) \circ s(\mathsf{e})$. If eff_i is a numeric effect $\mathsf{v_n} := \mathsf{e}$, then $s'(\mathsf{v_n}) = s(\mathsf{e})$. If eff_i is a propositional effect $\mathsf{v_p} := \mathsf{e_p}$ with $\mathsf{e_p} \in \{\mathit{true}, \mathit{false}\}$, $s'(\mathsf{v_p})$ is the new truth value $\mathsf{e_p}$. Finally, if a variable v does not occur in any effect, then $s'(\mathsf{v}) = s(\mathsf{v})$.

A plan π is a sequence of actions that leads from $\mathcal{I}$ to a state satisfying $\mathcal{G}$ such that each action is applicable in the state that follows by executing the plan up to that action.

We intend to relax numeric planning with the help of intervals. The next section establishes the foundations of interval arithmetic.

Interval Arithmetic

Interval arithmetic uses an upper and a lower bound to enclose the actual value of a number. Closed intervals $[\underline{x}, \overline{x}] = \{q \in \mathbb{Q}^\infty \mid \underline{x} \leq q \leq \overline{x}\}$ contain all rational numbers (or $\pm\infty$) from $\underline{x}$ to $\overline{x}$. Throughout this paper we refer to the lower bound of an interval x by $\underline{x}$ and to the upper bound by $\overline{x}$. The set $\mathbb{I}_c = \{[\underline{x}, \overline{x}] \mid \underline{x} \leq \overline{x}\}$ contains all closed intervals. Numbers q can be transformed into a degenerate interval $[q, q]$. The basic arithmetic operations in interval arithmetic are given as:

- addition: $[\underline{x}, \overline{x}] + [\underline{y}, \overline{y}] = [\underline{x} + \underline{y}, \overline{x} + \overline{y}]$,
- subtraction: $[\underline{x}, \overline{x}] - [\underline{y}, \overline{y}] = [\underline{x} - \overline{y}, \overline{x} - \underline{y}]$,
- multiplication: $[\underline{x}, \overline{x}] \times [\underline{y}, \overline{y}] = \left[\min(\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y}), \max(\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y})\right]$,
- division: $[\underline{x}, \overline{x}] \div [\underline{y}, \overline{y}] = \left[\min\left(\underline{x}/\underline{y}, \underline{x}/\overline{y}, \overline{x}/\underline{y}, \overline{x}/\overline{y}\right), \max\left(\underline{x}/\underline{y}, \underline{x}/\overline{y}, \overline{x}/\underline{y}, \overline{x}/\overline{y}\right)\right]$ if $0 \notin [\underline{y}, \overline{y}]$. Otherwise, at least one of the bounds diverges to $\pm\infty$. We do not explicate all cases of $\underline{x}, \overline{x}, \underline{y}$ and $\overline{y}$ being positive, negative or zero which determine which of the bounds diverge and refer the interested reader to the literature (Moore, Kearfott, and Cloud 2009).

Analogously we define open bounded intervals $(\underline{x}, \overline{x}) = \{q \in \mathbb{Q}^\infty \mid \underline{x} < q < \overline{x}\}$ and the set of open intervals $\mathbb{I}_o = \{(\underline{x}, \overline{x}) \mid \underline{x} < \overline{x}\}$, as well as half open intervals $[\underline{x}, \overline{x}) = \{q \in \mathbb{Q}^\infty \mid \underline{x} \leq q < \overline{x}\}$ and $(\underline{x}, \overline{x}] = \{q \in \mathbb{Q}^\infty \mid \underline{x} < q \leq \overline{x}\}$ and the respective sets $\mathbb{I}_{co} = \{[\underline{x}, \overline{x}) \mid \underline{x} < \overline{x}\}$ and $\mathbb{I}_{oc} = \{(\underline{x}, \overline{x}] \mid \underline{x} < \overline{x}\}$. Finally the set of mixed bounded intervals is given as $\mathbb{I}_m = \mathbb{I}_c \cup \mathbb{I}_o \cup \mathbb{I}_{oc} \cup \mathbb{I}_{co}$. Open and mixed bounded intervals follow the same arithmetic rules as closed intervals. Whenever open and closed bounds contribute to the new interval bound, the bound is open.

Example 1. The product $(-2, 3] \times [-4, 2)$ is the interval $[-12, 8)$. The lower bound is the result of 3×-4 and the resulting bound is closed because both contributing bounds are closed. The new upper bounds is computed by -2×-4 and the open bound of the left interval determines the "openness" of the resulting bound.

Definition 1. Let $x, y \in \mathbb{I}_m$ be intervals. The convex union $u = x \sqcup y$ is the interval with $\underline{u} = \min(\underline{x}, \underline{y})$ and $\overline{u} = \max(\overline{x}, \overline{y})$. Whether the bounds of u are open or closed depends on whether those of x and y are open or closed.

Definition 1 implicitly adds all values between the intervals to the resulting interval if $x \cap y = \emptyset$.

Complexity

Unlike classical planning, which is PSPACE-complete (Bylander 1994), numeric planning is *undecidable* (Helmert 2002). Even though completeness of numeric planning can therefore not be achieved in general, numeric planners can find plans or an assurance that the problem is unsolvable for many practical problems. Moreover, we will prove in the following section that the relaxed numeric plan existence problem is decidable in polynomial time for *acyclic dependency* tasks, tasks in which the expressions of numeric effects do not depend on the variable they alter.

Delete Relaxation

In this section we discuss natural extensions of delete relaxation to planning with numeric variables.

Motivation

As planning is hard, it is beneficial to consider a simplified problem in order to obtain guidance in the original problem. The delete relaxation of classical planning ignores delete effects, effects that set the truth value of a proposition to *false*. As action preconditions and the goal condition require propositions to evaluate to *true*, delete effects complicate plan search. Finding a relaxed plan on the other hand is possible in polynomial time because relaxed actions do not have delete effects and therefore each action has to be applied at most once. Plans for the original problem are also plans for

the corresponding delete relaxed planning task. While finding *any* relaxed plan is possible in polynomial time, finding a shortest relaxed plan is NP-hard (Bylander 1994).

Numeric Relaxation Approaches

The idea behind delete relaxation is that facts that are reached once stay achieved. We will now discuss several ways to extend this concept to numeric planning. Combinations of these approaches are subject of future research.

Enumeration. The number of values that a variable can attain after applying a fixed number of actions is finite. An idea is to store the set of all attained values for each variable. However, the number of attainable values grows exponentially with the number of applied operators. As such it becomes infeasible to maintain the set of possible values quickly.

Example 2. Consider a numeric planning task with initial state $\mathcal{I}(\mathsf{x}) = 0$ and operators $o_1 = \langle \emptyset \to \{\mathsf{x} \mathrel{+}= 1\}\rangle$ and $o_2 = \langle \emptyset \to \{\mathsf{x} \mathrel{\div}= 2\}\rangle$. Denoting by x_k, $k = 0, \ldots, 3$ the set of possible values of x after k steps, we have: $x_0 = \{0\}$, $x_1 = \{0, 1\}$, $x_2 = \{0, \frac{1}{2}, 1, 2\}$ and $x_3 = \{0, \frac{1}{4}, \frac{1}{2}, 1, 1\frac{1}{2}, 2, 3\}$.

For problems with bounded plan length, the enumeration approach requires space exponential in the bound. Even worse, the enumeration relaxation remains undecidable in general.

Theorem 1. *The numeric plan existence problem in an enumeration relaxation is undecidable.*

Proof sketch. We can basically adopt the proof for numeric planning by Helmert (2002). Formalizing Diophantine equations as planning problem results in a task that is not decidable as solutions have to be integers and the relaxation does not relax this property. ∎

Discretization. In order to restrict the number of possible values from the enumeration approach, multiple values can be aggregated into "buckets", where a representative approximates all values within. These representatives can be treated as multi-valued finite domain variables from classical SAS$^+$-planning. The state transition has to be defined in such a way that completeness is preserved – plans for the real problem have to act as plans in the relaxed problem. Proper abstractions offer potential for future research.

Higher values are better. Another approach is only feasible on a restricted set of planning tasks. If all preconditions and goals have the form $(\mathsf{x} > c)$ or $(\mathsf{x} \geq c)$ where x is a numeric variable and c a numeric constant, higher values are always beneficial for a variable. Numeric effects are only allowed to alter numeric variables by a positive constant, and therefore, decrease effects are considered harmful. The Metric-FF planning system uses this type of relaxation and Hoffmann (2003) shows that a large class of problems can be compiled into the required linear normal form.

Interval relaxation. An interval which encloses all values that a numeric variable can attain is a memory efficient method. Algebraic base operations are allowed in PDDL and supported by interval arithmetic. Therefore, we will focus on interval relaxation in the following section.

Interval Relaxation

In this section we elaborate on interval relaxation for numeric planning tasks. We will discuss the complexity of the plan existence problem for the presented semantics. We identify a class of tasks with *acyclic dependencies* between variables for which we can generate interval relaxed plans in polynomial time.

The interval relaxation of a numeric planning task differs only marginally from the original task description on a syntactic level. Propositional variables can now be both true and false at the same time and numeric variables are mapped to closed intervals.

Definition 2. Let Π be a numeric planning task. The interval delete relaxation $\Pi^+ = \langle \mathcal{V}_P^+, \mathcal{V}_N^+, \mathcal{O}^+, \mathcal{I}^+, \mathcal{G}^+ \rangle$ of Π is a 5-tuple where $\mathcal{V}_P^+$ are the propositional variables from Π with the domains replaced by $\mathrm{dom}(\mathsf{v_p}) = \{\mathit{true}, \mathit{false}, \mathit{both}\}$ and $\mathcal{V}_N^+$ are the numeric variables with the domains replaced by closed intervals $\mathrm{dom}(\mathsf{v_n}) = \mathbb{I}_c$ for all $\mathsf{v_n} \in \mathcal{V}_N^+$. The initial state $\mathcal{I}^+$ is derived from $\mathcal{I}$ by replacing numbers $\mathcal{I}(\mathsf{v_n})$ with degenerate intervals $\mathcal{I}^+(\mathsf{v_n}) = [\mathcal{I}(\mathsf{v_n}), \mathcal{I}(\mathsf{v_n})]$ and $\mathcal{I}^+(\mathsf{v_p}) = \mathcal{I}(\mathsf{v_p})$. $\mathcal{G}^+$ is the goal condition.

The semantic of Π^+ draws on interval arithmetic. *Numeric expressions* are defined recursively: let e_1 and e_2 be numeric expressions. The interpretation of a constant expression is $s^+(c) = [c, c]$ and compound expressions are interpreted as $s^+(\mathsf{e}_1 \circ \mathsf{e}_2) = s^+(\mathsf{e}_1) \circ s^+(\mathsf{e}_2)$ for $\circ \in \{+, -, \times, \div\}$ where "$\circ$" now operates on intervals. For (goal and operator) conditions, the relaxed semantic is defined as follows: let $\mathsf{v_p} \in \mathcal{V}_P^+$ be a propositional variable, then $s^+ \vDash \mathsf{v_p}$ iff $s^+(\mathsf{v_p}) \in \{\mathit{true}, \mathit{both}\}$. For numeric constraints let e_1 and e_2 be numeric expressions, and $\bowtie \in \{<, \leq, =, \neq\}$ a comparison operator. Then $s^+ \vDash (\mathsf{e}_1 \bowtie \mathsf{e}_2)$ iff $\exists q_1 \in s^+(\mathsf{e}_1), \exists q_2 \in s^+(\mathsf{e}_2)$ with $q_1 \bowtie q_2$. This implies that two intervals can be "greater" and "less" than each other at the same time.

The semantic of numeric effects $\mathsf{v_n} \circ\!= \mathsf{e}$ is relaxed twice: $\mathsf{v_n}$ keeps its old value and gains all values up to the new value which is an interval in the relaxation. The state $\mathrm{app}_o^+(s^+) = s'^+$ resulting from an application of o with effect $\mathrm{eff} \in \{\mathrm{eff}_1, \ldots, \mathrm{eff}_n\}$ is then $s'^+(\mathsf{v_n}) = s^+(\mathsf{v_n}) \sqcup (s^+(\mathsf{v_n}) \circ s^+(\mathsf{e}))$ if eff is a numeric effect. As we use the convex union from Definition 1, $s'^+(\mathsf{v_n})$ contains all values between the old value of $\mathsf{v_n}$ and the evaluated expression $(s^+(\mathsf{v_n}) \circ s^+(\mathsf{e}))$. For propositional effects, $s'^+(\mathsf{v_p}) = \mathit{both}$ if the effect changes the truth value $\mathrm{eff}_i(\mathsf{v_p}) \neq s^+(\mathsf{v_p})$ of $\mathsf{v_p}$, and $s'^+(\mathsf{v_p}) = s^+(\mathsf{v_p})$ otherwise. Again, $s'^+(\mathsf{v}) = s^+(\mathsf{v})$ if v occurs in no effect.

Example 3. Applying $o = \langle \emptyset \to \{\mathsf{x} \times\!= \mathsf{e}\}\rangle$ in a state mapping $\mathsf{x} \mapsto [8, 10]$ and $\mathsf{e} \mapsto [-\frac{1}{2}, \frac{1}{2}]$ leads to a state $s'(\mathsf{x}) = [8, 10] \sqcup ([8, 10] \times [-\frac{1}{2}, \frac{1}{2}]) = [8, 10] \sqcup [-5, 5] = [-5, 10]$.

Interval Relaxation Complexity

For the classical relaxed planning problem, a relaxed plan can be found by applying all applicable operators in parallel until a fix-point is reached. As no effect can destroy a condition in the relaxed task, the number of operators in the planning task restricts the required number of iterations until a fix-point is reached. The task is solvable if the goal condition holds in the resulting state. A serialized plan can be obtained by ordering actions from the same parallel layer arbitrarily.

We employ a similar method for interval relaxed numeric planning. We have to approach the challenge that numeric operators can have to be applied arbitrarily often. An idea is to transform the planning task into a semi-symbolic representation which captures repeated application of operators with numeric effects. We define interval relaxed and *repetition relaxed* planning tasks which we refer to as *repetition relaxed* for short. In repetition relaxed planning tasks we simulate the behavior of applying numeric effects arbitrarily often *independently*. As we will see later, the independence assumption is not justified for numeric effects $\mathsf{v_n} \circ\!= \mathsf{e}$ where the expression of the assignment e depends on the affected variable $\mathsf{v_n}$. We show that an adaptation of the algorithm from classical relaxed planning can be used to find plans for repetition relaxed planning tasks with *acyclic dependencies*, where the variables in e do not depend on $\mathsf{v_n}$.

Repetition relaxed planning tasks use mixed bounded intervals, intervals whose bounds can either be open or closed, to capture the attainable values of a numeric variable. We are interested in the behavior of numeric effects in the limit. We use different fonts to distinguish a variable and its value e.g. $s(\mathsf{x}) = x$ in the following, whenever the state s is not essential. If an operator o has an additive effect $\mathsf{x} \pm\!= \mathsf{e}$ for $\pm\!= \in \{+\!=, -\!=\}$ which can extend a bound of x once, it can extend that bound to any value by applying o multiple times. The result of applying an additive effect arbitrarily often in a state s only depends on whether e can be negative, zero or positive. The behavior of multiplicative effects $*\!= \in \{\times\!=, \div\!=\}$ is slightly more complex. Multiplicative effects $\mathsf{x} *\!= \mathsf{e}$ can contract or expand depending on whether e contains elements with absolute value greater one and switch signs if e negative which results in up to seven different behaviors of e.

Definition 3. Let Π^+ be an interval relaxed planning task. An (interval and) repetition relaxed planning task of Π^+ is a 5-tuple $\Pi^\# = \langle V_P^+, \mathcal{V}_N^\#, \mathcal{O}^\#, \mathcal{I}^\#, \mathcal{G}^\# \rangle$ with propositional variables V_P^+ from Π^+. The domains of numeric variables $\mathrm{dom}(\mathsf{v_n}) = \mathbb{I}_m$ for $\mathsf{v_n} \in \mathcal{V}_N^\#$ are extended to mixed-bounded intervals. The initial state $\mathcal{I}^\#(\mathsf{v_p}) = \mathcal{I}^+(\mathsf{v_p})$ assigns the same truth value from $\mathcal{I}^+$ to each propositional variable $\mathsf{v_p}$ and each numeric variable $\mathsf{v_n}$ is initialized to the same closed degenerate interval $\mathcal{I}^\#(\mathsf{v_n}) = \mathcal{I}^+(\mathsf{v_n})$.

Again, the relaxation does not change much on a syntactical level. The main difference lies in the semantic of numeric effects. The semantic of *numeric expressions* can be transferred directly from the interval relaxation as interval arithmetic operations are also defined for mixed bounded intervals. The interpretation of a numeric expression is given as $s^\#(\mathsf{e_1} \circ \mathsf{e_2}) = s^\#(\mathsf{e_1}) \circ s^\#(\mathsf{e_2})$ for expressions $\mathsf{e_1}$ and $\mathsf{e_2}$ and $\circ \in \{+, -, \times, \div\}$. The semantic of conditions is again $s^\# \vDash \mathsf{v_p}$ iff $s^\#(\mathsf{v_p}) \in \{\mathit{true}, \mathit{both}\}$ for propositions $\mathsf{v_p} \in \mathcal{V}_P^\#$. For numeric constraints $\mathsf{e_1} \bowtie \mathsf{e_2}$ where $\mathsf{e_1}$ and $\mathsf{e_2}$ are expressions and comparison operator $\bowtie \in \{<, \leq, =, \neq\}$, $s^\# \vDash (\mathsf{e_1} \bowtie \mathsf{e_2})$ iff $\exists q_1 \in s^\#(\mathsf{e_1}), \exists q_2 \in s^\#(\mathsf{e_2})$ with $q_1 \bowtie q_2$.

The semantic of *numeric effects* captures the repeated application of actions. We first define the *repetition relaxed* semantic of $x \circ\!= e$ for intervals x and e with $\circ\!= \in \{:=, +\!=, -\!=, \times\!=, \div\!=\}$. Let $x_0 = x$ and $x_{i+1} = x_i \sqcup (x_i \circ e)$ for $i \geq 0$ where $(x : e)$ is defined as e for assign effects. Let $\mathrm{succ}_\circ(x, e) = \bigcup_{i=0}^\infty x_i$. We are interested in the result of applying an operator arbitrarily often individually for each effect, where the interval e is fixed even if the expression e depends on x. As $x_{i+1} \supseteq x_i$ by definition of the convex union and because all x_i are convex, the resulting set $\mathrm{succ}_\circ(x, e)$ is an interval. However, open bounded intervals can be generated by the limit value consideration. The state $\mathrm{app}_o^\#(s^\#) = s'^\#$ resulting from an application of o with effect $\mathrm{eff} = \{\mathrm{eff}_1, \ldots, \mathrm{eff}_n\}$ is then again $s'^\#(\mathsf{v}) = s^\#(\mathsf{v})$ if v occurs in no effect, $s'^\#(\mathsf{v_p}) = \mathit{both}$ if $\mathrm{eff}_i(\mathsf{v_p}) \neq s^\#(\mathsf{v_p})$ is a propositional effect which changes the truth value of $\mathsf{v_p}$ and $s'^\#(\mathsf{v_p}) = s^\#(\mathsf{v_p})$ otherwise. For numeric effects $\mathrm{eff}_i = (\mathsf{v_n} \circ\!= \mathsf{e})$, $s'^\#(\mathsf{v_n}) = \mathrm{succ}_\circ(s^\#(\mathsf{v_n}), s^\#(\mathsf{e}))$.

Fixing expressions e of numeric effects $\mathsf{v_n} \circ\!= \mathsf{e}$ to the interval e they evaluate to in the previous state is beneficial to compute the successor, as changes in the assignment (which can be an arbitrary arithmetic expression) do not have to be considered immediately. The repetition relaxation $\Pi^\#$ of a planning task relaxes Π^+ further and plans for Π^+ are still plans for $\Pi^\#$. The reason is that each operator application can only extend the interval of affected numeric variables more than before. Evaluating the expression in the successor state $s'^\#(\mathsf{e})$ can only extend the interval $s^\#(\mathsf{e})$.

We want to use the fix-point algorithm which applies all operators of a planning task in parallel until a fix-point is reached to find a repetition relaxed plan. The successors $\mathrm{succ}_\circ(x, e)$ of numeric effects are defined by the limit $\bigcup_{i=0}^\infty x_i$ and we are interested in determining the result of such an effect in constant time. The result only depends on which of up to 21 symbolic *behavior classes* are covered by x and e. The seven *behavior classes* for e are $\mathcal{B}_e = \{(-\infty, -1), \{-1\}, (-1, 0), \{0\}, (0, 1), \{1\}, (1, \infty)\}$, and for x they are $\mathcal{B}_x = \{(-\infty, 0), \{0\}, (0, \infty)\}$. We decompose e and x into the hit behavior classes where $e \cap \tilde{e} \neq \emptyset$ for a behavior class $\tilde{e} \in \mathcal{B}_e$ and $x \cap \tilde{x} \neq \emptyset$ for a behavior class $\tilde{x} \in \mathcal{B}_x$, respectively. Table 1 contains partial behaviors $\mathcal{T}_\circ(x, e)$ for $\circ\!= \in \{+\!=, -\!=, \times\!=, \div\!=\}$ where $\mathcal{T}_\circ(x, e)$ is only defined if $x \subseteq \tilde{x} \in \mathcal{B}_x$ and $e \subseteq \tilde{e} \in \mathcal{B}_e$ and $\mathcal{T}_\circ(x, e)$ is the table entry with column $\tilde{x}$ and row $\tilde{e}$ in the table with the corresponding $\circ\!=$ operator. We use "indeterminate" parentheses $(\!|\cdot, \overline{\cdot}|\!)$ to denote intervals whose openness is determined by the terms

$+=$		$\tilde{e}$		
		$(-\infty, 0)$	$\{0\}$	$(0, \infty)$
$\tilde{x}$	$(-\infty, \infty)$	$(-\infty, \overline{x}\rbrack\!\rbrack$	$\lbrack\!\lbrack \underline{x}, \overline{x}\rbrack\!\rbrack$	$\lbrack\!\lbrack \underline{x}, \infty)$

$-=$		$\tilde{e}$		
		$(-\infty, 0)$	$\{0\}$	$(0, \infty)$
$\tilde{x}$	$(-\infty, \infty)$	$\lbrack\!\lbrack \underline{x}, \infty)$	$\lbrack\!\lbrack \underline{x}, \overline{x}\rbrack\!\rbrack$	$(-\infty, \overline{x}\rbrack\!\rbrack$

$\times=$		$\tilde{e}$						
		$(-\infty, -1)$	$\{-1\}$	$(-1, 0)$	$\{0\}$	$(0, 1)$	$\{1\}$	$(1, \infty)$
$\tilde{x}$	$(-\infty, 0)$	$(-\infty, \infty)$	$\lbrack\!\lbrack \underline{x}, -\underline{x}\rbrack\!\rbrack$	$\lbrack\!\lbrack \underline{x}, \underline{x}\times\underline{e}\rbrack\!\rbrack$	$\lbrack\!\lbrack \underline{x}, 0]$	$\lbrack\!\lbrack \underline{x}, 0)$	$\lbrack\!\lbrack \underline{x}, \overline{x}\rbrack\!\rbrack$	$(-\infty, \overline{x}\rbrack\!\rbrack$
	$\{0\}$	$[0, 0]$						
	$(0, \infty)$	$(-\infty, \infty)$	$(-\overline{x}, \overline{x}\rbrack\!\rbrack$	$(\overline{x}\times\underline{e}, \overline{x}\rbrack\!\rbrack$	$[0, \overline{x}\rbrack\!\rbrack$	$(0, \overline{x}\rbrack\!\rbrack$	$\lbrack\!\lbrack \underline{x}, \overline{x}\rbrack\!\rbrack$	$\lbrack\!\lbrack \underline{x}, \infty)$

$\div=$		$\tilde{e}$						
		$(-\infty, -1)$	$\{-1\}$	$(-1, 0)$	$\{0\}$	$(0, 1)$	$\{1\}$	$(1, \infty)$
$\tilde{x}$	$(-\infty, 0)$	$\lbrack\!\lbrack \underline{x}, \underline{x}\div\overline{e}\rbrack\!\rbrack$	$\lbrack\!\lbrack \underline{x}, -\underline{x}\rbrack\!\rbrack$	$(-\infty, \infty)$	undefined	$(-\infty, \overline{x}\rbrack\!\rbrack$	$\lbrack\!\lbrack \underline{x}, \overline{x}\rbrack\!\rbrack$	$\lbrack\!\lbrack \underline{x}, 0)$
	$\{0\}$	$[0, 0]$			undefined	$[0, 0]$		
	$(0, \infty)$	$(\overline{x}\div\overline{e}, \overline{x}\rbrack\!\rbrack$	$(-\overline{x}, \overline{x}\rbrack\!\rbrack$	$(-\infty, \infty)$	undefined	$\lbrack\!\lbrack \underline{x}, \infty)$	$\lbrack\!\lbrack \underline{x}, \overline{x}\rbrack\!\rbrack$	$(0, \overline{x}\rbrack\!\rbrack$

Table 1: Partial behaviors for numeric effects

contributing to it. For assignment effects $:=$ we do not need a table as the behavior is equal for all classes, and $\mathcal{T}_{:}(x, e) = \lbrack\!\lbrack \min(\underline{x}, \underline{e}), \max(\overline{x}, \overline{e})\rbrack\!\rbrack$.

Theorem 2. *The partial behaviors $\mathcal{T}_{\circ}(x, e)$ are equal to $\mathrm{succ}_{\circ}(x, e)$ for $x \subseteq \tilde{x} \in \mathcal{B}_x$ and $e \subseteq \tilde{e} \in \mathcal{B}_e$.*

We prove Theorem 2 exemplarily for two of the less obvious entries in Table 1. The proofs for the remaining cases can be done similarly.

Proof for multiplication, $x \subseteq \tilde{x} = (0, \infty)$ and $e \subseteq \tilde{e} = (0, 1)$: We have to show that $\mathrm{succ}_{\times}(x, e) = (0, \overline{x}\rbrack\!\rbrack$.

"$\subseteq$": In order to prove $\mathrm{succ}_{\times}(x, e) \subseteq (0, \overline{x}\rbrack\!\rbrack$, we show that for every element $q \in \mathrm{succ}_{\times}(x, e) = \bigcup_{i=0}^{\infty} x_i$ there exists an index $k \in \mathbb{N}$ with $q \in x_k = \lbrack\!\lbrack \underline{x_k}, \overline{x_k}\rbrack\!\rbrack$ and $x_k \subseteq (0, \overline{x}\rbrack\!\rbrack$. We prove this subset relation separately for each bound of x_k.

Lower bound: We show $\underline{x_k} > 0$ for all $k \in \mathbb{N}$ by induction. The base case $\underline{x_0} > 0$ holds as $x_0 = x \subseteq (0, \infty)$. Inductively $\underline{x_{i+1}} = \underline{x_i \sqcup (x_i \times e)} = \min(\underline{x_i}, \underline{x_i} \times \underline{e}, \underline{x_i} \times \overline{e}, \overline{x_i} \times \underline{e}, \overline{x_i} \times \overline{e})$. As $x \subseteq (0, \infty)$ and $e \subseteq (0, 1)$ are both positive, the result is positive as well. The minimum is obtained for $\underline{x_i} \times \underline{e}$ because e is a contraction. Thus, for all k it holds that $q \geq \underline{x_k} > 0$.

Upper bound: Again, we show $\overline{x_k} \leq \overline{x}$ for all $k \in \mathbb{N}$ by induction. The base case holds because $\overline{x_0} = \overline{x}$ with interval open/closed as for $\overline{x}$. The upper bound does not change in the inductive step and we have $\overline{x_{i+1}} = \overline{x_i}$ because $\overline{x_{i+1}} \geq \overline{x_i}$ by definition of the convex union and $\overline{x_{i+1}} \leq \overline{x_i}$ because $\overline{x_{i+1}} = \overline{x_i \times e}$ which is smaller than $\overline{x_i}$ because $0 < \underline{e} \leq \overline{e} < 1$ is a contraction. The upper bound of the interval remains open/closed as $\overline{x}$ as well for $\overline{x_{i+1}}$. Thus, for every q it holds that $q \leq \overline{x_k} \leq \overline{x}$. Together with the lower bound $0 < q$ we can conclude that $q \in (0, \overline{x}\rbrack\!\rbrack$.

"$\supseteq$": Now we have to show the converse direction $\mathrm{succ}_{\times}(x, e) \supseteq (0, \overline{x}\rbrack\!\rbrack$. Let $q \in (0, \overline{x}\rbrack\!\rbrack$. We have to show that $q \in \mathrm{succ}_{\times}(x, e)$. As $\overline{x_k} = \overline{x}$ for all $k \in \mathbb{N}$ we only have to show that there exists a $k \in \mathbb{N}$ with $q \in x_k = \lbrack\!\lbrack \underline{x_k}, \overline{x_k}\rbrack\!\rbrack$ because $x_{i+1} \supseteq x_i$ and therefore $q \in \mathrm{succ}_{\times}(x, e) = \bigcup_{i=0}^{\infty} x_i$. Such a k exists because to obtain $\underline{x} \times \underline{e}^k < q$ respectively $\underline{e}^k < q \div \underline{x}$. Building the logarithm alters the inequality because $\underline{e} < 0$: $\log_{\underline{e}}(\underline{e}^k) > \log_{\underline{e}}(q \div \underline{x})$. Therefore, $q \in \mathrm{succ}_{\times}(x, e)$ for $k \geq \lceil \log_{\underline{e}}(q \div \underline{x}) \rceil$. □

Proof for division, $x \subseteq \tilde{x} = (-\infty, 0)$ and $e \subseteq \tilde{e} = (-\infty, -1)$: We have to show that $\mathrm{succ}_{\div}(x, e) = \lbrack\!\lbrack \underline{x}, \underline{x} \div \overline{e}\rbrack\!\rbrack$.

"$\subseteq$": We prove $\mathrm{succ}_{\div}(x, e) \subseteq \lbrack\!\lbrack \underline{x}, \underline{x} \div \overline{e}\rbrack\!\rbrack$ again by showing that for every $q \in \mathrm{succ}_{\div}(x, e) = \bigcup_{i=0}^{\infty} x_i$ there exists an index $k \in \mathbb{N}$ with $q \in x_k = \lbrack\!\lbrack \underline{x_k}, \overline{x_k}\rbrack\!\rbrack$. We show inductively that $x_k \subseteq \lbrack\!\lbrack \underline{x}, \underline{x} \div \overline{e}\rbrack\!\rbrack$ for all $k \in \mathbb{N}$.

Base case: For $q \in x_0 = \lbrack\!\lbrack \underline{x}, \overline{x}\rbrack\!\rbrack$ with bounds open/closed as in x, it is easy to show that also $q \in \lbrack\!\lbrack \underline{x}, \underline{x} \div \overline{e}\rbrack\!\rbrack$ because from $x \subseteq (-\infty, 0)$ and $e \subseteq (-\infty, -1)$ we know that $\underline{x}$, $\overline{x}$, $\underline{e}$ and $\overline{e}$ are all negative and therefore $\underline{x} \div \overline{e} > 0 > \overline{x}$ so the upper bound on the right hand side is always greater than the upper bound of $\overline{x_0}$ and we have $\underline{x} = \underline{x_0} \leq q \leq \overline{x_0} < \underline{x} \div \overline{e}$.

Inductive step, lower bound: We have to prove that the lower bound $\underline{x_{i+1}} \geq \underline{x}$, with the induction hypothesis that $\underline{x_i} \geq \underline{x}$ holds. The new bound $\underline{x_{i+1}} = \underline{x_i \sqcup (x_i \div e)} = \min(\underline{x_i}, \underline{x_i} \div \underline{e}, \underline{x_i} \div \overline{e}, \overline{x_i} \div \underline{e}, \overline{x_i} \div \overline{e})$ is attained at $\underline{x_i}$ because $-\infty < \underline{e} < \overline{e} < 0$ and $\underline{x_i}$ are negative making the results $\underline{x_i} \div \underline{e}$ and $\underline{x_i} \div \overline{e}$ positive and obviously greater than 0 and as such also greater than $\underline{x_i}$. If the upper bound $\overline{x_i}$ is also negative, the minimum for $\underline{x_{i+1}}$ is clearly attained by $\underline{x_i}$ but even if x_i is positive, it is bounded by $0 \leq \overline{x_i} \leq (\underline{x} \div \overline{e})$. Because the division by $e \subseteq (-\infty, -1)$ is a contraction, the highest absolute value is attained by dividing by $\overline{e}$ but still $(\underline{x} \div \overline{e}) \div \overline{e} \geq (\overline{x_i} \div \overline{e}) \geq \underline{x_i}$. Therefore, the minimum of $\underline{x_{i+1}} = \underline{x_i}$. The lower bound remains open/closed for all k and $q \geq \underline{x_{i+1}} = \underline{x_i} = \underline{x_0} = \underline{x}$.

Inductive step, upper bound: We now have to show that $\overline{x_{i+1}} \leq \underline{x} \div \overline{e}$. The new bound $\overline{x_{i+1}}$ is computed as $\overline{x_i \sqcup (x_i \div e)} = \max(\overline{x_i}, \underline{x_i} \div \underline{e}, \underline{x_i} \div \overline{e}, \overline{x_i} \div \underline{e}, \overline{x_i} \div \overline{e})$. The maximum is attained at $\underline{x_i} \div \overline{e}$ (and at $\overline{x_i}$ if they are equal). The reasoning is as follows: all elements of e are negative and if $\underline{x_i}$ and $\overline{x_i}$ are both negative, $\underline{x_i}$ has the higher absolute value. If $\overline{x_i}$ is positive, the division by a negative number will not contribute to a higher upper bound. As $\underline{e} < \overline{e} < -1$ is a contraction, the highest value is achieved for $\underline{x} \div \overline{e}$ with bounds closed if the bounds corresponding to $\underline{x}$ and to $\overline{e}$ are both closed, and open otherwise. With $\underline{x_i} > \underline{x}$ by induction hypothesis, we can therefore conclude

that $\overline{x_{i+1}} \leq \underline{x_i} \div \overline{e} \leq \underline{x} \div \overline{e}$. Therefore, $q \leq \overline{x_k} \leq \underline{x} \div \overline{e}$. Together with the lower bound $\underline{x} \leq \underline{x_k} \leq q$ we can conclude that $q \in (\!|\underline{x}, \underline{x} \div \overline{e}|\!)$.

"$\supseteq$": We have to show that $\mathrm{succ}_\div(x, e) \supseteq (\!|\underline{x}, \underline{x} \div \overline{e}|\!)$. Let $q \in (\!|\underline{x}, \underline{x} \div \overline{e}|\!)$. We have to show that it then also follows that $q \in \mathrm{succ}_\div(x, e)$. As $\underline{x_k} = \underline{x}$ for all $k \in \mathbb{N}$ we only have to show that there exists a $k \in \mathbb{N}$ with $q \in x_k = (\!|\underline{x_k}, \overline{x_k}|\!)$ because $x_{i+1} \subseteq x_i$ and therefore $q \in \mathrm{succ}_\div(x, e) = \bigcup_{i=0}^{\infty} x_i$. The maximum is obtained after $k = 1$ steps because the maximum to compute $\overline{x_{i+1}} = \underline{x_i} \div \overline{e}$ only depends on the lower bound $\underline{x_i}$ which equals $\underline{x}$ for all $k \geq 0$. □

With such a decomposition, numeric effects can now be computed in constant time. Unfortunately, the union of the partial behaviors of an effect does not equal the semantic of a successor.

Hypothesis 1. The successor $\mathrm{succ}_\circ(x, e)$ of an effect x ∘=e is the union of the successors obtained by decomposition of the effect into behavior classes, i.e. $\bigcup_{\tilde{x} \in \mathcal{B}_x, \tilde{e} \in \mathcal{B}_e} \mathrm{succ}_\circ(x \cap \tilde{x}, e \cap \tilde{e}) = \mathrm{succ}_\circ(x, e)$ where $\mathrm{succ}_\circ(\emptyset, e) = \mathrm{succ}_\circ(x, \emptyset) = \emptyset$.

Hypothesis 1 does not hold in general, as the following example illustrates. The successor can grow into behavior classes which were not covered by the decomposition:

Example 4. Let $o = \langle \emptyset \to \{\mathsf{x} \times\!= \mathsf{e}\} \rangle$ have a numeric effect on x in a state where $x = [1, 4]$ and $e = [-\frac{1}{2}, 2]$. The successor $\mathrm{succ}_\times(x, e)$ is $(-\infty, \infty)$. However, the partial behaviors of the decomposition are $\mathrm{succ}_\times(x, [-\frac{1}{2}, 0)) = [-2, 4]$, $\mathrm{succ}_\times(x, [0, 0]) = [0, 4]$, $\mathrm{succ}_\times(x, (0, 1)) = (0, 4]$, $\mathrm{succ}_\times(x, [1, 1]) = [1, 4]$ and $\mathrm{succ}_\times(x, (1, 2]) = [1, \infty)$. With the union $\mathrm{succ}_\times(x \cap \tilde{x}, e \cap \tilde{e}) = [-2, \infty)$ which differs from $\mathrm{succ}_\times(x, e) = (-\infty, \infty)$.

However, the number of behavior classes is restricted, and therefore, new behavior classes can only be hit a restricted number of times. The hypothesis can therefore be fixed by including the partial behaviors $\mathcal{T}_\circ(x, e)$ of the classes attained by x in a nested fix-point iteration: Let $x_0 = x$ and $x_{j+1} = \bigcup_{\tilde{x} \in \mathcal{B}_x, \tilde{e} \in \mathcal{B}_e} \mathrm{succ}_\circ(x_j \cap \tilde{x}, e \cap \tilde{e})$ with $\mathrm{succ}_\circ(\emptyset, e) = \mathrm{succ}_\circ(x, \emptyset) = \emptyset$ for $j \geq 0$. Let $\widetilde{\mathrm{succ}}_\circ(x, e) = \bigcup_{j=0}^{\infty} x_j$. Now, the newly attained behavior classes become part of the decomposition in the next iteration.

Example 5. Recall Example 4 starting with $x_0 = x = [1, 4]$ where the successor $\mathrm{succ}_\times(x_0 \cap \tilde{x}, e \cap \tilde{e}) = [-2, \infty)$. Building the decomposition over the newly achieved behavior classes with $x_1 = [-2, \infty)$ and $e = [-\frac{1}{2}, 2]$ contains among others $\mathrm{succ}_\times([-2, 0), (1, 2]) = (-\infty, 0)$. The union still contains partial behaviors which set the upper bound to ∞ and therefore $\mathrm{succ}_\times(x_1 \cap \tilde{x}, e \cap \tilde{e}) = (-\infty, \infty)$. Now, a fix-point is reached and $\widetilde{\mathrm{succ}}_\circ(x, e) = \mathrm{succ}_\circ(x, e)$.

Lemma 1. The union of decomposed successors $\widetilde{\mathrm{succ}}_\circ(x, e)$ converges after at most 21 steps.

Proof sketch. The number of behaviors in each class is restricted to $|\mathcal{B}_x| = 3$ and $|\mathcal{B}_e| = 7$. Most partial behaviors $\mathcal{T}_\circ(x, e)$ either set a new bound to a certain value (0 or $\pm\infty$), or leave a bound of x unchanged. The only unsafe cases are multiplications or divisions of a bound with -1 or e. However, none of these cases is problematic because e is fixed: $\mathcal{T}_\times(x, e)$ with $x \subseteq \tilde{x} = (-\infty, 0)$ and $e \subseteq \tilde{e} = (-1, 0)$ sets a new upper bound $\underline{x} \times \underline{e} > 0$. However, for all classes $\mathcal{T}_\times(x, e)$ with $x \subseteq \tilde{x} = (0, \infty)$, the upper bound is either set to ∞ or it remains the same. Therefore no problematic interactions occur. The same reasoning holds for $\mathcal{T}_\times(x, e)$ with $x \subseteq \tilde{x} = (0, \infty)$ and $e \subseteq \tilde{e} = (-1, 0)$ as well as $\mathcal{T}_\div(x, e)$ with $e \subseteq \tilde{e} = (-\infty, -1)$. ■

The feasibility of a decomposition can therefore be reformulated to the following Theorem:

Theorem 3. *The successor* $\mathrm{succ}_\circ(x, e)$ *of an effect* $x \circ\!= e$ *is the fix-point of the convex union of the successors obtained by decomposition of the effect into behavior classes, i.e.* $\widetilde{\mathrm{succ}}_\circ(x, e) = \mathrm{succ}_\circ(x, e)$.

Proof sketch. It should be evident that $\widetilde{\mathrm{succ}}_\circ(x, e) \subseteq \mathrm{succ}_\circ(x, e)$. In the first iteration of $\widetilde{\mathrm{succ}}_\circ(x, e)$ all partial behaviors $\mathrm{succ}_\circ(x \cap \tilde{x}, e \cap \tilde{e})$ are operations on subsets of x and e. As interval arithmetic is well defined, an arithmetic operation on a interval x will therefore always subsume the interval resulting from the same operation of a sub-interval $x' \subseteq x$. During each iteration of $\widetilde{\mathrm{succ}}_\circ(x, e)$, the decomposition can only grow to behavior classes that were part of $\mathrm{succ}_\circ(x, e)$ in the first place.

The converse direction $\widetilde{\mathrm{succ}}_\circ(x, e) \supseteq \mathrm{succ}_\circ(x, e)$ is shown by contradiction. Let $q \in \mathrm{succ}_\circ(x, e)$ but not in $\widetilde{\mathrm{succ}}_\circ(x, e)$. Both successor functions are defined recursively starting with $x_0 = x$. Therefore $q \notin x_0$, and there has to be a $k > 0$ in $\mathrm{succ}_\circ(x, e)$ with $k_{k+1} = x_k \sqcup (x_k \circ e)$ so that $x_k \subset \widetilde{\mathrm{succ}}_\circ(x, e)$ but $x_{k+1} \not\subset \widetilde{\mathrm{succ}}_\circ(x, e)$. After k steps, the bound of the successors extended beyond the decomposition $\widetilde{\mathrm{succ}}_\circ(x, e)$ for the first time. Obviously, the new bound does not originate in x_k but the new interval x_{k+1} is obtained from $(x_k \circ e)$. The resulting interval depends on $\underline{x_k}$, $\overline{x_k}$, $\underline{e}$, $\overline{e}$ and in case of division also on whether $0 \in e$. Each combination of these extreme bounds is contained in one partial behavior $\mathcal{T}_\circ(x_k, e)$. If $(x_k \circ e)$ hits a new behavior class or extends the bounds within a behavior class, this is a contradiction to $\widetilde{\mathrm{succ}}_\circ(x, e)$ being a fix-point. If $(x_k \circ e)$ stays within a behavior, this is a contradiction to $\mathcal{T}_\circ(x_k, e)$ being well defined (Theorem 2). Thus, such a k cannot be found, and therefore, it is impossible for $q \in \mathrm{succ}_\circ(x, e)$ but not $q \in \widetilde{\mathrm{succ}}_\circ(x, e)$. ■

With the help of the decomposed successor $\widetilde{\mathrm{succ}}_\circ(x, e)$ we can compute the result of applying an operator $\mathrm{app}_o^\#$ with the repetition relaxed semantic in constant time. This allows us to use the parallel fix-point algorithm from the classical case analogously: apply all applicable operators in parallel until a fix-point is reached. If the algorithm terminates, the plan is indeed a plan.

Theorem 4. *The parallel fix-point algorithm for repetition relaxed planning is correct, i.e. if the algorithm outputs an alleged plan, it is indeed a plan for* $\Pi^\#$.

Proof. Operators are only applied if the precondition is fulfilled. □

Unfortunately, the algorithm does not necessarily terminate. In the definition of the semantic of a repetition relaxed planning task, we fix the effect e even if it depends on x. However, this implicit independence assumption is

not justified. Inspecting the entries in Table 1 reveals critical entries (marked in red) for multiplicative effects which contract x and flip the arithmetic sign at the same time. The same is true for assignment effects where $\mathcal{T}_{:}(x, e) = (\!|\min(\underline{x}, \underline{e}), \max(\overline{x}, \overline{e})|\!)$. In these cases, the new value of x can have a different behavior, if e also depends on x. As e can change when x changes, the algorithm does not necessarily terminate.

Example 6. Let $x = [-1, -1]$ and $o = \langle \emptyset \to \{\mathsf{x} \times\!= \mathsf{e}\}\rangle$ with $\mathsf{e} = -\frac{\mathsf{x}+1}{2}$. The goal is $\mathcal{G} = \{\mathsf{x} \geq 1\}$.

Applying the operator arbitrarily often according to the repetition semantic yields the following progression for k operator applications:

k	x	e
0	$[-1, -1]$	$[0, 0]$
1	$[-1, 0]$	$[-0.5, 0]$
2	$[-1, 0.5]$	$[-0.75, 0]$
3	$[-1, 0.75]$	$[-0.875, 0]$
4	$[-1, 0.875]$	$[-0.9375, 0]$
5	$[-1, 0.9375]$	$[-0.96875, 0]$
	$\vdots$	$\vdots$

Obviously, interval x does *not* only change a restricted number of times, so the fix-point algorithm for interval relaxed numeric planning will not terminate.

If we succeeded in directly computing the fix-point to which the intervals converge with a symbolic interval we could continue the fix-point algorithm from here. In Example 6 we could continue if we would set $x = [-1, 1)$ and $e = (-1, 0]$. Unfortunately, the authors did not succeed in finding a general approach to do so (or to prove that such a general approach does not exist). Instead, we will now restrict the problem to planning tasks where the aforementioned problem does not occur. The problem in Example 6 is that e depends on x. Thus, we will restrict planning tasks to contain only effects where the assigned expression is independent from the affected variable. We will then show that such planning tasks are solvable in polynomial time.

Definition 4. A numeric variable v_1 is *directly dependent* on a numeric variable v_2 in task Π if there exists an $o \in \mathcal{O}$ with a numeric effect $\mathsf{v}_1 \circ\!= \mathsf{e}$ so that e contains v_2.

Note that a variable can be *directly dependent* on itself. Also, the definition of *direct dependence* does not consider operator applicability.

Definition 5. A planning task Π is an *acyclic dependency* task, if the *direct dependency* relation is acyclic.

Theorem 5. *The parallel fix-point algorithm for repetition relaxed planning terminates for* acyclic dependency *tasks.*

Proof. As the planning task has *acyclic dependencies*, the *direct dependency* relation induces a topology. Let a *phase* of the algorithm be a sequence of parallel operator applications, where no new operator becomes applicable. During each *phase*, we consider numeric effects in topological order concerning the dependency graph. Let $V_N^{\#l} \subseteq V_N^{\#}$ be the variables in dependency layer l. We iterate over the layers $k \geq 0$ of the topology assuming that a fix-point is reached for all variables $V_N^{\#k}$. Variables $V_N^{\#k+1}$ only depend on variables $V_N^{\#l}$ with $0 \leq l \leq k$ or on constants. A fix-point is reached for all those variables by induction hypothesis. Inductively, we can assume that the expressions of numeric effects which alter the variables of layer $V_N^{\#k+1}$ are fixed. Therefore, the successor $\mathrm{succ}_\circ(x, e)$ of an effect $(\mathsf{x} \circ\!= \mathsf{e})$ with $x = s^{\#}(\mathsf{x})$ and $e = s^{\#}(\mathsf{e})$ does not change the variable more than once (or more than 21 times, if we also consider the intermediate variable updates of the nested fix-point iteration from Lemma 1).

The number of *phases* is restricted, too, with the same argument as for the fix-point algorithm in the classical case. No precondition can be invalidated once it holds, and during each phase at least one operator which was not applicable before must become applicable. The number of phases is therefore restricted to the number of operators in the planning task. □

Theorem 6. *The fix-point algorithm for repetition relaxed planning is complete for* acyclic dependency *tasks.*

Proof. We prove completeness by contradiction and show that it is impossible that the algorithm terminates and reports unsolvable although a plan exists. Now assume there is a plan, but the algorithm terminates and reports unsolvable. Therefore, a satisfiable condition must have been unsatisfied. For propositional conditions, this is impossible, as $s^{\#}(\mathsf{v_p}) \vDash \mathsf{v_p}$ if $\mathsf{v_p} \in \{\mathit{true}, \mathit{both}\}$ and no effect can set a propositional variable to *false*. Additionally, all operators are applied as soon as they are applicable. Thus, without loss of generality, a satisfiable numeric constraint was not achieved by the algorithm. This implies that a numeric effect $(\mathsf{v_n} \circ\!= \mathsf{e})$ would have been able to assign a value to a variable which was not reached by our algorithm. Therefore, the successor defined by the semantic $\mathrm{succ}_\circ(s^{\#}(\mathsf{v_n}), s^{\#}(\mathsf{e}))$ has to be different from the successor computed by the algorithm $\widetilde{\mathrm{succ}}_\circ(s^{\#}(\mathsf{v_n}), s^{\#}(\mathsf{e}))$ which is impossible for numeric tasks with Theorem 3, a contradiction. □

Until now we have an algorithm which can compute parallel plans for repetition relaxed planning tasks in polynomial time for acyclic dependency tasks. As intervals can only grow by applying an operator, the plan can be serialized by applying parallel operators from the same layer in an arbitrary order. Beneficial effects may make the application of some operators unnecessary, but it cannot harm conditions.

We are interested in plans for the interval relaxation without the symbolic description of numeric variables. We will now show that we can derive interval relaxed plans π^+ from repetition relaxed plans $\pi^{\#}$.

Theorem 7. *Plans for the repetition relaxation correspond to plans for the interval relaxation.*

Proof sketch. A serialization of a repetition relaxed plan $\pi^{\#} = \langle o_1, o_2, \ldots, o_n\rangle$ where $0 < i < n$ are the operators applied in the i-th step where the same operator can be applied in multiple steps. We seek to find a repetition constant k_i for each operator in order to satisfy the constraints from interval relaxed planning corresponding to those of the repetition relaxed planning plan. However, repetition relaxed

tasks operate on mixed bounded intervals $\mathbb{I}_m$ whereas interval relaxed tasks are restricted to closed intervals. Thus, we have to explicate the interval bounds as well. The repetition relaxed fix-point algorithm is split into *phases*, were during each phase the same operators are applicable. Within each phase, the operators are applied in parallel at most 21 times for each variable (if all variables have different topology levels in the dependence graph). In order to determine the repetition constants k_i, we look at each constraint $[\underline{a}, \overline{a}] \bowtie [\underline{b}, \overline{b}]$. By definition of $\bowtie$, there exist q_a and q_b in the respective intervals so that $q_a \bowtie q_b$. Let q_a and q_b be such numbers which satisfy the constraint $a \bowtie b$ where a and b are intervals which are obtained by evaluating the corresponding expressions $s^\#(\mathsf{e_a})$ and $s^\#(\mathsf{e_b})$. We investigate each expression individually. For each expression we have a target value q. For the constraints above the expressions are $\mathsf{e_a}$ and $\mathsf{e_b}$ and the corresponding target values q_a and q_b. Unless the expression is a variable, the target value has to be obtained recursively from the expressions $\mathsf{e_1} \circ \mathsf{e_2}$.

Example 7. Let $x = [0, 1)$, $y = [0, 1)$ and $z = (1.7, 3]$ be the symbolic values of variables x, y and z with a condition $\mathsf{x{+}y} > \mathsf{z}$. From $\mathsf{e_a} = \mathsf{x{+}y} \mapsto [0, 2)$ and $\mathsf{e_b} = \mathsf{z}$ we choose an arbitrary $q_a = 1.9 \in s^\#(\mathsf{e_a})$ an arbitrary $q_b = 1.8 \in s^\#(\mathsf{e_b})$ from within the expression intervals so that the constraint is satisfied. Now we have to recursively find appropriate q_x and q_y in the sub-expressions. A leeway of $2 - 1.9 = 0.1$ can be distributed arbitrarily to the target values of the sub-expressions. We could for example continue with target values 0.95 for x and y each.

We can choose arbitrary target values for the sub-expressions within a leeway of feasible choices. Eventually, all expressions induce target values for the numeric variables. This can induce multiple different target values for each variable where only the most extreme target values have to be considered (an interval including the most extreme target values will also include intermediate target values). All target values originate from a repetition relaxed symbolic state, so they are indeed reachable. In the repetition relaxed plan, each operator which has a numeric effect on a variable with a target value achieved the symbolic value for this variable with a partial behavior from Table 1. For each operator, the constant k is now computed by solving $x \pm k \cdot e = q$ for additive effects and $x \circledast e^k = q$ for multiplicative effects. The k is therefore the same k from the proof of Theorem 2. Each operator then has to be applied n times, where n is the sum over all k_p in the phases of the algorithm, where k_p is the maximum number of applications required for that operator in that phase. ∎

Theorem 8. *The problem to generate an interval relaxed numeric plan is in* P *for tasks with* acyclic dependencies.

Proof. The fix-point algorithm for repetition relaxed planning tasks is correct (Theorem 4) and complete (Theorem 6) and it terminates in polynomial time (Theorem 5). Therefore, generating a repetition relaxed plan $\pi^\#$ is possible in polynomial time. An interval relaxed plan π^+ can be constructed from $\pi^\#$ (Theorem 7) in polynomial time. □

The definition of a relaxation is *adequate* (Hoffmann 2003) if it is *admissible*, i.e. any plan π for the original task Π is also a relaxed plan for Π^+, if it offers *basic informedness*, i.e. the empty plan is a plan for Π iff it is a plan for Π^+ and finally the plan existence problem for the relaxation is in P.

Theorem 9. *The* interval relaxation *is adequate for* acyclic dependency *tasks.*

Proof. Admissibility. After each step of the plan π, if propositional variables of the relaxed state differ from the original state, they assign to *both* which cannot invalidate any (goal or operator) conditions. The original value of numeric variables is contained in the interval of the relaxed state. As comparison constraints are defined with the relaxed semantic that a constraint holds if it holds for any pair of elements from the two intervals, admissibility follows directly.

Basic informedness. No (goal or operator) conditions are dropped from the task. Relaxed numeric variables are mapped to degenerate intervals which only contain one element. Therefore, conditions in the original task $x \bowtie y$ correspond to interval constraints $[x, x] \bowtie [y, y]$ which are satisfied iff they are satisfied in the relaxed task.

Polynomiality. As a corollary to Theorem 8, we can also conclude that interval relaxed numeric plan existence is in P for tasks with *acyclic dependencies*.

The interval relaxation is admissible and offers basic informedness. For *acyclic dependency* tasks, the plan existence problem can be decided in polynomial time. Thus, the *interval relaxation* is adequate. □

The proposed relaxation advances the state of the art even though the adequacy of interval relaxation was only shown for a restricted set of tasks. However, the requirement of acyclic dependency for numeric expressions is a strict generalization of expressions e being required to be constant, which is required for other state-of-the-art approaches e.g. (Hoffmann 2003). On the practical side, many interesting planning problems are restricted to constant expressions.

Conclusion and Future Work

We presented interval algebra as a means to carry the concept of a delete relaxation from classical to numeric planning. We proved that this relaxation is adequate for *acyclic dependency tasks*, tasks where the expressions of numeric effects do not depend on the affected variable. The complexity of the approach for arbitrary interval relaxed planning problems remains an open research issue though. It is imaginable that a clever approach can find the fix-point of arbitrary operator application in polynomial time.

In the future, we intend to adapt the most iconic heuristics from classical planning, $h_{\max}$, h_{add} and h_{FF} to the interval relaxation framework.

Acknowledgments

This work was partly supported by the DFG as part of the SFB/TR 14 AVACS.

References

Bäckström, C., and Nebel, B. 1993. Complexity results for SAS$^+$ planning. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI 1993)*.

Bonet, B., and Geffner, H. 1999. Planning as Heuristic Search: New Results. In *Proceedings of the 5th European Conference on Planning (ECP 1999)*, 360–372.

Bonet, B., and Geffner, H. 2001. Planning as Heuristic Search. *Artificial Intelligence* 129(1–2):5–33.

Bonet, B.; Loerincs, G.; and Geffner, H. 1997. A Robust and Fast Action Selection Mechanism for Planning. In *Proceedings of the 14th National Conference on Artificial Intelligence and 9th Innovative Applications of Artificial Intelligence Conference (AAAI 1997/ IAAI 1997)*, 714–719.

Bylander, T. 1994. The Computational Complexity of Propositional STRIPS Planning. *Artificial Intelligence 69 (AI 1994)* 165–204.

Coles, A.; Fox, M.; Long, D.; and Smith, A. 2008. A Hybrid Relaxed Planning Graph-LP Heuristic for Numeric Planning Domains. In *Proceedings of the 20th International Conference on Automated Planning and Search (ICAPS 2008)*.

Fikes, R., and Nilsson, N. J. 1971. STRIPS: a New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence 2 (AI 1971)* 189–208.

Fox, M., and Long, D. 2003. PDDL2.1 : An Extension to PDDL for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research 20 (JAIR 2003)* 61–124.

Helmert, M. 2002. Decidability and Undecidability Results for Planning with Numerical State Variables. In *Proceedings of the 6th International Conference on Artificial Intelligence Planning and Scheduling (AIPS 2002)*, 303–312.

Hoffmann, J. 2003. The Metric-FF Planning System: Translating 'Ignoring Delete Lists' to Numeric State Variables. *Journal of Artificial Intelligence Research 20 (JAIR 2003)* 291–341.

Moore, R. E.; Kearfott, R. B.; and Cloud, M. J. 2009. *Introduction to Interval Analysis*. Society for Industrial and Applied Mathematics.

Young, R. C. 1931. The Algebra of Many-valued Quantities. *Mathematische Annalen 104* 260–290.

Tight Bounds for HTN planning with Task Insertion[1]

Ron Alford
ASEE/NRL Postdoctoral Fellow
Washington, DC, USA
ronald.alford.ctr@nrl.navy.mil

Pascal Bercher
Ulm University
Ulm, Germany
pascal.bercher@uni-ulm.de

David W. Aha
U.S. Naval Research Laboratory
Washington, DC, USA
david.aha@nrl.navy.mil

Abstract

Hierarchical Task Network (HTN) planning with Task Insertion (TIHTN planning) is a formalism that hybridizes classical planning with HTN planning by allowing the insertion of operators from outside the method hierarchy. This additional capability has some practical benefits, such as allowing more flexibility for design choices of HTN models: the task hierarchy may be specified only partially, since "missing required tasks" may be inserted during planning rather than prior planning by means of the (predefined) HTN methods.

While task insertion in a hierarchical planning setting has already been applied in practice, its theoretical properties have not been studied in detail, yet – only EXPSPACE membership is known so far. We lower that bound proving NEXPTIME-completeness and further prove tight complexity bounds along two axes: whether variables are allowed in method and action schemas, and whether methods must be totally ordered. We also introduce a new planning technique called *acyclic progression*, which we use to define provably efficient TIHTN planning algorithms.

1 Introduction

Hierarchical Task Network (HTN) planning (Erol *et al.* 1996) is a planning approach, where solutions are generated via step-wise refinement of an initial task network in a top-down manner. Task networks may contain both compound and primitive tasks. Whereas primitive tasks correspond to standard STRIPS-like actions that can be applied in states where their preconditions are met, compound tasks are abstractions thereof. That is, for every compound task, the domain features a set of decomposition methods, each mapping that task to a task network. Solutions are obtained by applying methods to compound tasks thereby replacing these tasks by the task network of the respective method. HTN planning is strictly more expressive than non-hierarchical (i.e., classical) planning. That is, solutions of HTN problems may be structured in a way that are more complex than solutions of classical planning problems (Höller *et al.* 2014). However, HTN planning is also harder than classical planning. The complexity of the plan existence problem ranges up to undecidable even for *propositional* HTN planning (Erol *et al.* 1996; Geier and Bercher 2011; Alford *et al.* 2015). Even the verification of HTN solutions is harder than in classical planning (Behnke *et al.* 2015).

Table 1: Summary of our TIHTN plan existence results (not including Corollary 7). All results are completeness results.

Ordering	Variables	Complexity	Theorem
total	no	PSPACE	Thm. 3
total	yes	EXPSPACE	Thm. 4
partial	no	NEXPTIME	Thm. 5, 6
partial	yes	2-NEXPTIME	Thm. 5, 6

Hierarchical planning approaches are often chosen for real-world application scenarios (Nau *et al.* 2005; Lin *et al.* 2008; Biundo *et al.* 2011) due to the ability to specify solution strategies in terms of methods, but also because human expert knowledge is often structured in a hierarchical way and can thus be smoothly integrated into HTN planning models. On the other side, these methods also make HTN planning *less flexible* than non-hierarchical approaches, because only those solutions may be generated that are "reachable" via the decomposition of the methods. So, defining only a partially hierarchical domain is not sufficient to produce all desired solutions. Several HTN researchers have thus investigated how partially hierarchical domain knowledge can be exploited during planning without relying on the restricted (standard) HTN formalism (Kambhampati *et al.* 1998; Biundo and Schattenberg 2001; Alford *et al.* 2009; Geier and Bercher 2011; Shivashankar *et al.* 2013).

The most natural way to overcome that restriction is to allow *both* the definition and decomposition of compound tasks *and* the insertion of tasks from outside the decomposition hierarchy as it is done, for example, by Kambhampati *et al.* (1998) and Biundo and Schattenberg (2001) in *Hybrid Planning* – a planning approach fusing HTN planning with Partial-Order Causal-Link (POCL) planning.

This additional flexibility also pays off in terms of computational complexity: allowing arbitrary insertion of tasks into task networks lowers the complexity of the plan existence problem from undecidable (for standard HTN plan-

[1]This paper will appear in IJCAI-2015.

ning) to EXPSPACE membership (for HTN planning with task insertion – TIHTN planning) (Geier and Bercher 2011). This reduction of the complexity also has its negative consequences, however. Some planning problems that can be easily expressed in the HTN planning setting may not be expressed in the TIHTN setting (others only with a blowup of the problem size) (Höller *et al.* 2014). Also, the set of TIHTN solutions of a problem may not correspond to the ones the domain modeler intended: in HTN planning – as opposed to TIHTN planning – only those plans are regarded solutions that can be obtained by decomposition only. Thus, certain plans may be ruled out even if they are executable.

In this paper, we investigate the influence of method structure (partially vs. totally ordered) and of variables (propositional vs. lifted TIHTN problems), and provide tight complexity bounds of the plan existence problem for the four resulting classes of TIHTN problems. The results are summarized in Table 1 (and compared to the respective results from HTN planning in Table 2 in the last section). Notably, we show that propositional TIHTN planning is NEXPTIME-complete, easier than the previously known EXPSPACE upper bound. Besides providing tight complexity bounds for the plan existence problem, another contribution is a new algorithm, called acyclic progression, that efficiently solves TIHTN problems. The paper closes with a discussion about the new complexity findings for TIHTN planning that puts them into context of already known results for HTN planning.

2 Lifted HTN Planning with Task Insertion

Geier and Bercher (2011) defined a propositional (set-theoretic) formalization of HTN and TIHTN planning problems. Both problem classes are syntactically identical – they differ only in the solution criteria. Recently, we extended their formalization of HTN planning to a lifted representation based upon a function-free first order language (Alford *et al.* 2015), where the semantics are given via grounding. For the purpose of this paper, we replicate the definitions for lifted HTN planning and extend them by *task insertion* to allow specifying lifted TIHTN planning problems.

Task names represent activities to accomplish and are syntactically first-order atoms. Given a set of task names X, a *task network* is a tuple $tn = (T, \prec, \alpha)$ such that:

- T is a finite nonempty set of *task symbols*.
- $\prec$ is a strict partial order over T.
- $\alpha : T \to X$ maps from task symbols to task names.

Since task networks are only partially ordered and any task name might be required to occur several times, we need a way to "identify" them uniquely. For that purpose, we use the task symbols T as unique place holders. A task network is called *ground* if all task names occurring in it are variable-free.

A *lifted TIHTN problem* is a tuple $(\mathcal{L}, \mathcal{O}, \mathcal{M}, s_I, tn_I)$, where:

- $\mathcal{L}$ is a function-free first order language with a finite set of relations and constants.
- $\mathcal{O}$ is a set of *operators*, where each $o \in \mathcal{O}$ is a triple (n, χ, e), n being its task name (referred to as $name(o)$) not occurring in $\mathcal{L}$, χ being a first-order logic formula called the *precondition* of o, and e being a conjunction of positive and negative literals in $\mathcal{L}$ called the *effects* of o. We refer to the set of task names in $\mathcal{O}$ as *primitive*.
- $\mathcal{M}$ is a set of *(decomposition) methods*, where each method m is a pair (c, tn), c being a (non-primitive or compound) task name, called the method's *head* not occurring in $\mathcal{O}$ or $\mathcal{L}$, and tn being a task network, called the method's *subtasks*, defined over the names in $\mathcal{O}$ and the method heads in $\mathcal{M}$.
- s_I is the (ground) initial state and tn_I is the initial task network that is defined over the names in $\mathcal{O}$.

We define the semantics of lifted TIHTN planning through grounding. For the details of the grounding process, we refer to (Alford *et al.* 2015). The ground (or propositional) TIHTN planning problem obtained from $(\mathcal{L}, \mathcal{O}, \mathcal{M}, s_I, tn_I)$ is given by $P = (\mathcal{L}, O, M, s_I, tn'_I)$.

The operators O form an implicit *state-transition function* $\gamma : 2^{\mathcal{L}} \times O \to 2^{\mathcal{L}}$ for the problem, where:

- A state is any subset of the ground atoms in $\mathcal{L}$. The finite set of states in a problem is denoted by $2^{\mathcal{L}}$;
- o is *applicable* in a state s iff $s \models prec(o)$;
- $\gamma(s, o)$ is defined iff o is applicable in s; and
- $\gamma(s, o) = (s \setminus del(o)) \cup add(o)$.

Executability A sequence of ground operators $\langle o_1, \ldots, o_n \rangle$ is *executable* in a state s_0 iff there exists a sequence of states $s_1, \ldots, s_n$ such that $\forall_{1 \leq i \leq n} \gamma(s_{i-1}, o_i) = s_i$. A ground task network $tn = (T, \prec, \alpha)$ is *primitive* iff it contains only task names from $\mathcal{O}$. tn is *executable* in a state s_0 iff tn is primitive and there exists a total ordering $t_1, \ldots, t_n$ of T consistent with $\prec$ such that $\langle \alpha(t_1), \ldots, \alpha(t_n) \rangle$ is executable in s_0.

Task Decomposition Primitive task networks can only be obtained by refining the initial task network via *decomposition* of compound tasks. Intuitively, decomposition is done by selecting a task with a non-primitive task name, and replacing the task in the network with the task network of a corresponding method. More formally, let $tn = (T, \prec, \alpha)$ be a task network and let $m = (\alpha(t), (T_m, \prec_m, \alpha_m)) \in M$ be a method. Without loss of generality, assume $T \cap T_m = \emptyset$. Then the decomposition of t in tn by m into a task network tn', written $tn \xrightarrow[t,m]{}_D tn'$, is given by:

$$
\begin{aligned}
T' &\coloneqq (T \setminus \{t\}) \cup T_m \\
\prec' &\coloneqq \{(t, t') \in \prec \mid t, t' \in T'\} \cup \prec_m \\
&\quad \cup \{(t_1, t_2) \in T_m \times T \mid (t, t_2) \in \prec\} \\
&\quad \cup \{(t_1, t_2) \in T \times T_m \mid (t_1, t) \in \prec\} \\
\alpha' &\coloneqq \{(t, n) \in \alpha \mid t \in T'\} \cup \alpha_m \\
tn' &\coloneqq (T', \prec', \alpha')
\end{aligned}
$$

If tn' is reachable by any finite sequence of decompositions of tn, we write $tn \to_D^* tn'$.

Task Insertion TIHTN planning, in addition to decomposition, allows tasks to be inserted directly into task networks. Let $tn = (T, \prec, \alpha)$ be a task network, t be a fresh task symbol not in T, and o be a primitive task name. Then *task insertion*, written $tn \xrightarrow[t,o]{}_I tn'$, results in the task network $tn' = (T \cup \{t\}, \prec, \alpha \cup \{(t, o)\})$. If tn' is reachable by any sequence of insertions to tn, we write $tn \rightarrow_I^* tn'$.

Task insertion commutes with decomposition, i.e., if $tn_1 \xrightarrow[t,m]{}_D tn_2 \xrightarrow[t',o]{}_I tn_3$, then there exists a tn_2' such that $tn_1 \xrightarrow[t',o]{}_I tn_2' \xrightarrow[t,m]{}_D tn_3$. If tn' is reachable by any sequence of decompositions and insertions to tn, we write $tn \rightarrow_{DI}^* tn'$.

Solutions Under HTN semantics, a problem P is solvable iff $tn_I \rightarrow_D^* tn'$ and tn' is executable in s_I. The task network tn' is then called an HTN solution of P. Under TIHTN semantics, P is solvable iff there exists a tn' such that $tn_I \rightarrow_{DI}^* tn'$ and tn' is executable in s_I.

The following definitions go beyond those in (Alford *et al.* 2015; Geier and Bercher 2011).

Acyclic Decompositions Let $\overline{tn}$ be a task network sequence starting in tn_I and ending in a task network tn', s.t. $tn_I \rightarrow_{DI}^* tn'$. For each network $tn_j \in \overline{tn}$, every task symbol was either present in tn_I, inserted directly via task insertion, or is the result of a sequence of decompositions of a task symbol in tn_I. For the tasks arrived at by decomposition, we can define their ancestors in the usual way: For $tn_i, tn_{i+1} \in \overline{tn}$ with $tn_i \xrightarrow[t_i,m_i]{}_D tn_{i+1}$, t_i is an ancestor of each task $t' \in tn_{i+1}$ that comes from m_i; and ancestry is transitive, i.e., if t_i is an ancestor of t_j and t_j is an ancestor of t_k, then t_i is an ancestor of t_k. $\overline{tn}$ is *acyclic* if for every task t in its final task network, the ancestors of t all have unique task names. Thus, an acyclic series of decompositions and insertions $tn_I \rightarrow_{DI}^* tn'$ makes no use of recursive methods, regardless of their presence in the set of methods.

Geier and Bercher (2011) represent ancestry using decomposition trees and show that, given a TIHTN solution tn that is obtained via cyclic method application, one can repeatedly remove cycles (replacing orphaned sequences of decomposition with task insertion) to arrive at an acyclic solution tn'.

The next corollary follows as a special case of the application of Lemma 1 and Lemma 2 by Geier and Bercher (2011).

Corollary 1. *Let $P = (\mathcal{L}, O, M, s_I, tn_I)$ be a ground planning problem and let $tn = (T, \prec, \alpha)$ be a task network such that $tn_I \rightarrow_{DI}^* tn$ and $\langle t_1, \ldots, t_k \rangle$ is an executable task sequence of tn that does not violate the ordering constraints $\prec$. Then there exists a task network $tn' = (T', \prec', \alpha')$ with an acyclic decomposition $tn \rightarrow_{DI}^* tn'$ and an executable task sequence $\langle t_1', \ldots, t_k' \rangle$ of tn' not violating α', such that $\forall_i \alpha(t_i) = \alpha'(t_i')$.*

3 Acyclic Progression for TIHTN Planning

HTN planners generally solve problems either using decomposition directly (Erol *et al.* 1994; Bercher *et al.* 2014), or by using *progression* (Nau *et al.* 2003), which interleaves decomposition and finding a total executable order over the primitive tasks (Alford *et al.* 2012). Since progression-based HTN algorithms can be efficient across a number of syntactically identifiable classes of HTN problems (Alford *et al.* 2015), it makes a useful starting point for designing efficient TIHTN algorithms.

We define *acyclic progression* for TIHTN planning as a procedure that performs progression on a current state. To that end, it maintains a task network of primitive and compound tasks that still need to be applied to the current state or to be decomposed, respectively. To avoid recursive definitions, it maintains a set of ancestral task names. More precisely, search nodes are tuples (s, tn, h), where s is a state, $tn = (T, \prec, \alpha)$ is a task network, and h is a mapping of the tasks in T to the set of ancestral task names, represented as a set of task-task-name pairs. For each node, there are three possible operations:

- Task insertion: If o is an operator such that $s \models prec(o)$, then $(\gamma(s, o), tn, h)$ is an acyclic progression of (s, tn, h)
- Task application: If $t \in T$ is an *unconstrained* primitive task ($\forall_{t' \neq t} t' \not\prec t$) and $s \models prec(\alpha(t))$, then we can apply $\alpha(t)$ to s and remove it from tn and h. So, given

$$\begin{aligned} T' &\coloneqq T \setminus \{t\} \\ \prec' &\coloneqq \{(t_1, t_2) \in \prec \mid t_1 \neq t \wedge t_2 \neq t\} \\ \alpha' &\coloneqq \{(t', n) \in \alpha \mid t' \neq t\} \\ tn' &\coloneqq (T', \prec', \alpha') \\ h' &\coloneqq \{(t', n) \in h \mid t' \neq t\} \end{aligned}$$

  then $(\gamma(s, \alpha(t)), tn', h')$ is an acyclic progression of (s, tn, h).
- Task decomposition: If $t \in T$ is an unconstrained nonprimitive task, $(\alpha(t), tn_m) \in M$ is method with $tn_m = (T_m, \prec_m, \alpha_m)$, and none of the task names in α occur as an ancestral task name of t in h, then we can decompose t and update the history. So if $tn \xrightarrow[t,m]{}_D tn'$ and

$$\begin{aligned} h' \coloneqq &\{(t', n) \in h \mid t' \neq t\} \\ &\cup \{(t_m, n) \mid t_m \in T_m, (t, n) \in h\} \\ &\cup \{(t_m, \alpha(t)) \mid t_m \in T_m\} \end{aligned}$$

  then (s, tn', h') is an acyclic progression of (s, tn, h).

If there is a sequence of acyclic progressions from the triple (s, tn, h) to (s', tn', h'), we write $(s, tn, h) \rightarrow_{AP}^* (s', tn', h')$. Notably, acyclic progression is only acyclic over decompositions, not states reached. If there is any sequence of acyclic decompositions to an empty task network, then the problem has a TIHTN solution:

Lemma 2. *Given a ground planning problem $P = (\mathcal{L}, O, M, s_I, tn_I)$, there is a series of acyclic progressions $(s_I, tn_I, \emptyset) \rightarrow_{AP}^* (s, tn_\emptyset, \emptyset)$ (where $tn_\emptyset$ is the empty task network) if and only if P is solvable under TIHTN semantics.*

Proof. ($\Rightarrow$) Let TD be the subsequence of the acyclic progression $(s_I, tn_I, \emptyset) \rightarrow_{AP}^* (s, tn_\emptyset, \emptyset)$ containing all the decompositions performed, TA be the subsequence of task applications, TI be the subsequence of task insertions, and TIA be the subsequence of both task applications and insertions.

Since task application only removes primitive tasks, TD must be a decomposition of tn_I to a primitive task network tn', specifically one with a partial order $\prec'$ which is consistent with the order and content of task applications, TA. Then, given that task insertion commutes with decomposition, TI gives us a set of insertions $tn' \rightarrow_I^* tn''$, and TIA is a witness that tn'' has an executable ordering.

($\Leftarrow$) If P is solvable, by Corollary 1 there is an acyclic decomposition sequence $\overline{tn}$ such that $tn_I \rightarrow_D^* tn$, and a sequence of insertions $tn \rightarrow_I^* tn'$ such that $tn' = (T', \prec', \alpha')$ has an executable ordering $TIA = \langle t_1, \ldots, t_k \rangle$. Insertions commute with both themselves and with decomposition, and decompositions commute with each other so long as one decomposed task is not an ancestor of the other. So the following procedure gives an acyclic progression of P to the empty network. Given that t_i is the last task from TIA to be applied to the state (by insertion or application) and the current triple under progression is (s_j, tn_j, h_j):

- If there is a non-primitive task t_j in tn_j such that t_j is an ancestor of t_{i+1} in the sequence $\overline{tn}$, use the decomposition from $\overline{tn}$ to decompose t_j.
- If t_{i+1} exists in tn_j, progress it out of the task network with task application and move on to t_{i+2}.
- Else, t_{i+1} was obtained by insertion, and so use insertion to apply $\alpha'(t_{i+1})$ to s_j and move on to t_{i+1}.

□

A problem's *acyclic progression bound* is the size of the largest task network reachable via any sequence of acyclic progressions. Only decomposition can increase the task network size. Since decomposition is required to be acyclic, every problem has a finite acyclic progression bound. Given the tree-like structure of decomposition, if m is the max number of subtasks in any method and n is the number of task names, and T is the set of task symbols in the initial task network, then $|T| \cdot m^n$ is an acyclic progression bound of the problem.

If methods are totally ordered (i.e., the $\prec$ relation in each method is a total order), then we reach a much tighter bound:

Theorem 3. *Propositional TIHTN planning for problems with totally-ordered methods is* PSPACE*-complete.*

Proof. Classical planning provides a PSPACE lower bound (Geier and Bercher 2011). Here, we provide an upper bound.

Let $P = (\mathcal{L}, O, M, s_I, tn_I)$ be a ground TIHTN problem where each method is totally ordered. The initial task network tn_I may be partially ordered. Letting $tn_I = (T, \prec, \alpha)$, there are $|T|$ initial tasks. Since the methods are totally ordered, any sequence of progressions $(s_I, tn_I, \emptyset) \rightarrow_{AP}^* (s, tn, h)$ preserves that tn can be described by the relationship of $|T|$ or fewer totally ordered chains of tasks.

Progression can only affect the unconstrained tasks in the chains. While the acyclic decomposition phase of progression can lengthen a chain by $m-1$ tasks (m being the size of the largest method), each of those tasks has a strictly larger set of ancestral task names, and the size of that set is capped. If n is the number of ground task names, each chain can only grow to a length of $n \cdot (m-1)$. So the acyclic progression bound of problems with totally ordered methods is $|T| \cdot n \cdot (m-1)$. Since the size of any state is also polynomial, totally ordered proposition TIHTN plan existence is PSPACE-complete. □

Theorem 4. *TIHTN planning is* EXPSPACE*-complete for lifted problems with totally-ordered methods.*

Proof. Grounding a lifted domain produces a worst-case exponential increase in the number of task names providing EXPSPACE membership. Lifted classical planning provides the EXPSPACE lower bound (Erol *et al.* 1995). □

4 2^{2^k} Bottles of Beer on the Wall

At most $2^{\mathcal{L}}$ tasks need to be inserted between each pair of two consecutive primitive tasks in any acyclic decomposition (Geier and Bercher 2011). This provides an upper bound on the number of task insertions needed to show a TIHTN problem is solvable. The song "m Bottles of Beer on the Wall" uses a decimal counter to encode a bound on the number of refrains in space logarithmic to m (Knuth 1977). Much like this, we will give transformations of TIHTN operators so that a binary counter ensures strict upper bounds on the number of primitive tasks in any solution. This will limit the length of any sequence of progressions, giving us upper complexity bounds for TIHTN planning.

Clearly, we could just limit acyclic progression to a bounded depth instead. However we will also use the bounding transformation below in the following section to provide a polynomial transformation of acyclic HTN problems which preserves solvability under TIHTN semantics.

Let P be a propositional problem with a set of operators O and language $\mathcal{L}$. Given a bound of the form 2^k, we create a language $\mathcal{L}'$ which contains $\mathcal{L}$ and the following propositions: $counting$, $count\_init$, and $counter_i$ and $decrement_i$ for $i \in \{1 \ldots k+1\}$. We define the operator set O' to be each operator in O with the added the precondition $\neg counting$ and the additional effect $counting \wedge decrement_1$. We define another set of operators O_{count} with the following operators:

- $(count\_init\_op, \neg count\_init, count\_init \wedge counter_k)$, initializing the counter to the value 2^k.
- $(decrement\_i\_0\_op, pre, \mathit{eff})$ for $i \in \{1..k\}$, where:

  $$pre \coloneqq count\_init \wedge decrement_i \wedge \neg counter_i$$
  $$\mathit{eff} \coloneqq \neg decrement_i \wedge decrement_{i+1} \wedge counter_i$$

  setting the ith bit of the counter to 1 if it was zero and moves on to the next bit.
- $(decrement\_i\_1\_op, pre, \mathit{eff})$ for $i \in \{1..k\}$, where:

  $$pre \coloneqq count\_init \wedge decrement_i \wedge counter_i$$
  $$\mathit{eff} \coloneqq \neg decrement_i \wedge \neg counting \wedge \neg counter_i$$

  setting the ith bit of the counter to 0 and stops counting.

Then in the problem P' with operators $O' \cup O_{count}$ and the language $\mathcal{L}'$, any executable sequence consists of an alternating pattern of an operator from O' followed by a sequence of counting operators from O_{count}. Since there is

no operator to decrement $counter_{k+1}$, we can only apply 2^k operators from O' to the state. Notice that by setting the appropriate $counter_i$ propositions in the $count\_init\_op$ operator, we could have expressed any bound between 0 and $2^{k+1} - 1$.

We can extend this to doubly-exponential bounds for lifted problems. Let $\mathcal{P}$ be a lifted problem with language $\mathcal{L}$ and operators $\mathcal{O}$, and let 2^{2^k} be our bound on operators from $\mathcal{O}$ to encode. Let $\mathcal{L}'$ contain $\mathcal{L}$ and the following predicates: $counting()$, $count\_init()$, and $counter\,(v_1, \ldots, v_k)$ and $decrement\,(v_1, \ldots, v_k)$, and let $0, 1$ be arbitrary distinct constants in $\mathcal{L}'$.

As with the $counter_i$ predicates, the ground $counter(\ldots)$ predicates will express a binary counter in the state, with a binary index (the variables) into the exponential number of bits. Let $cr_1 \ldots, cr_k$ be predicates such that each cr_i has the form $counter\,(v_k, \ldots, v_{i+1}, 0, 1, \ldots, 1)$ and let $dec_1 \ldots, dec_k$ be predicates such that each dec_i has the form $decrement\,(v_k, \ldots, v_{i+1}, 0, 1, \ldots, 1)$ where each v_m is a variable. So:

- $dec_1 = decrement\,(v_k, \ldots, v_2, 0)$
- $dec_2 = decrement\,(v_k, \ldots, v_3, 0, 1)$
- $dec_{k-1} = decrement\,(v_k, 0, 1, \ldots, 1)$, and
- $dec_k = decrement\,(0, 1, \ldots, 1)$

Similarly, let $dec'_1, \ldots, dec'_k$ be predicates of the form $decrement\,(v_k, \ldots, v_{i+1}, 1, 0, \ldots, 0)$, so:

- $dec'_1 = decrement\,(v_k, \ldots, v_2, 1)$
- $dec'_2 = decrement\,(v_k, \ldots, v_3, 1, 0)$
- $dec'_{k-1} = decrement\,(v_k, 1, 0, \ldots, 0)$
- $dec'_k = decrement\,(1, 0, \ldots, 0)$

So if v is an assignment of $v_1, \ldots, v_k$ to $\{0, 1\}$ and we view the proposition $dec_i\,[v]$ as an instruction to decrement the jth bit of the counter, then $dec'_i\,[v]$ is for decrementing the bit with index $j + 1$.

Let $\mathcal{O}'$ consist of each operator in $\mathcal{O}$ with the added the precondition $\neg counting()$ and the additional effect $counting() \wedge decrement\,(0, \ldots, 0)$. We define $\mathcal{O}_{count}$ to be the following operators:

- $(count\_init\_op(), pre, \mathit{eff})$, where:
$$pre \coloneqq \neg count\_init()$$
$$\mathit{eff} \coloneqq count\_init() \wedge counter\,(1, 0, \ldots, 0)$$
which initializes the counter to the value 2^{2^k}.
- $(decrement\_i\_0\_op(), pre, \mathit{eff})$ for $i \in \{1..k\}$, where:
$$pre \coloneqq count\_init() \wedge dec_i \wedge \neg cr_i$$
$$\mathit{eff} \coloneqq \neg dec_i \wedge dec_{i+1} \wedge cr_i$$
which, if v is an assignment of $v_1, \ldots, v_k$ to $\{0, 1\}$, sets $cr_i\,[v]$ to 1 if it was zero and moves on to the next bit.
- $(decrement\_i\_1\_op(), pre, \mathit{eff})$ for $i \in \{1..k\}$, where:
$$pre \coloneqq count\_init() \wedge dec_i \wedge cr_i$$
$$\mathit{eff} \coloneqq \neg dec_i \wedge \neg counting() \wedge \neg cr_i$$
which, if v is an assignment of $v_1, \ldots, v_k$ to $\{0, 1\}$, sets $cr_i\,[v]$ to 0 and stops counting.

So after $decrement\,(0, \ldots, 0)$ is set by an operator (and $count\_init\_op()$ has been applied in the past), the only legal sequence of operators involves stepping sequentially through the 2^j possible $counter\,(\ldots)$ predicates until the decrement operation is finished.

Similar to the propositional transformation, we can start the counter at some number which is a polynomial sum of 2^i for $i \in \{1..2^k\}$. These two transformations are the dual of the counting tasks in Theorems 4.1 and 4.2 from Alford *et al.* (2015). Where the counting tasks gave methods that enforced exactly 2^k and 2^{2^k} repetitions of a given task be in any solution, this transformation ensures that there are no more than the specified number of primitive tasks.

Theorem 5. *Propositional TIHTN planning is in* NEXPTIME*; lifted TIHTN planning is in 2-*NEXPTIME.

Proof. Use the appropriate transformation from above to limit the number of primitive operators in any solution to $|T| \cdot m^n \cdot 2^{\mathcal{L}}$, where $|T|$ is the number of tasks in the initial network, m is the max method size, n is the number of non-primitive task names in the grounded problem (exponential for lifted problems), and $2^{\mathcal{L}}$ is the total number unique states expressible by $\mathcal{L}$. This ensures every sequence of acyclic progressions ends after an exponential number of steps for propositional problems and a doubly-exponential number of steps for lifted problems. Thus, a depth-first non-deterministic application of acyclic progression until it reaches a solution or can progress no more is enough to prove the existence of a TIHTN solution. □

5 Acylic HTN Planning with TIHTN Planners

Theorem 5 provides upper bounds for TIHTN planning. Section 2 describes acyclic decomposition. An *acyclic problem* is one in which every sequence of decompositions is acyclic (Erol *et al.* 1996; Alford *et al.* 2012). HTN plan existence for propositional partially ordered acyclic problems is NEXPTIME-complete and 2-NEXPTIME-complete for lifted partially ordered acyclic problems.

Theorem 6.1 of (Alford *et al.* 2015) encodes NEXPTIME- and 2-NEXPTIME-bounded Turing machines almost entirely within the task network of propositional and lifted acyclic HTN problems, respectively. Of particular interest here, though, is that, for a given time bound and Turing machine, every primitive decomposition of the initial task network in these encodings has exactly the same number of primitive tasks. This lets us use the bounding transformation from Section 4 to prevent rogue task insertion under TIHTN semantics:

Theorem 6. *Propositional TIHTN planning is* NEXPTIME*-hard; lifted TIHTN planning is 2-*NEXPTIME*-hard.*

Proof. Let N be a nondeterministic Turing machine (NTM), let $K = 2^{2^k}$ be the time bound for N, and let $\mathcal{P}$ with operators $\mathcal{O}$ be the encoding of N as a lifted acyclic HTN problem.

One can calculate exactly how many primitive tasks are in any decomposition of $\mathcal{P}$, but it is roughly of the form

$B = c \cdot K^2 + d \cdot K + 1$ for constants c and d, which we can express as a polynomial sum of 2^{2^i}. Let $\mathcal{P}'$ be the B-task-bounded transformation of $\mathcal{P}$.

From the hardness proof of Theorem 6.1 of (Alford *et al.* 2015), we know: if N can be in an accepting state after K steps, there is an executable primitive decomposition of $\mathcal{P}$ with B tasks from $\mathcal{O}$, and so $\mathcal{P}'$ has a TIHTN solution.

Let tn be a non-executable primitive decomposition of the initial task network, $tn_I \rightarrow_D^* tn$. Since the bounding transformation does not affect the methods, this decomposition sequence is also legal in $\mathcal{P}'$ (with operators $\mathcal{O}' \cup \mathcal{O}_{count}$). Since any insertions of operators from $\mathcal{O}'$ would put tn over the limit B, no sequence of insertions can make this task network executable in $\mathcal{P}'$. Thus if no run of N is in an accepting state after K steps, no primitive decomposition of $\mathcal{P}$ is executable, and there is no TIHTN solution to $\mathcal{P}'$.

Since $\mathcal{P}'$ has a TIHTN solution iff $\mathcal{P}$ has a solution, and $\mathcal{P}$ encodes a 2^{2^k}-bounded NTM, lifted TIHTN planning is 2-NEXPTIME-hard.

The proof is the same for propositional TIHTN problems using the propositional encoding of 2^k time-bounded NTMs into acyclic HTN problems, so propositional acyclic TIHTN planning is NEXPTIME-hard. □

As a corollary, we obtain NEXPTIME and 2-NEXPTIME completeness for propositional and lifted TIHTN planning, respectively.

6 A comparison with HTN complexity classes

Based on the recursion structure classification for HTN problems (Alford *et al.* 2012), we now have an extensive classification of HTN problems by structure and complexity:

- *Acyclic* problems, discussed earlier, where every decomposition is guaranteed to be acyclic.
- *Tail-recursive* problems, where methods can only recurse through their last task. All acyclic problems are also tail-recursive.
- *Arbitrary-recursive* problems, which includes all HTN problems.

However, by Corollary 1, non-acyclic (i.e., cyclic) decompositions can be ignored, limiting the impact of recursion structure on the complexity of TIHTN planning.

This is not to say that method structure (outside of ordering) has no effect on the complexity of TIHTN planning. For instance, *regular* TIHTN problems (defined by Erol *et al.* (1996) for HTN planning) with a partially ordered initial task network and partially ordered methods are easier than non-regular (partially ordered) problems. Regular problems are a special case of tail-recursive problems, where every method is constrained to have at most one non-primitive task in the method's network, and that task must be constrained to come after all the primitive tasks. Regular problems have a linear progression bound regardless of whether the primitive tasks are totally-ordered amongst themselves. Since acyclic progression is a special case of progression, regular problems have linear acyclic progression bounds, and thus partially ordered regular problems have the same complexity under TIHTN semantics as totally-ordered regular problems:

Table 2: Comparison of the complexity classes for HTN planning (completeness results) from (Alford *et al.* 2015) with our TIHTN planning results (indicated by TI=yes).

Vars.	Ordering	TI	Recursion	Complexity
no	total	no	acyclic	PSPACE
no	total	no	regular	PSPACE
no	total	no	tail	PSPACE
no	total	no	arbitrary	EXPTIME
no	total	yes	–	PSPACE
no	partial	no	acyclic	NEXPTIME
no	partial	no	regular	PSPACE
no	partial	no	tail	EXPSPACE
no	partial	no	arbitrary	undecidable
no	partial	yes	regular	PSPACE
no	partial	yes	–	NEXPTIME
yes	total	no	acyclic	EXPSPACE
yes	total	no	regular	EXPSPACE
yes	total	no	tail	EXPSPACE
yes	total	no	arbitrary	2-EXPTIME
yes	total	yes	–	EXPSPACE
yes	partial	no	acyclic	2-NEXPTIME
yes	partial	no	regular	EXPSPACE
yes	partial	no	tail	2-EXPSPACE
yes	partial	no	arbitrary	undecidable
yes	partial	yes	regular	EXPSPACE
yes	partial	yes	–	2-NEXPTIME

Corollary 7. *TIHTN plan-existence for regular problems is* PSPACE*-complete when they are propositional, and* EXPSPACE*-complete otherwise, regardless of ordering.*

There are also times when HTN planning is *simpler* than TIHTN planning. Alford *et al.* (2014) show that HTN planning for propositional regular problems which are also acyclic is NP-complete. Since an empty set of methods and a single primitive task in the initial network is enough to encode classical planning problems under TIHTN semantics, acyclic-regular problems are still PSPACE-hard for propositional domains and EXPSPACE-hard when lifted.

So, while TIHTN planning is not always easier than HTN planning, we have shown that its complexity hierarchy is, in general, both simpler and less sensitive to method structures. In future work, we want to investigate the plan existence problem along the further axis of syntactic restrictions on the task hierarchy and methods: Alford *et al.* (2015) defines a new restriction on HTN problems, that of *constant-free methods*, that forbids mixing constants and variables in task names appearing in methods. This significantly reduces the progression bound for lifted partially ordered acyclic and tail recursive problems, and thus may also impact the complexity of those problems under TIHTN semantics.

7 Conclusions

We studied the plan existence problem for TIHTN planning, a hierarchical planning framework that allows more flexibility than standard HTN planning. The complexity varies from PSPACE-complete for the totally ordered propositional setting to 2-NEXPTIME-complete for TIHTN planning where variables are allowed and the methods' task networks may be only partially ordered.

We showed that totally ordered TIHTN planning has the same plan existence complexity as classical planning (both with and without variables). Given that plan existence for both delete-relaxed propositional TIHTN and classical problems is in polynomial time (Alford *et al.* 2014), we hope that many of the algorithms and heuristics developed for classical planning can be adapted for totally-ordered TIHTN problems.

We also provided a new planning technique for TIHTN planning, called acyclic progression, that let us define provably efficient TIHTN planning algorithms. We hope it inspires the creation of future planners that are both provably and empirically efficient.

Acknowledgment This work is sponsored in part by OSD ASD (R&E) and by the Transregional Collaborative Research Centre SFB/TRR 62 "Companion-Technology for Cognitive Technical Systems" funded by the German Research Foundation (DFG). The information in this paper does not necessarily reflect the position or policy of the sponsors, and no official endorsement should be inferred.

References

Ron Alford, Ugur Kuter, and Dana S Nau. Translating HTNs to PDDL: A small amount of domain knowledge can go a long way. In *Proc. of the 21st Int. Joint Conf. on Artificial Intelligence (IJCAI)*, pages 1629–1634. AAAI Press, 2009.

Ron Alford, Vikas Shivashankar, Ugur Kuter, and Dana S Nau. HTN problem spaces: Structure, algorithms, termination. In *Proc. of the 5th Annual Symposium on Combinatorial Search (SoCS)*, pages 2–9. AAAI Press, 2012.

Ron Alford, Vikas Shivashankar, Ugur Kuter, and Dana S. Nau. On the feasibility of planning graph style heuristics for HTN planning. In *Proc. of the 24th Int. Conf. on Automated Planning and Scheduling (ICAPS)*, pages 2–10. AAAI Press, 2014.

Ron Alford, Pascal Bercher, and David W. Aha. Tight bounds for HTN planning. In *Proc. of the 25th Int. Conf. on Automated Planning and Scheduling (ICAPS)*. AAAI Press, 2015.

Gregor Behnke, Daniel Höller, and Susanne Biundo. On the complexity of HTN plan verification and its implications for plan recognition. In *Proc. of the 25th Int. Conf. on Automated Planning and Scheduling (ICAPS)*. AAAI Press, 2015.

Pascal Bercher, Shawn Keen, and Susanne Biundo. Hybrid planning heuristics based on task decomposition graphs. In *Proc. of the Seventh Annual Symposium on Combinatorial Search (SoCS)*, pages 35–43. AAAI Press, 2014.

Susanne Biundo and Bernd Schattenberg. From abstract crisis to concrete relief (a preliminary report on combining state abstraction and HTN planning). In *Proc. of the 6th Europ. Conf. on Planning (ECP)*, pages 157–168. AAAI Press, 2001.

Susanne Biundo, Pascal Bercher, Thomas Geier, Felix Müller, and Bernd Schattenberg. Advanced user assistance based on AI planning. *Cognitive Systems Research*, 12(3-4):219–236, April 2011. Special Issue on Complex Cognition.

Kutluhan Erol, James A. Hendler, and Dana S. Nau. UMCP: A sound and complete procedure for hierarchical task-network planning. In *Proc. of the 2nd Int. Conf. on Artificial Intelligence Planning Systems (AIPS)*, pages 249–254. AAAI Press, 1994.

Kutluhan Erol, Dana S. Nau, and V. S. Subrahmanian. Complexity, decidability and undecidability results for domain-independent planning. *Artificial Intelligence*, 76(1):75–88, 1995.

Kutluhan Erol, James A. Hendler, and Dana S. Nau. Complexity results for HTN planning. *Annals of Mathematics and Artificial Intelligence*, 18(1):69–93, 1996.

Thomas Geier and Pascal Bercher. On the decidability of HTN planning with task insertion. In *Proc. of the 22nd Int. Joint Conf. on Artificial Intelligence (IJCAI)*, pages 1955–1961. AAAI Press, 2011.

Daniel Höller, Gregor Behnke, Pascal Bercher, and Susanne Biundo. Language classification of hierarchical planning problems. In *Proc. of the 21st Europ. Conf. on Artificial Intelligence (ECAI)*, pages 447–452. IOS Press, 2014.

Subbarao Kambhampati, Amol Mali, and Biplav Srivastava. Hybrid planning for partially hierarchical domains. In *Proc. of the 15th Nat. Conf. on Artificial Intelligence (AAAI)*, pages 882–888. AAAI Press, 1998.

Donald E. Knuth. The complexity of songs. *SIGACT News*, 9(2):17–24, July 1977.

Naiwen Lin, Ugur Kuter, and Evren Sirin. Web service composition with user preferences. In *Proc. of the 5th Europ. Semantic Web Conference (ESWC)*, pages 629–643, Berlin, Heidelberg, 2008. Springer.

Dana S. Nau, Tsz-Chiu Au, Okhtay Ilghami, Ugur Kuter, J William Murdock, Dan Wu, and Fusun Yaman. SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research*, 20:379–404, 2003.

Dana S. Nau, Tsz-Chiu Au, Okhtay Ilghami, Ugur Kuter, Dan Wu, Fusun Yaman, Héctor Muñoz-Avila, and J. William Murdock. Applications of SHOP and SHOP2. *Intelligent Systems, IEEE*, 20:34–41, March - April 2005.

Vikas Shivashankar, Ron Alford, Ugur Kuter, and Dana Nau. The GoDeL planning system: a more perfect union of domain-independent and hierarchical planning. In *Proc. of the 23rd Int. Joint Conf. on Artificial Intelligence (IJCAI)*, pages 2380–2386. AAAI Press, 2013.

From FOND to Probabilistic Planning: Guiding search for quality policies

Alberto Camacho[†], Christian Muise[*], Akshay Ganeshen[†], Sheila A. McIlraith[†]
[†]Department of Computer Science, University of Toronto
[*]Department of Computing and Information Systems, University of Melbourne
[†]{acamacho,akshay,sheila}@cs.toronto.edu, [*]{christian.muise}@unimelb.edu.au

Abstract

We address the class of probabilistic planning problems where the objective is to maximize the probability of reaching a prescribed goal (MAXPROB). State-of-the-art probabilistic planners, and in particular MAXPROB planners, offer few guarantees with respect to the quality or optimality of the solutions that they find. The complexity of MAXPROB problems makes it difficult to compute high quality solutions for big problems, and existing algorithms either do not scale well, or provide poor quality solutions. We exploit core similarities between probabilistic and fully observable non-deterministic (FOND) planning models to extend the state-of-the-art FOND planner, PRP, to be a sound and sometimes complete MAXPROB solver that is guaranteed to sidestep avoidable dead ends. We evaluate our planner, Prob-PRP, on a selection of benchmarks used in past probabilistic planning competitions. The results show that Prob-PRP outperforms previous state-of-the-art algorithms for solving MAXPROB, and computes substantially more robust policies, at times doing so orders of magnitude faster.

1 Introduction

Planning involves finding action strategies, also called *plans*, that lead an agent to a desired *goal condition*. In scenarios with exogenous events, or where the effects of an agent's actions cannot be accurately modeled, the execution of a plan may not be fully predictable. In the planning community, uncertainty in the outcome of actions is modeled as a variant of the classical planning formalism that incorporates non-deterministic actions. Two similar models can be distinguished: fully observable non-deterministic (FOND) planning, and probabilistic planning. The former assumes fair non-determinism on the potential effects of the actions, while the latter breaks this assumption and assigns a probability distribution over the action outcomes. We address the subclass of probabilistic problems called MAXPROB, whose objective is to find policies that maximize the probability of reaching a prescribed goal (Kolobov et al. 2011).

We reflect on *what makes for a good quality policy*. Whereas optimal solutions to MAXPROB problems maximize the probability of reaching a goal, solutions often have other properties that are also desirable. Further, in contrast to *online* solutions, computing *offline* solutions makes it possible to offer guarantees with respect to the quality of solutions.

The state of the art in MAXPROB planning is the online planner *Robust-FF* (RFF) (Teichteil-Königsbuch, Kuter, and Infantes 2010). RFF is sound and complete in the all-outcomes determinization of problems with no dead ends, but it offers no guarantees with respect to the quality and optimality of the solutions in the presence of dead ends.

We exploit the core similarities between probabilistic and FOND planning models to extend the state-of-the-art FOND planner, PRP, to be a sound *offline* MAXPROB solver, which we call Prob-PRP. Prob-PRP prunes the search space by identifying state-action pairs that lead to undesired states, making it possible to handle large and probabilistically complex MAXPROB problems. The dead end detection mechanism in Prob-PRP guarantees finding optimal solutions in domains where all of the dead ends are avoidable.

We compare Prob-PRP with RFF across a variety of benchmarks from the International Probabilistic Planning Competition (IPPC). The results show that the quality of the policies is better for Prob-PRP across the majority of problems; particularly in domains with avoidable dead ends. Further, the computation time required is at times orders of magnitude faster, despite being offline.

Illustrative Example: The River Problem Consider the *River* problem introduced in (Little and Thiébaux 2007). In this problem, the agent has two options to cross a river: (i) traverse a path of slippery rocks with a 25% chance of success, a 25% chance of slipping and falling into the river, and a 50% chance of reaching a small island. In the latter case, she can swim towards the other side of the river with an 80% chance of success and a 20% chance of drowning; (ii) swim from one side of the river to the other, with a 50% chance of success, and a 50% chance of falling in.

The optimal solution is for the agent to attempt to traverse the rocks, and if she falls and survives, to swim from the island. This policy causes the agent to reach the other side of the river with a 65% probability of success, whereas the greedy action of swimming has only a 50% chance of success. Online replanners (e.g., (Yoon, Fern, and Givan 2007)) may be attracted by the shortest path to the goal (i.e. swimming directly across): once a wrong choice is made, online replanning approaches have no recourse to undo the bad action. This motivates the need for effective offline planning for MAXPROB problems with dead ends.

2 Preliminaries

We adopt the notation of Mattmüller et al. (2010) and Muise, McIlraith, and Beck (2012) for non-deterministic SAS$^+$ planning problems, extending it to probabilistic planning with conditional effects.

A SAS$^+$ *probabilistic planning problem* is a tuple $\langle \mathcal{V}, s_0, s_\star, \mathcal{A} \rangle$, where $\mathcal{V}$ is a finite set of variables v, each with domain $\mathcal{D}_\mathcal{V}$. We denote $\mathcal{D}^+_\mathcal{V}$ to be the extended domain that includes the undefined status, $\bot$, of v. A *partial state* (or simply, a state) is an assignment to variables $s : \mathcal{V} \to \mathcal{D}^+_\mathcal{V}$. If $s(v) \neq \bot$, we say that v is *defined* for s. When every variable is defined for s we say that s is a *complete* state. The *initial state* s_0 is a complete state, and the *goal state* $s_\star$ is a partial state. A state s *entails* a state s', denoted $s \models s'$, when $s(v) = s'(v)$ for all v defined for s'. The state obtained from *applying* s' to s is the *updated* state $s \oplus s'$ that assigns $s'(v)$ when v is defined for s', and $s(v)$ otherwise.

The actions $a \in \mathcal{A}$ have the form $a = \langle Pre_a, \mathit{Eff}_a \rangle$, where Pre_a is a state describing the condition that a state s needs to entail in order for a to be applicable in s. $\mathit{Eff}_a = \langle (p_1; \mathit{Eff}^1_a), \ldots, (p_n; \mathit{Eff}^n_a) \rangle$ is a finite set of tuples $(p_i; \mathit{Eff}^i_a)$ where $\Sigma_i p_i = 1$. Each *effect* Eff^i_a is a set of the form $\{\langle \mathit{cond}_1, v_1, d_1 \rangle, \ldots, \langle \mathit{cond}_k, v_k, d_k \rangle\}$, where for each j the *condition* cond_j is a partial state, and $v_j \in \mathcal{V}$, $d_j \in \mathcal{D}_{\mathcal{V}_i}$. The result of applying the action a in the partial state $s \models Pre_a$ with effect Eff^i_a is the partial state $\mathit{Result}(s, \mathit{Eff}^i_a) = \{v = d \mid \langle \mathit{cond}, v, d \rangle \in \mathit{Eff}^i_a \text{ and } s \models \mathit{cond}\}$. Finally, the *progression* of s w.r.t. action a and effect Eff^i_a is the updated state $Prog(s, a, \mathit{Eff}^i_a) = s \oplus \mathit{Result}(s, \mathit{Eff}^i_a)$.

An action a is applicable in a partial state s when $s \models Pre_a$. With a probability p_i, the outcome of a is one of the effects Eff^i_a that leads to the updated state $s' = Prog(s, a, \mathit{Eff}^i_a)$. The term (s, a, s') is called a *transition*, and has associated *transition probability* $T(s, a, s') = p_i$.

A solution to a probabilistic planning problem is a policy that maps (partial) states into actions with the objective of reaching a state that entails $s_\star$. Given a policy π, we denote S_π the set of states in which π is defined. We say that π is *well defined* when $\pi(s)$ is applicable in s for all $s \in S_\pi$. A policy is *closed* when it is defined in all states, and otherwise it is said to be a *partial* policy. A well-defined policy defines sequences of state-action trajectories $s_0, a_0, s_1, a_1 \ldots$ where $\pi(s_k) = a_k$, and $s_{k+1} = Prog(s_k, a_k, \mathit{Eff})$ for some $\mathit{Eff} \in \mathit{Eff}_{a_k}$. A sequence $P = a_0, a_1, \ldots$ of such actions is called a *plan*. We associate $P(s_i)$ with the action a_i.

We address the class of probabilistic planning problems where the objective is to maximize the probability of reaching a state that entails $s_\star$ (MAXPROB). We refer to this as the *probability of success*. The probability of success ignores action costs, and focuses on goal-achievement. A policy is optimal when its probability of success is maximal.

2.1 Related Planning Problems

A SAS$^+$ *Fully Observable Non-Deterministic* (FOND) planning problem is likewise described as a tuple $\langle \mathcal{V}, s_0, s_\star, \mathcal{A} \rangle$. Nevertheless, in contrast to a probabilistic planning problem, the FOND model assumes fair nondeterminism with respect to the effects of the actions. As such, action effects are assumed to be equally likely, so there are no probabilities, p_i associated with the effects of actions (Muise, McIlraith, and Beck 2012). A solution to a FOND problem is a policy, usually distinguished according to the criteria introduced in (Cimatti et al. 2003). *Weak solutions* are plans that reach the goal under some sequence of nondeterministic action outcomes, and *strong solutions* are policies that are guaranteed to reach the goal in a bounded number of transitions. Finally, *strong cyclic solutions* are policies that are guaranteed to eventually reach the goal, under an assumption of fairness – effectively that in the limit, each action will yield each of its non-deterministic outcomes infinitely often.

Markov Decision Processes (MDPs) are another probabilistic planning model. The traditional MDP model introduced in Puterman (1994) associates rewards to state transitions. MAXREWARD MDPs have either a finite horizon or apply a discount factor $\gamma < 1$, and the solutions attempt to maximize the expected reward. Stochastic Shortest Path (SSP) MDPs (Bertsekas and Tsitsiklis 1991) is a class of goal-oriented probabilistic planning problems. Their solutions attempt to minimize the expected plan length under the assumption that the goal state is reachable from any state. A MAXPROB problem can be translated into a goal-oriented MDP with infinite horizon and discount factor $\gamma = 1$ by fixing the rewards to 1 in the goal states, and zero elsewhere (cf. (Kolobov, Mausam, and Weld 2012))

2.2 State of the Art in MAXPROB Planning

The International Probabilistic Planning Competition (IPPC) tests the performance of probabilistic planners periodically. It is notable that none of the winners of past IPPCs was an offline planner. As Little and Thiébaux (2007) point out, one of the reasons why online planners have outperformed offline planners is that the latter simply cannot handle large instances of complex problems.

The first IPPCs (IPPC-04 and IPPC-06) were dedicated to solving MAXPROB problems. The benchmark domains used in these competitions were probabilistically simple (Little and Thiébaux 2007), and a planner without probabilistic reasoning entitled *FF-Replan* (Yoon, Fern, and Givan 2007) won the IPPC-04 edition, and outperformed the participants of the IPPC-06 with only minor changes.

The IPPC-08 adopted more complex benchmark domains and focused on solving MAXREWARD probabilistic problems. The winner of the IPPC-08 was, with minor modifications, the MAXPROB planner *Robust-FF* (RFF) (Teichteil-Königsbuch, Kuter, and Infantes 2010). The IPPC-2011 and IPPC-2014 focused on solving MAXREWARD MDPs rather than goal-oriented probabilistic planning problems. The winner of these competitions, PROST (Keller and Eyerich 2012), works well for reward-based problems, but its performance suffers somewhat in goal-oriented settings such as MAXPROB. To the best of our knowledge, RFF is the state of the art for solving MAXPROB problems.

RFF computes plans in the determinization of the problem that are used to extend an *envelope* gradually until the esti-

mated probability of failure during execution is lower than a threshold parameter ρ. A failure during execution is understood as either falling into a deadend state, or falling into a state s that is unhandled by the envelope (in which case, RFF replans from s). The first envelope computed by RFF is a deterministic plan returned by a call to FF (Hoffmann and Nebel 2001); for each reachable state s not considered in the current envelope, RFF computes a deterministic plan for s and keeps iterating until the estimated probability of failure for the policy in the envelope is lower than ρ.

RFF offers optional modes of operation that have the potential to improve its performance. The *best-goals* strategy computes the set of states handled by the envelope that are the most likely to be reached during execution. Then FF is used to search for plans that lead to the goal, or to states in the best-goals set. Another mode performs *policy optimization* during the search process. Using dynamic programming, the policy is updated in each state to minimize the expected cost, assuming the following: for each transition (s, a, s'), the cost of a is 0 when s' is a goal state; a fixed penalty $1/(1-\gamma)$ when s' is a terminal state; and 1 otherwise. Alternatively, RFF has a mode for solving MAXREWARD problems that performs policy optimization and uses the actual transition costs given in the specification of the problem[1].

The policy optimization mechanism is an effort to avoid falling into terminal states, but it prioritizes the search for plans that reach a goal state in the lowest number of state transitions possible (i.e. stochastic shortest paths (Bertsekas and Tsitsiklis 1991)) that are not necessarily the best quality MAXPROB solutions. As an example, suppose that RFF explores two different plans from state s: (i) the first plan π_1 has a short path to the goal, but may also fall into a dead end; (ii) the second plan π_2 has no dead ends, but all paths to the goal are very long. Then, for sufficiently long paths in π_2, the policy optimization process in s selects π_1 over π_2 even though the latter policy is of higher quality.

RFF is sound and complete in the all-outcomes determinization of problems with no dead ends, but it offers no guarantees with respect to the optimality of the solutions found in the most probable determinization – even in problems without dead ends.

3 Quality of Solutions

The quality of the solutions to MAXPROB planning problems has been traditionally measured according to the probability of success, i.e., the probability of reaching the goal eventually. In this section we discuss the importance of other properties that account for *good* solutions – namely, the size of the policies and the expected length of the plans. To this point, there is no uniform theory of utility elicitation for solutions to MAXPROB problems. This paper does not intend to build that theory, but rather present an algorithm that finds MAXPROB solutions that maintain a good balance between the policy size and the expected plan length.

[1]Personal correspondence with Florent Teichteil-Königsbuch.

3.1 Policy Size

The FOND community has pursued policies that are *small* in the number of state-action pairs, and *compact* in so far as the policies can exploit partial state representations to capture a family of states. An obvious advantage of small, compact policies is that they are easily integrated into simple systems. While the FOND model often assumes the existence of either strong or strong cyclic solutions, core methods for obtaining small compact FOND policies can be applied to find solutions to probabilistic planning problems.

3.2 Expected Plan Length

Policy search should take into account the state transition probabilities, so that the expected length of the resulting plans do not become unnecessarily large.

Relevant work has been done to find stochastic shortest path solutions to SSP MDPs (e.g. (Bonet and Geffner 2003; Trevizan and Veloso 2012)). Teichteil-Königsbuch (2012) proposed the class of *Stochastic Safest and Shortest Path* (S^3P) problems, that generalizes SSP MDPs and allows for unavoidable dead ends. Solutions to S^3P MDPs attempt to minimize the expected length of the plans and maximize the probability of success. Unfortunately, this model is not completely solved yet. Kolobov, Mausam, and Weld (2012) distinguishes the class of SSP MDPs with unavoidable dead ends, and presents a new family of heuristic search algorithms, FRET (Find, Revise, Eliminate Traps). Work in (Teichteil-Königsbuch, Vidal, and Infantes 2011) extends the classical planning heuristics h_{add} and h_{max} (Bonet and Geffner 2001) into heuristics for SSP MDPs with dead ends that, when plugged into graph-based MDP search algorithms, achieve competitive results compared to RFF. Unfortunately, none of these algorithms offer guarantees of optimality in problems with unavoidable dead ends.

3.3 Example

As an illustrative example, consider the controllers C_1 and C_2 in Figure 1, representing the state transitions of two policies, π_1 and π_2 respectively, that solve a MAXPROB problem $\mathcal{P}$. The nodes represent states and the arrows describe the state transitions, that may be non-deterministic (e.g., the action $\pi_1(s_0) = \pi_2(s_0)$ maps s_0 into s_1 or s_g with a certain probability distribution). Both solutions are strong cyclic and map s_0 into the goal s_g eventually, but only π_1 is strong and reaches the goal in a bounded number of state transitions. The expected length of the plans increases with the transition probability $T(s_0, \pi_2(s_0), s_1)$. For a sufficiently high value, π_1 may be preferred to π_2 despite the fact that it has a greater size. Furthermore, π_1 may always be preferred because the length of its plans are bounded.

4 Approach

Our contribution is to bring state-of-the-art FOND planning technology to probabilistic planning. We extend the FOND planner, PRP (Muise, McIlraith, and Beck 2012), to solve goal-oriented probabilistic planning problems, and we aim to maximize the probability of success (i.e. computing a solution to the MAXPROB problem).

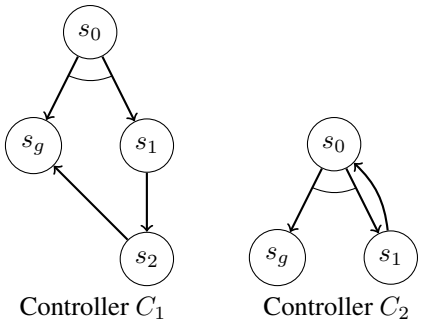


Figure 1: Both solutions map the initial state s_0 into the goal state s_g eventually, but the size of the policies and the expected length of the plans is different.

4.1 Background

To the best of our knowledge, PRP is the state of the art in FOND planning. PRP searches for strong cyclic plans in a FOND problem, and produces a policy that maps partial states to actions. Key components of PRP include a mechanism to evade avoidable dead ends, and a compact representation of the policy in the form of state-action pairs (p, a) that tell the agent to perform the action a in a state s when s entails the condition p.

PRP runs a series of calls to Algorithm 1 until a strong cyclic solution is found, or the algorithm converges. In all cases, PRP returns the best quality policy found. The *Seen* and *Open* lists manage the states that belong to the incumbent policy. More precisely, the *Seen* list contains the states that have been processed already, whereas the *Open* list, initialized to the initial state s_0, contains the states that need to be processed. In each iteration, a state s from the *Open* list is processed. The procedure GENPLANPAIRS processes a non-goal state s for which a policy is undefined. This involves (i) computing a plan P for the all-outcomes determinization of $\langle \mathcal{V}, s, s_\star, \mathcal{A} \rangle$, such that P reaches the goal or a state handled by the policy , and (ii) augmenting the policy with the state-action pairs from the regression of P. The action $P(s)$ is added as a rule to the corresponding state-action pair in s. Processing a non-goal state s that is not in the *Seen* list involves adding all potential successors of s by $P(s)$ into the *Open* list, so that every reachable state of the policy is eventually processed.

Algorithm 1 is guaranteed to find a strong cyclic plan when the problem has no dead ends. When the problem has dead ends, GENPLANPAIRS may not find a plan for a certain state s. In that case, PROCESSDEADENDS computes a minimal partial state p such that $s \models p$ and every $s' \models p$ is a dead end. A set of forbidden state-action pairs is computed by regressing the dead ends through the computed plans. In the next calls to Algorithm 1, PRP resets the policy and the forbidden state-action pairs are excluded from search. The forbidden state-action pairs mechanism ensures completeness of the planner when the dead ends are avoidable.

Theorem 1 ((Muise, McIlraith, and Beck 2012))**.** *PRP returns a strong cyclic policy in problems with avoidable or no dead ends.*

Input: FOND planning task $\mathcal{P} = \langle \mathcal{V}, s_0, s_\star, \mathcal{A} \rangle$
Output: Partial policy π

```
1  InitPolicy();
2  while π changes do
3      Open ← {s_0} ;
4      Seen ← {} ;
5      while Open ≠ ∅ do
6          s = Open.pop();
7          if s ⊭ s_⋆ ∧ s ∉ Seen then
8              Seen.add(s);
9              if π(s) is undefined then
10                 GenPlanPairs(⟨V, s, s_⋆, A⟩, π);
11             if P(s) is defined then
12                 ⟨p, a⟩ = π(s);
13                 for e ∈ Eff_a do
14                     Open.add(Prog(s, a, e));
15     ProcessDeadends();
16 return π;
```

Algorithm 1: Generate Strong Cyclic Plan

The state-action avoidance, and the succinct states representations in PRP demonstrate an improved performance over existing state-of-the-art FOND planners (Muise, McIlraith, and Beck 2012). More recently, PRP was extended for use in FOND problems with conditional action effects (Muise, McIlraith, and Belle 2014) thus extending the class of domains that PRP can solve.

Similarly to RFF, Prob-PRP constructs a robust policy gradually by exploring weak plans. However, PRP does not store a complete representation of the states, but the portion of the states relevant to the policy computed via regression. The partial state representation enhances the possibilities of GENPLANPAIRS to generate a plan for a (partial) state already handled by the policy, providing substantial savings over approaches that use the complete state. Forbidden-state pairs are an effective mechanism to avoid dead ends. Such a mechanism is non-existent in RFF, leading to few alternatives to avoid dead ends already added to the envelope.

4.2 From FOND to MAXPROB

Given a MAXPROB problem $\mathcal{P}$, consider the FOND problem FOND($\mathcal{P}$) that results from ignoring the transition probabilities in $\mathcal{P}$. The strong cyclic solutions π in FOND($\mathcal{P}$) are certainly well-defined policies in $\mathcal{P}$ because the conditions $s \models Pre_{\pi(s)}$ hold equally in $\mathcal{P}$ and in FOND($\mathcal{P}$). Furthermore, all (fair) executions of π in FOND($\mathcal{P}$) eventually reach the goal. Therefore, all executions of π in $\mathcal{P}$ reach the goal with probability 1, independently of the probability distribution on the action effects, and π is also optimal in $\mathcal{P}$. Conversely, if π is an optimal policy in $\mathcal{P}$ that reaches the goal with probability 1, then every reachable state s has a weak plan that leads to the goal also in FOND($\mathcal{P}$), because π is well defined in the FOND problem. Therefore, π is a strong cyclic solution in FOND($\mathcal{P}$).

Lemma 1. *Let $\mathcal{P}$ be a probabilistic planning problem. If π is a well-defined policy in FOND($\mathcal{P}$), then π is a well-defined policy in $\mathcal{P}$.*

Lemma 2. *Let $\mathcal{P}$ be a MAXPROB planning problem. If π is a strong cyclic solution for FOND($\mathcal{P}$), then π is an optimal solution for $\mathcal{P}$ that reaches the goal with probability* 1.

Lemma 3. *A MAXPROB planning problem $\mathcal{P}$ has an optimal solution that reaches the goal with probability* 1 *iff FOND($\mathcal{P}$) has a strong cyclic solution.*

All strong cyclic solutions in FOND($\mathcal{P}$) are equally optimal to the MAXPROB problem $\mathcal{P}$. The similarities between the FOND and MAXPROB formalisms, as well as the equivalence of their solutions under certain conditions, suggest that existing approaches in FOND planning can be adapted to solve MAXPROB problems.

4.3 From PRP to Prob-PRP

The compact representation of states in PRP makes it possible to find small policies for FOND problems. We take advantage of the core similarities between FOND and MAXPROB formalisms, and define Prob-PRP as an extension of PRP that finds MAXPROB solutions to probabilistic planning problems and provides limited optimality guarantees.

Definition 1 (Plain Prob-PRP)**.** *The plain* Prob-PRP *algorithm on a MAXPROB problem $\mathcal{P}$ is a call to PRP on the mapped problem FOND($\mathcal{P}$).*

The soundness of PRP (Muise, McIlraith, and Beck 2012), and the result of Lemma 1 guarantee that the solutions found by PRP in FOND($\mathcal{P}$) are in fact well defined solutions for $\mathcal{P}$ (Lemma 4). PRP is guaranteed to find a strong cyclic solution to FOND($\mathcal{P}$) whenever one exists (Theorem 1), and FOND($\mathcal{P}$) has a strong cyclic solution whenever a proper plan exists for $\mathcal{P}$ (Theorem 3). Therefore, PRP is guaranteed to find a a strong cyclic solution to FOND($\mathcal{P}$) iff a plan with probability of success 1 exists for $\mathcal{P}$. The last condition occurs in domains with avoidable or no dead ends.

Lemma 4. *The plain Prob-PRP algorithm is sound.*

Theorem 2. *The plain Prob-PRP algorithm is guaranteed to find an optimal solution to MAXPROB problems with avoidable or no dead ends.*

Corollary 1. *Given a MAXPROB problem $\mathcal{P}$, if the plain Prob-PRP algorithm returns a solution where the probability of reaching a goal state is less than 1, then $\mathcal{P}$ has unavoidable dead ends.*

The MAXPROB solutions found by Prob-PRP are computed offline, with the advantage that no further computation is needed during execution. Computing offline solutions makes it possible to estimate the quality prior to execution – e.g. using Monte Carlo simulations as done in Prob-PRP– or even to compute it analytically. A consequence of Theorem 2 is that, if Prob-PRP returns a solution to $\mathcal{P}$ whose probability of success is lower than 1, then the problem $\mathcal{P}$ necessarily has unavoidable dead ends (Corollary 1).

4.4 Towards Better Quality Solutions

We propose two mechanisms that extend the plain version of Prob-PRP and potentially improve the quality of the solutions while maintaining the soundness of the solutions and the validity of Theorem 2.

Full Exploration in Last Iteration The forbidden state-action pairs mechanism in PRP may reduce the size of the search space dramatically, and improve the efficiency of the algorithm in searching for strong cyclic plans (Muise, McIlraith, and Beck 2012). This mechanism is also useful in solving MAXPROB problems with avoidable dead ends, but the search remains incomplete in domains with unavoidable dead ends. More precisely, when a state-action pair leads recognizably to a dead end, it is forbidden from successive searches. That direction in the search, however, may still lead to a goal state with non-zero probability.

Based on the previous observation, Prob-PRP performs a final iteration, where the best incumbent policy is used to initialize the policy on line 1 of Algorithm 1, and the problem is solved *with forbidden state-action pairs and dead end detection disabled*. The returned policy handles a superset of the states that it was able to previously, thus improving the quality of the solution. This final pass optimistically closes every *Open* state in a best effort manner.

Exploring Most Likely Plans Non-deterministic planning algorithms, like RFF or PRP, extend the policy under construction with plans that map unhandled states to any state that has been previously handled by the policy. This mechanism benefits creation of smaller policies, but the subsequent plans to the goal may be unnecessarily large, as the previous example illustrates. In order to reduce this effect, Prob-PRP skews the search towards short plans that have high likelihood. Previously, this method has been used to search plans in the determinization relaxation that minimise the risk of failing (c.f. (Jimenez, Coles, and Smith 2006)). Formally, the *likelihood* of a state-action plan $P = s_0, a_0, s_1, a_1, \ldots, a_{n-1}, s_n$ is defined as the product:

$$L_P = \Pi_{i=0}^{n-1} T(s_i, a_i, s_{i+1})$$

The likelihood L_P measures how probable it is that the sequence of states $s_0, s_1, \ldots, s_n$ are reached when the stochastic plan $a_0, a_2, \ldots, a_{n-1}$ is executed. In certain domains, like *triangle-tireworld* or *climber*, the most probable outcome of the actions correspond to the desired effects that lead to the goal situation (Little and Thiébaux 2007). A priori, in these kinds of domains it seems reasonable to give preference to exploring the deterministic plans that maximize the likelihood. Loopy or unnecessarily large plans that belong to the policy have necessarily lower likelihood than alternative shorter plans. Subsequently, the expected length of the plans is potentially low.

Prob-PRP modifies the search process in GENPLAN-PAIRS performed by PRP, so that the plans that maximize the likelihood function are given preference to be explored. Since maximizing L_P is equivalent to maximizing the *log-likelihood* $l_P := log(L_P) = \Sigma_{i=0}^{n-1} log(T(s_i, a_i, s_{i+1}))$, and the latter expression offers a clear computational advantage, Prob-PRP maximizes the log-likelihood of the plans instead.

5 Evaluation

We compared the performance of Prob-PRP with RFF in a selection of benchmark problems from past IPPC competitions. The problems are described using standard PPDDL files with probabilistic outcomes and occasionally with conditional action effects (Fox and Long 2003). When required, the goal was modified to request MAXPROB solutions. We used the client-server architecture MDPSim – used in past editions of the IPPC – to simulate the execution of the solutions produced by each planner. All experiments were conducted on a Linux PC with an Intel Xeon W3550 CPU @3.07GHz, limiting the memory usage of each process to 2GB and the run time to 30 minutes.

The probabilistic planner Prob-PRP is implemented as an extension of the FOND planner PRP, and inherits the capability to handle problems with universally quantified formulas and conditional effects. We used the configuration of RFF that reported best results in (Teichteil-Königsbuch, Kuter, and Infantes 2010), namely, the most probable outcome determinization (so that the planner can scale to handle larger instances), and the *best goals* goal-selection strategy with policy optimization enabled. We fixed the probability of failure threshold ρ to 0.2, a number that results in good policy success rates without compromising the run times.

The results of the tests are shown in Table 1. The solution to each problem was run 100 times. For each planner, we report the percentage of successful runs, the average length of the successful plans, the average size of the policies, and the run times – that include only the computation time spent on generating the policy, and exclude the domain pre-processing time (usually negligible). For RFF, the average number of replans during successful runs needed by RFF is also reported, where the computation of the initial policy is counted as a replan.

5.1 General Analysis

Both Prob-PRP and RFF algorithms perform Monte Carlo simulations to estimate the probability of success of the policies found. We set the number of Monte Carlo samples to 1000, a number that does not compromise the execution time significantly and should be, in principle, high enough to provide sufficiently accurate estimations. We found that the cutoff mechanism in RFF is insufficient to compute reliable policies with sufficient guarantees. Indeed, the actual failure rate of the initial policy obtained by RFF in *boxworld-p02* and *boxworld-p12* is nearly 50% – clearly higher than ρ – suggesting that the probability of failure estimated by RFF is not accurate even with 1000 Monte Carlo samples.

The average number of RFF replans with $\rho = 0.2$ is lower than 2 in most of the problems. In practice, decreasing ρ results in a lower number of replans, but the run times increase significantly while the success rate of the solutions don't. The run times of Prob-PRP are comparable or lower than those of RFF with $\rho = 0.2$. More precisely, in many problems the run time needed by Prob-PRP to compute an offline solution is lower than the time needed by RFF to compute even the initial policy.

The size of the solutions found by Prob-PRP are comparable to those found by RFF, but it seems that Prob-PRP scales better in some domains. The compact state representation, and the forbidden-state action mechanisms in Prob-PRP prove efficient in the *triangle-tireworld* domain, leading to solutions that are orders of magnitude smaller, and computed orders of magnitude faster compared to RFF.

	RFF					Prob-PRP			
Problem	%	L	S	T	R	%	L	S	T
blocksworld-p01	**100**	23,2	18,0	0,02	1,00	**100**	**20,9**	**17**	**0,00**
blocksworld-p02	**100**	22,1	18,0	**0,02**	1,00	**100**	**20,8**	**17**	**0,02**
blocksworld-p03	**100**	22,6	18,0	0,02	1,00	**100**	**20,8**	**17**	**0,00**
blocksworld-p04	**100**	22,4	18,0	**0,02**	1,00	**100**	**20,9**	**17**	**0,02**
blocksworld-p05	**100**	64,9	60,6	0,72	1,01	**100**	**50,0**	**43**	**0,16**
blocksworld-p06	**100**	64,3	59,9	0,69	1,00	**100**	**50,6**	**43**	**0,16**
blocksworld-p07	**100**	64,3	60,6	0,69	1,01	**100**	**49,5**	**43**	**0,16**
blocksworld-p08	**100**	64,5	60,0	0,69	1,00	**100**	**50,0**	**43**	**0,16**
blocksworld-p09	**100**	**40,5**	**38,0**	0,67	1,00	**100**	68,4	61	**0,46**
blocksworld-p10	**100**	**41,2**	**39,1**	0,68	1,03	**100**	68,8	61	**0,46**
blocksworld-p11	**100**	**42,4**	**38,7**	0,66	1,02	**100**	68,6	61	**0,46**
blocksworld-p12	**100**	**41,7**	**38,4**	0,68	1,01	**100**	68,4	61	**0,46**
blocksworld-p13	0	∞	117	16,5	1,00	**100**	125	**107**	**1,38**
blocksworld-p14	0	∞	117	16,6	1,00	**100**	**125**	**107**	**1,40**
blocksworld-p15	0	∞	117	16,6	1,00	**100**	**125**	**107**	**1,38**
boxworld-p01	**100**	**29,4**	**49,3**	0,43	1,27	**100**	31,3	57	**0,06**
boxworld-p02	**100**	**29,3**	**49,2**	0,43	1,26	**100**	31,4	57	**0,06**
boxworld-p03	**100**	**29,0**	**47,9**	0,38	1,24	**100**	31,4	57	**0,06**
boxworld-p04	**100**	39,4	**75,7**	1,73	1,31	**100**	**38,6**	105	**0,24**
boxworld-p05	**100**	39,3	**80,9**	1,77	1,38	**100**	**38,5**	105	**0,24**
boxworld-p06	**100**	**64,9**	**166**	13,0	1,35	**100**	68,1	266	**2,34**
boxworld-p07	**100**	**64,5**	**160**	13,0	1,29	**100**	68,1	266	**2,32**
boxworld-p08	**100**	64,6	**125**	7,5	1,30	**100**	**62,4**	207	**1,82**
boxworld-p09	**100**	64,7	**132**	7,56	1,38	**100**	**62,5**	207	**1,84**
boxworld-p10	**100**	**74,3**	**199**	22,9	1,37	**100**	102	415	**17,2**
boxworld-p11	**100**	**73,3**	**183**	22,3	1,27	**100**	102	415	**17,9**
boxworld-p12	**100**	**74,1**	**199**	23,3	1,36	**100**	102	415	**18,0**
boxworld-p13	0	∞	344	35,6	1,94	**100**	178	906	130
boxworld-p14	0	∞	325	34,9	1,83	**100**	177	906	157
boxworld-p15	0	∞	347	35,2	1,96	**100**	177	906	160
ex-blocksworld-p01	60	8,0	20,8	0,05	1,06	**100**	**8,0**	**9**	**0,00**
ex-blocksworld-p02	28	12,0	36,8	0,11	1,16	**54**	14,0	**15**	**0,02**
ex-blocksworld-p03	38	10,0	31,8	0,10	1,14	**60**	**10,0**	**12**	0,12
ex-blocksworld-p04	52	14,0	49,5	0,09	1,13	**59**	32,9	**18**	**0,06**
ex-blocksworld-p05	**100**	**6,0**	11,6	**0,01**	1,09	**100**	**6,0**	**11**	0,02
ex-blocksworld-p06	90	12,6	62,2	0,10	1,35	**96**	20,7	**28**	0,34
ex-blocksworld-p07	60	12,0	36,2	0,20	1,12	**100**	**12,0**	**21**	**0,04**
ex-blocksworld-p08	7	24,0	68,9	0,64	1,20	**36**	30,0	**32**	**0,38**
ex-blocksworld-p09	13	25,2	95,7	1,07	1,23	–	–	–	t
ex-blocksworld-p10	2	36,0	76,8	0,97	1,24	**3**	116	105	14,3
ex-blocksworld-p11	**13**	**32,0**	92,7	**1,59**	1,31	**13**	93,4	**82**	7,42
ex-blocksworld-p12	1	38,0	96,6	2,15	1,21	**2**	91,5	**78**	6,28
ex-blocksworld-p13	**10**	**59,2**	**451**	**5,76**	1,45	–	–	–	t
ex-blocksworld-p14	**0**	**0**	**130**	**116**	1,24	–	–	–	t
ex-blocksworld-p15	**9**	**43,6**	**172**	**8,91**	1,28	–	–	–	t
schedule-p02	**100**	59,2	**5,00**	**0,01**	1,00	**100**	**51,0**	7	0,04
schedule-p03	**100**	100	**5,00**	**0,01**	1,00	**100**	**95,0**	7	0,12
schedule-p04	96	57,8	14,3	0,02	1,12	**100**	**46,9**	21	0,14
schedule-p05	89	116	14,5	0,03	1,15	**100**	**92,0**	16	0,18
schedule-p06	**45**	**364**	**141**	**1,42**	3,01	–	–	m	–
schedule-p07	**36**	**390**	**146**	**1,34**	3,11	–	–	m	–
schedule-p08	**34**	**354**	**146**	**3,94**	3,17	–	–	m	–
schedule-p09	**4**	**402**	**317**	**3,17**	4,31	–	–	m	–
triangle-tireworld-p01	**100**	**5,5**	22,7	0,02	1,07	**100**	**5,5**	**10**	**0,00**
triangle-tireworld-p02	**100**	13,2	80,7	0,17	1,32	**100**	**12,0**	**23**	**0,00**
triangle-tireworld-p03	**100**	21,8	135	0,66	1,13	**100**	**18,5**	**38**	**0,02**
triangle-tireworld-p04	**100**	29,6	248	1,76	1,20	**100**	**25,1**	**55**	**0,06**
triangle-tireworld-p05	**100**	37,6	348	3,76	1,12	**100**	**31,5**	**74**	**0,10**
triangle-tireworld-p06	**100**	45,6	490	7,98	1,14	**100**	**37,9**	**95**	**0,22**
triangle-tireworld-p07	**100**	53,4	714	17,4	1,21	**100**	**44,5**	**118**	**0,42**
triangle-tireworld-p08	**100**	61,5	958	36,5	1,19	**100**	**50,9**	**143**	**0,72**
triangle-tireworld-p09	**100**	69,5	1222	64,1	1,20	**100**	**57,5**	**170**	**1,40**
triangle-tireworld-p10	**100**	77,6	1595	111	1,21	**100**	**64,0**	**199**	**2,38**

Table 1: Performance of RFF and Prob-PRP. % indicates the percentage of successful executions; S indicates the size of the policy; T indicates run-time, in seconds and R indicates the number of replans in RFF. Dash (–) indicates that the experiments ran out of time (t) or memory (m).

5.2 Analysis of the Probability of Success

We split the analysis of the probability of success between problems without dead ends, problems with avoidable dead ends, and problems with non-avoidable dead ends.

Problems without Dead Ends Prob-PRP solves all the problems without dead ends in lower run times than RFF. The probabilistic reasoning overhead performed by Prob-PRP in the *blocksworld*, a simple domain without dead ends, does not translate into bigger run times than those in RFF.

We identified a looping behaviour in the solutions found by RFF to *blocksworld-p13*, *blocksworld-p14*, and *blocksworld-p15*, most likely originating in the policy optimization mechanism. The *problem-goals* strategy does not make use of the policy optimization mechanism, but fails to scale up to handle big problem instances (Teichteil-Königsbuch, Kuter, and Infantes 2010). We also found that the cutoff mechanism in RFF is insufficient to compute reliable policies with sufficient guarantees. Indeed, the actual success rate obtained in *boxworld-p04* and *boxworld-p12* is significantly lower than the threshold $1 - \rho$, suggesting that the probability of failure estimated by RFF is not accurate.

Problems with Avoidable Dead Ends The *triangle-tireworld*, introduced in (Little and Thiébaux 2007), requires a car to drive to a goal location via a number of intermediate locations. During each move, there is a possibility of getting a flat tire, and only some locations have spare tires. It is a challenging domain because both the size of the state space and the number of deadend states increase exponentially with the number of variables in the problem. We used the variant in which the car cannot carry a tire. The forbidden state-action mechanism makes it possible to identify the causes that lead to dead end states, and reduce the size of the state space significantly. Prob-PRP is able to optimally solve big instances of the *triangle-tireworld* domain orders of magnitude faster than RFF.

Problems with Unavoidable Dead Ends Neither RFF nor Prob-PRP offers guarantees on the optimality of the solutions in problems with unavoidable dead ends. As discussed earlier, RFF has a poor mechanism for *backtracking* once a bad plan that leads to a dead end has been explored. The forbidden state-action pairs mechanism and the full exploration in the last iteration performed by Prob-PRP makes it possible to improve the quality of the solutions relative to RFF in the *exploding-blocksworld* and *schedule* domains.

We again detected a looping behavior in some of the solutions given by RFF to the *exploding-blocksworld* problems. Prob-PRP exceeds the time limit (t) of 30 minutes in four instances. In *ex-blocksworld-p09*, this is not to the run-time itself, but to the time needed to decode and process the policy in a manageable format to be used by MDPSim, thus maintaining dead end avoidance. In the other instances, Prob-PRP does not converge within the time limits. Similarly, Prob-PRP exceeded the memory limits (m) in large instances of the *schedule* domain before convergence of the algorithm. These issues will be fixed by enabling an anytime mode in Prob-PRP, making it possible to output the best incumbent policy before the time or memory limits are reached.

5.3 Analysis of the Expected Length of the Plans

The policy optimization mechanism used by RFF prioritizes the search for plans that reach a goal state in the lowest number of state transitions possible. The results reported in Table 1 reflect that, on average, the plan length of the successful runs of the solutions computed by Prob-PRP are in the same order of magnitude than those of the solutions computed by RFF. In this section we evaluate the impact of the log-likelihood plan maximization strategy used by Prob-PRP towards the construction of policies with smaller expected plan length.

We compared the solutions obtained by Prob-PRP with a version that omits probabilities in the search of deterministic plans, and considers uniform action costs instead. We refer to this variation as Prob-PRP$_{uc}$. Note that this is not the same as assuming uniformly distributed outcome probabilities: a plan π has cost equal to its length regardless of the branching factor of the non-deterministic actions in π, whereas the branching factor influences the derived log-likelihood cost.

Table 2 shows the quality, run time, size, and expected plan length of the MAXPROB solutions to the different probabilistic planning problems obtained by Prob-PRP and Prob-PRP$_{uc}$. The overhead in Prob-PRP due to the probabilistic reasoning does not penalize the overall run time significantly, that remains in the same order of magnitude. Both algorithms find essentially the same solutions to the *triangle-tireworld* problems – where the optimal solutions w.r.t. success rate extend to stochastic shortest plan solutions quite straightforwardly. In the *blocksworld* domain, Prob-PRP obtains smaller policies with, in general, smaller expected plan length. In the *boxworld* domain, both algorithms find policies that are similar in size, but the expected plan length of the solutions found by Prob-PRP is considerably smaller in big problem instances. The *exploding-blocksworld* is a domain with many dead end states. In general, the size of the policies and the expected length of the plans found by each algorithm differ. In this domain, the most likely paths are not necessarily the more robust and the quality of the solutions do not seem to rely directly on a likelihood maximization criterion, nor on the election of a good heuristic to find deterministic plans. Rather, the quality of the solution appears to depend on a serendipitous election of the right sequence of actions during the search process.

The inconsistent quality of the plans obtained for the *exploding-blocksworld* domain suggests that the heuristic used by Prob-PRP in the search of deterministic weak plans is not informative in this domain. We obtained the best global results using a best-first search with the h_{FF} heuristic, although the combination of an $A^\star$ or breadth-first search with other heuristics is also competitive in certain problems. In particular, a best-first search with the additive heuristic h_{add}, which is usually informative, leads to similar results in most of the problems. Remarkably, in the *blocksworld* domain, the h_{add} heuristic performed better than the h_{FF} heuristic, generating smaller policies with smaller expected length. In the last three instances, this configuration finds solutions with 58 states and an expected plan length of 64 transitions, thus reducing the size and expected plan length to one half of the results reported in Table 2.

	Prob-PRP$_{uc}$				Prob-PRP			
problem	%	T	S	L	%	T	S	L
blocksworld-p01	**100**	0,02	21	24	**100**	**0,00**	**17**	**19**
blocksworld-p02	**100**	**0,00**	21	24	**100**	0,02	**17**	**19**
blocksworld-p03	**100**	**0,00**	21	24	**100**	0,02	**17**	**19**
blocksworld-p04	**100**	**0,00**	21	24	**100**	0,02	**17**	**19**
blocksworld-p05	**100**	**0,14**	**35**	**39**	**100**	0,18	43	47
blocksworld-p06	**100**	**0,14**	**35**	**39**	**100**	0,16	43	47
blocksworld-p07	**100**	**0,14**	**35**	**39**	**100**	0,16	43	47
blocksworld-p08	**100**	**0,14**	**35**	**39**	**100**	0,16	43	47
blocksworld-p09	**100**	**0,46**	71	75	**100**	0,48	**61**	**65**
blocksworld-p10	**100**	**0,44**	71	75	**100**	0,46	**61**	**65**
blocksworld-p11	**100**	**0,44**	71	75	**100**	0,46	**61**	**65**
blocksworld-p12	**100**	**0,46**	71	75	**100**	0,46	**61**	**65**
blocksworld-p13	**100**	**1,38**	110	119	**100**	1,40	**107**	**115**
blocksworld-p14	**100**	**1,38**	110	119	**100**	1,44	**107**	**115**
blocksworld-p15	**100**	**1,38**	110	119	**100**	1,40	**107**	**115**
boxworld-p01	**100**	0,14	**57**	156	**100**	**0,06**	**57**	**32**
boxworld-p02	**100**	0,16	**57**	156	**100**	**0,06**	**57**	**32**
boxworld-p03	**100**	0,14	**57**	156	**100**	**0,06**	**57**	**32**
boxworld-p04	**100**	0,36	**101**	86	**100**	**0,26**	105	**39**
boxworld-p05	**100**	0,34	**101**	86	**100**	0,28	105	**39**
boxworld-p06	**100**	6,56	269	363	**100**	**2,44**	**266**	**69**
boxworld-p07	**100**	6,60	269	363	**100**	**2,44**	**266**	**69**
boxworld-p08	**100**	**1,44**	**166**	170	**100**	1,92	207	**63**
boxworld-p09	**100**	**1,46**	**166**	170	**100**	1,92	207	**63**
boxworld-p10	**100**	**9,74**	**301**	328	**100**	18,2	415	**102**
boxworld-p11	**100**	**9,84**	**301**	328	**100**	18,2	415	**102**
boxworld-p12	**100**	**9,86**	**301**	328	**100**	18,0	415	**102**
boxworld-p13	**100**	576	949	1000+	**100**	**161**	**906**	**178**
boxworld-p14	**100**	507	949	1000+	**100**	**160**	**906**	**178**
boxworld-p15	**100**	586	949	1000+	**100**	**159**	**906**	**178**
ex-blocksworld-p01	**100**	**0,00**	**9**	**9**	**100**	**0,00**	**9**	**9**
ex-blocksworld-p02	**53,9**	0,04	**15**	**10**	**53,9**	**0,02**	**15**	**10**
ex-blocksworld-p03	59,3	0,16	**11**	9	**59,7**	**0,14**	12	**8**
ex-blocksworld-p04	60,1	**0,06**	**16**	**21**	**61**	**0,06**	18	**21**
ex-blocksworld-p05	**100**	**0,02**	**8**	23	**100**	0,02	11	**7**
ex-blocksworld-p06	96,3	0,68	**25**	**22**	**96,8**	**0,32**	28	**22**
ex-blocksworld-p07	**100**	1,76	**14**	31	**100**	**0,04**	21	**13**
ex-blocksworld-p08	**39,2**	1,00	**32**	23	36,6	**0,38**	**32**	**18**
ex-blocksworld-p09	**22,8**	**10,3**	**45**	31	10,2	58,7	77	**28**
ex-blocksworld-p10	**10,2**	**4,08**	**52**	28	4,6	14,0	105	**26**
ex-blocksworld-p11	9,6	33,8	89	29	**19,2**	**7,20**	**82**	**27**
ex-blocksworld-p12	–	–	m	–	**2,4**	**5,90**	**78**	**17**
schedule-p02	**100**	**0,04**	**7**	**48**	**100**	**0,04**	**7**	**48**
schedule-p03	**100**	**0,12**	**7**	**87**	**100**	**0,12**	**7**	**87**
schedule-p04	**100**	**0,08**	**16**	**43**	**100**	**0,14**	21	46
schedule-p05	**100**	**0,18**	**16**	96	**100**	0,20	16	**95**
triangle-tireworld-p01	**100**	**0,00**	**10**	**6**	**100**	**0,00**	**10**	**6**
triangle-tireworld-p02	**100**	**0,00**	**23**	**12**	**100**	**0,00**	**23**	**12**
triangle-tireworld-p03	**100**	**0,02**	**38**	**19**	**100**	**0,02**	**38**	**19**
triangle-tireworld-p04	**100**	**0,06**	**55**	**25**	**100**	**0,06**	**55**	**25**
triangle-tireworld-p05	**100**	**0,12**	**74**	**32**	**100**	**0,12**	**74**	**32**
triangle-tireworld-p06	**100**	**0,20**	**95**	**39**	**100**	**0,20**	**95**	**39**
triangle-tireworld-p07	**100**	**0,32**	**118**	**45**	**100**	0,36	**118**	**45**
triangle-tireworld-p08	**100**	**0,54**	**143**	**52**	**100**	0,62	**143**	**52**
triangle-tireworld-p09	**100**	**0,94**	**170**	**58**	**100**	1,14	**170**	**58**
triangle-tireworld-p10	**100**	**1,56**	**199**	**65**	**100**	1,84	**199**	**65**

Table 2: Solutions obtained by Prob-PRP using uniform action costs and log-prob action costs. % indicates the percentage of successful executions; T indicates run time, in seconds; S indicates the size of the policy; and L indicates the average length of the plans.

6 Extended Evaluation

In this section we introduce the benefits of a planner to be robust to small probability perturbations and different orderings used in the declaration of the actions.

6.1 Robustness to Probability Perturbations

The probabilistic planning model specifies the probability distribution of the outcomes of the actions, and is assumed to be known by the planner. In many real problems, however, these probabilities are unknown, or not known with complete accuracy. Ideally, the probability of success of a good solution is robust to small fluctuations in these probabilities. In practice, the search space explored by probabilistic algorithms exhibit *phase transitions* that change the structure of the solutions found. These phase transitions are most evident when the most probable outcome of the action changes.

In this section we evaluate the robustness of the solutions in the face of small deviations in the transition probabilities declared in the *triangle-tireworld* model. In this domain, the probability of a flat tire after moving the car is 0.5. We informed the planners with a slightly inaccurate probability, 0.45, breaking the uniform non-determinism of the action *move-car*. Strong solutions to the problem need to consider the faulty effect, that is no longer the most probable outcome. For that reason, we configured RFF to use the all-outcomes determinization.

	RFF		Prob-PRP	
problem	% sol	% sim	% sol	% sim
triangle-tireworld-p01	56.7	53.4	**100**	**100**
triangle-tireworld-p02	16.8	12.9	**100**	**100**
triangle-tireworld-p03	4.9	3.2	**100**	**100**
triangle-tireworld-p04	1.8	0.9	**100**	**100**
triangle-tireworld-p05	0.5	0.2	**100**	**100**
triangle-tireworld-p06	0.0	0.0	**100**	**100**
triangle-tireworld-p07	0.0	0.0	**100**	**100**
triangle-tireworld-p08	0.1	0.0	**100**	**100**
triangle-tireworld-p09	0.0	0.1	**100**	**100**
triangle-tireworld-p10	0.0	0.0	**100**	**100**

Table 3: Quality of the solutions in the *triangle-tireworld* domain when the probability of having a flat tire is perturbed from 0.5 to 0.45.

Table 3 shows the probability of success for solutions computed by RFF and Prob-PRP. The columns % *sol* and % *sim* in Table 3 indicate the probability of success when the environment model corresponds, respectively, to the model given to the planner, or to the original model.

The solutions found by RFF are not optimal anymore. Even when using the all-outcomes determinization, FF skews the search towards the shortest, but also optimistic plans that go through unsafe locations not equipped with a spare. Therefore, the envelope constructed by RFF does not converge to an optimal policy.

The success rate of the solutions found by RFF drops dramatically from the 100% achieved in the original domain description, and the performance in the simulation has an even lower percentage of success. On the other hand, since the *triangle-tireworld* domain has avoidable dead ends, Prob-PRP is guaranteed to find a strong cyclic independent of the transition probabilities of the model.

6.2 Robustness to Action Orderings

For evaluation purposes, we swapped the order in the declaration of the (equally probable) effects of the action *move-car*. In the new model, the *first* effect is optimistic and considers that the car's tire will not become flat, whereas the second effect is pessimistic and considers that the car's tire will become flat. With this ordering, the deterministic plans computed by FF in the most probable determinization of the problem are optimistic and not robust. As a consequence, RFF consistently fails to find robust policies, and the quality

of the solutions drops dramatically reaching a failure rate of 100% in the fourth instance. On the other hand, Prob-PRP is guaranteed to find optimal solutions to problems with avoidable dead ends regardless of the order used in the declaration of the actions and probability fluctuations.

7 Summary and Discussion

We introduced Prob-PRP, an algorithm that extends the state-of-the-art FOND planner PRP to compute solutions to MAXPROB problems. Prob-PRP is sound and complete for problems with avoidable dead ends. Probabilistic planning in the presence of avoidable and/or unavoidable dead ends is a challenging and important task (Kolobov, Mausam, and Weld 2012). We detailed a number of related approaches. Perhaps most similar to Prob-PRP's use of partial state policies and forbidden state-action pairs are the "basis functions" and "nogoods" computed by the GOTH planner and SixthSense algorithm (Kolobov and Weld 2010). Key distinctions include how Prob-PRP uses and updates the policy during the search for a plan, and how our dead ends are computed and used as forbidden state-action pairs.

MAXPROB solutions found by Prob-PRP are often optimal, and outperform the solutions found by RFF. We examined different properties that make for a good quality policy. Prob-PRP's solutions nicely balance their compactness and the expected length of the plans. Moreover, Prob-PRP demonstrates better scalability than RFF, and produces offline solutions. Computing offline solutions makes it possible to estimate the probability of success prior to execution, thus offering a better guarantee of the policy's quality than the solutions computed by online planners. Moreover, the guarantees on the optimality of the solutions in Prob-PRP makes it robust to small variations in the transition probabilities such as those found from an imprecise planning model.

We found that different search techniques for the deterministic subsolver – namely, a combination of breadth-first search, best-first search, $A^\star$, different heuristics, and uniform or probabilistic costs – offer similar results that are sometimes not of high quality. Whether the selection of an effective heuristic or search algorithm will significantly improve the results in these types of domains remains an open question. In future work we will explore these and other related issues associated with finding high-quality policies for such non-deterministic domains.

Establishing the correspondence of the computational core that is shared by FOND and probabilistic planning enables advances in either discipline to be exploited by the other. In this paper we demonstrated this by exploiting compact policy representations, relevance reasoning, and dead end avoidance developed within the FOND community and used these to advance the state of the art in probabilistic planning. Moving forward, we aim to inspire new methods for solving FOND problems using some of the insights from probabilistic planning, such as sample-based search.

Acknowledgements: The authors gratefully acknowledge funding from the Natural Sciences and Engineering Research Council of Canada (NSERC) and from Australian Research Council (ARC) discovery grant DP130102825.

References

Bertsekas, D. P., and Tsitsiklis, J. N. 1991. An Analysis of Stochastic Shortest Path Problems. *Mathematics of Operations Research* 16(3):580–595.

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence* 129(1-2):5–33.

Bonet, B., and Geffner, H. 2003. Labeled RTDP: Improving the Convergence of Real-Time Dynamic Programming. In *ICAPS*, 12–21.

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

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

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:263–312.

Jimenez, S.; Coles, A.; and Smith, A. 2006. Planning in probabilistic domains using a deterministic numeric planner. *PlanSIG*.

Keller, T., and Eyerich, P. 2012. PROST: Probabilistic Planning Based on UCT. *ICAPS*.

Kolobov, A., and Weld, D. S. 2010. SixthSense: Fast and reliable recognition of dead ends in MDPs. *Twenty-Fourth AAAI Conference on Artificial Intelligence*.

Kolobov, A.; Mausam; Weld, D. S.; and Geffner, H. 2011. Heuristic search for generalized stochastic shortest path MDPs. *ICAPS* 130–137.

Kolobov, A.; Mausam; and Weld, D. S. 2012. A theory of goal-oriented MDPs with dead ends. In *Proceedings of the 28th Conference on Uncertainty in Artificial Intelligence*, 438–447.

Little, I., and Thiébaux, S. 2007. Probabilistic planning vs. replanning. *ICAPS Workshop on IPC: Past, Present and Future*.

Mattmüller, R.; Ortlieb, M.; Helmert, M.; and Bercher, P. 2010. Pattern database heuristics for fully observable nondeterministic planning. In *ICAPS*, 105–112.

Muise, C.; McIlraith, S. A.; and Beck, J. C. 2012. Improved Non-deterministic Planning by Exploiting State Relevance. In *ICAPS*, 172–180.

Muise, C.; McIlraith, S. A.; and Belle, V. 2014. Non-deterministic planning with conditional effects. In *ICAPS*, 370–374.

Puterman, M. 1994. *Markov Decision Processes: Discrete Dynamic Programming*. New York: Wiley.

Teichteil-Königsbuch, F.; Kuter, U.; and Infantes, G. 2010. Incremental plan aggregation for generating policies in MDPs. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1*, number 1, 1231–1238.

Teichteil-Königsbuch, F.; Vidal, V.; and Infantes, G. 2011. Extending Classical Planning Heuristics to Probabilistic Planning with Dead-Ends. *AAAI* 1017–1022.

Teichteil-Königsbuch, F. 2012. Stochastic Safest and Shortest Path Problems. *AAAI* 1825–1831.

Trevizan, F., and Veloso, M. 2012. Short-Sighted Stochastic Shortest Path Problems. *ICAPS*.

Yoon, S.; Fern, A.; and Givan, R. 2007. FF-Replan: A baseline for probabilistic planning. In *ICAPS*, 352–359.

A Heuristic Estimator based on Cost Interaction

Yolanda E-Martín[1,2] and **María D. R-Moreno**[1] and **David E. Smith**[3]
[1] Departamento de Automática. Universidad de Alcalá. Ctra Madrid-Barcelona, Km. 33,6 28871 Alcalá de Henares (Madrid), Spain.
{yolanda,mdolores}@aut.uah.es
[2] Universities Space Research Association. 615 National Ave, Suite 220, Mountain View, CA 94043
[3] Intelligent Systems Division. NASA Ames Research Center. Moffett Field, CA 94035-1000
david.smith@nasa.gov

Abstract

In planning with action costs, heuristic estimators are commonly used to guide the search towards lower cost plans. Some admissible cost-based heuristics are weak (non-informative), while others are expensive to compute. In contrast, non-admissible cost-based heuristics are in general more informative, but may overestimate the computed cost, yielding non-optimal plans. This paper introduces a domain-independent non-admissible heuristic for planning with action costs that computes more accurate cost estimates. This heuristic is based on cost propagation in a plan graph, but uses *interaction* to compute information about the relationships between pairs of propositions and pairs of actions to calculate more accurate cost estimates. We show the trade-off between the solution quality and the planning performance when applying this heuristic in classical planning. This heuristic is expensive for planning, but we demonstrate that it is quite useful for goal recognition.

Introduction

In planning with action costs, heuristic estimators are commonly used to guide the search towards lower cost plans. Some admissible cost-based heuristics, like the max heuristic h^{max} (Bonet and Geffner, 2001), are weak (non-informative), while others, like the higher-order heuristics h^m (Haslum and Geffner, 2000), are expensive to compute. In contrast, non-admissible cost-based heuristics, like the additive heuristic h^{add} (Bonet and Geffner, 2001) and the set-additive heuristic h^s_a (Keyder and Geffner, 2007), are more informative and have shown good performance in time and solution quality. However, they may overestimate the computed cost in some cases, yielding non-optimal plans.

In this paper, we introduce a domain-independent non-admissible heuristic for planning with action costs that computes more accurate cost estimates. This heuristic is based on cost propagation in a plan graph, but we use *interaction* (Bryce and Smith, 2006) to compute information about the relationships between pairs of propositions and pairs of actions, yielding more accurate cost estimates. We show the trade-off between the solution quality and the planning performance when applying this heuristic in classical planning. This heuristic is expensive for planning, but we demonstrate it is quite worthwhile for goal recognition.

This paper begins with a review of cost propagation in a plan graph. Then, we introduce *interaction* and its use in cost propagation in a plan graph. Then we describe two cost-based heuristic estimators and evaluate their accuracy. We present an empirical study of the heuristics in classical planning and goal recognition.

Cost propagation in a plan graph

Plan graphs (Blum and Furst, 1997) provide an optimistic method to estimate the set of achievable propositions, and the set of feasible actions, for a particular starting state and goal state. They have been commonly used to compute heuristic distance estimates between states and goals and, recently, to compute estimates of the cost that propositions can be achieved, and an action can be performed. Propagation of cost estimates in a plan graph is a technique that has been used in a number of planning systems (Bonet, Loerincs, and Geffner, 1997; Nguyen, Kambhampati, and Nigenda, 2002; Do and Kambhampati, 2002). The computation of cost estimates starts from the initial conditions and works progressively forward through each successive layer of the plan graph. For level 0 it is assumed that the cost of the propositions is zero. With this assumption, the propagation starts by computing the cost of the actions at level zero. In general, the cost of performing an action a at level l with a set of preconditions $\mathcal{P}_a$ is equal to the cost of achieving its preconditions. This may be computed in two different ways: (1) Maximization: the cost of an action is equal to the cost of reaching its costliest precondition, i.e., $cost(a) = \max_{x_i \in \mathcal{P}_a} cost(x_i)$, (2) Summation: the cost of an action is equal to the cost of reaching all its preconditions, i.e., $cost(a) = \sum_{x_i \in \mathcal{P}_a} cost(x_i)$. The first method is admissible since it underestimates the cost of an action. This allows for dependence among the preconditions of an action. In contrast, the second method is inadmissible since it assumes independence among all preconditions of an action. As a consequence, the cost may be underestimated, if some of the preconditions interfere with each other, or overestimated, if some of the preconditions are achieved by a common action.

The cost of achieving a proposition x at level l, achieved by actions $\mathcal{A}_x$ at the preceding level, is the minimum cost among all $a \in \mathcal{A}_x$. It is defined as $cost(x) = \min_{a \in \mathcal{A}_x} [cost(a) + Cost_a]$, where $Cost_a$ is the cost of ap-

plying the action a.

Taking the above calculations into consideration, a cost-plan graph is built in the same way that an ordinary planning graph is created. The construction process finishes when two consecutive proposition layers are identical and there is quiescence in cost for every proposition and action in the plan graph. On completion, each possible goal proposition has an estimated cost.

Our work focuses on computing more accurate estimates of cost by introducing the concept of interaction in the cost propagation in a plan graph. This primarily impacts the calculation of cost for sets of preconditions, although the consequences propagate through to propositions. We describe the calculation and use of interaction in the next section.

Interaction

Interaction is a value that represents how more or less costly it is that two propositions or actions are established together instead of independently. This concept is a generalization of the mutual exclusion concept used in classical planning graphs. Formally, the optimal Interaction, I^*, considers n-ary interaction relationships among propositions and among actions (p_0 to p_n) in the plan graph, and it is defined as:

$$I^*(p_0, ..., p_n) = cost^*(p_0 \wedge ... \wedge p_n) - (cost^*(p_0) + ... + cost^*(p_n)) \quad (1)$$

where the term $cost^*(p_0 \wedge ... \wedge p_n)$ is the minimum cost among all the possible plans that achieve all the members in the set. Computing I^* would be computationally prohibitive. As a result, we limit the calculation of these values to pairs of propositions and pairs of actions in each level of a plan graph – in other words, binary interaction:

$$I^*(p, q) = cost^*(p \wedge q) - (cost^*(p) + cost^*(q)) \quad (2)$$

It has the following features:

$$I^*(p,q) \ \ is \ \begin{cases} < 0 & \text{if } p \text{ and } q \text{ are } \textit{synergistic} \\ = 0 & \text{if } p \text{ and } q \text{ are } \textit{independent} \\ > 0 & \text{if } p \text{ and } q \textit{ interfere} \end{cases}$$

In other words, I provides information about the degree of interference or synergy between pairs of propositions and pairs of actions in a plan graph. When $0 < I(p, q) < \infty$ it means that there is some interference between the best plans for achieving p and q so it is harder (more costly) to achieve them both than to achieve them independently. In the extreme case, $I = \infty$, the propositions or actions are mutually exclusive. Similarly, $I(p, q) < 0$ (synergy) means that the cost of establishing both p and q is less than the sum of the costs for establishing the two independently. However, this cost cannot be less than the cost of establishing the most difficult of p and q. As a result $I(p, q)$ is bounded below by $-\min[cost(p), cost(q)]$.

Instead of computing mutex information, the new cost propagation technique computes interaction information between all pairs of propositions and all pairs of actions at each level. Interaction is taken into account in the cost propagation, which establishes a better estimation of the cost for two propositions or two actions performed at the same time.

The computation of cost and interaction information begins at level zero of the plan graph and sequentially proceeds to higher levels. For level zero it is assumed that (1) the cost of each proposition at this level is 0, and (2) the interaction between each pair of propositions at this level is 0, which means independence between propositions. Neither of these assumptions are essential, but they are adopted here for simplicity.

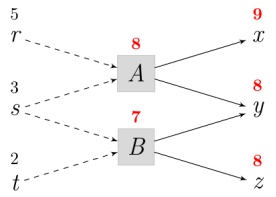


Figure 1: A cost plan graph level with cost and interaction calculation and propagation.

Computing action cost and interaction

The cost and interaction information of a proposition layer at a given level of the plan graph is used to compute the cost and the interaction information for the subsequent action layer. Considering an action a at level l with a set of preconditions $\mathcal{P}_a$, the cost of an action is approximated as the cost that all the preconditions are achieved plus the interaction between all pairs of preconditions:

$$cost^*(a) = cost^*(\mathcal{P}_a) \approx \sum_{x_i \in \mathcal{P}_a} cost(x_i) + \sum_{\substack{(x_i, x_j) \in \mathcal{P}_a \\ j > i}} I(x_i, x_j) \quad (3)$$

As an example, consider one level of the cost-plan graph shown in Figure 1. There are three propositions r, s, and t with costs 5, 3, and 2 respectively, and interaction values $I(r, s) = 0$, $I(r, t) = -1$, and $I(s, t) = 2$. There are two actions A and B, which have cost 1. The number above the propositions and actions are the costs computed for each one during the cost propagation process (those highlighted are the cost that we will compute in this section). For this example, the costs of actions A and B are:

$$\begin{aligned} cost(A) &\approx cost(r) + cost(s) + I(r, s) = 5 + 3 = 8 \\ cost(B) &\approx cost(s) + cost(t) + I(s, t) = 3 + 2 + 2 = 7 \end{aligned}$$

The next step is to compute the interaction between actions, which is defined as follows:

$$I^*(a, b) = \begin{cases} \infty & \text{if } a \text{ and } b \text{ are mutex by inconsistent effects or interference} \\ cost^*(a \wedge b) - cost^*(a) - cost^*(b) & \text{otherwise} \end{cases}$$

where $cost^*(a \wedge b)$ is defined to be $cost(\mathcal{P}_a \cup \mathcal{P}_b)$. This is approximated as in (3) by:

$$cost(\mathcal{P}_a \cup \mathcal{P}_b) \approx \sum_{x_i \in \mathcal{P}_a \cup \mathcal{P}_b} cost(x_i) + \sum_{\substack{(x_i, x_j) \in \mathcal{P}_a \cup \mathcal{P}_b \\ j > i}} I(x_i, x_j)$$

If the actions are mutex by inconsistent effects, or interference, then the interaction is ∞. Otherwise, the interaction

above simplifies to:

$$I(a,b) \approx \sum_{\substack{x_i \in \mathcal{P}_a - \mathcal{P}_b \\ x_j \in \mathcal{P}_b - \mathcal{P}_a}} I(x_i,x_j) - \Big[\sum_{x_i \in \mathcal{P}_a \cap \mathcal{P}_b} cost(x_i) + \sum_{\substack{(x_i,x_j) \in \mathcal{P}_a \cap \mathcal{P}_b \\ j>i}} I(x_i,x_j) \Big]$$

For the example in Figure 1, the interaction between actions A and B reduces to:

$$I(A,B) \approx I(r,t) - cost(s) = -1 - 3 = -4$$

The fact that $I(A,B) = -4$ means that there is some degree of synergy between both actions. This synergy comes from the fact that they have a common precondition s.

Computing proposition cost and interaction

The next step consists of calculating the cost of the propositions at the next level. In this calculation, all the possible actions at the previous level that achieve each proposition are considered. We make the usual optimistic approximation that the least expensive action can be used. Therefore, the cost of a proposition is the minimum over the costs of all the actions that can achieve the proposition. More formally, for a proposition x at level l, achieved by actions $\mathcal{A}_x$ at the preceding level, the cost is calculated as:

$$cost^*(x) = \min_{a \in \mathcal{A}_x} [cost(a) + Cost_a] \quad (4)$$

In our example, the cost of proposition y of the graph is:

$$\begin{aligned} cost(y) &= \min[cost(A) + Cost_A, cost(B) + Cost_B] \\ &= \min[8+1, 7+1] = 8 \end{aligned}$$

Finally, the interaction between a pair of propositions x and y is computed. In order to compute the interaction between two propositions at a level l, all possible ways of achieving those propositions at the previous level are considered. In other words, all the actions that achieve the pair of propositions are considered. Suppose that $\mathcal{A}_x$ and $\mathcal{A}_y$ are the sets of actions that achieve propositions x and y respectively at level l. The interaction between x and y is then:

$$\begin{aligned} I^*(x,y) &= cost^*(x \wedge y) - cost^*(x) - cost^*(y) \\ &= \min_{\substack{a \in \mathcal{A}_x \\ b \in \mathcal{A}_y}} \Big\{ cost(a \wedge b) \Big\} - cost^*(x) - cost^*(y) \\ &\approx \min \left\{ \begin{array}{l} \min\limits_{a \in \mathcal{A}_x \cap \mathcal{A}_y} cost(a) + Cost_a \\ \\ \min\limits_{\substack{a \in \mathcal{A}_x - \mathcal{A}_y \\ b \in \mathcal{A}_y - \mathcal{A}_x}} \left[\begin{array}{l} cost(a) + Cost_a + \\ cost(b) + Cost_b + \\ I(a,b) \end{array} \right] \end{array} \right\} \\ & \quad - cost(x) - cost(y) \end{aligned}$$

In our simple example, consider the calculation of the interaction between x and y where the possible ways to achieve both are performing A or A and B. Only the first of these possibilities is considered, since In this case, $\mathcal{A}_x - \mathcal{A}_y$ is empty. The interaction is therefore simply:

$$\begin{aligned} I(x,y) &\approx [cost(A) + Cost_A] - cost(x) - cost(y) \\ &= [8+1] - 9 - 8 = -8 \end{aligned}$$

Heuristic estimator based on interactions

The cost-plan graph described in the previous section includes, for each proposition and action in the plan graph, an approximate cost of achievement. These cost estimates may be used as heuristic estimators. The next subsections describe two different methods that make use of this information to compute heuristic estimates. The first method computes the estimated cost using the information given in the plan graph, while the second one builds a relaxed plan where the propagated cost information is taken into account in the relaxed-plan construction.

The h^I heuristic

The h^I heuristic is based directly on the cost and interaction information computed in the cost-plan graph. It defines the estimated cost of achieving a (conjunctives) goal $G = \{g_1, ..., g_n\}$ as:

$$h^I = cost(G) \approx \sum_{g_i \in G} \left[cost(g_i) + \sum_{\substack{(g_i,g_j) \in G \\ j<i}} I(g_i,g_j) \right] \quad (5)$$

The interaction information helps to compute more accurate estimates of cost when subgoals interfere with each other. However, the fact that the interaction computation is binary makes the heuristic h^I non-admissible. Therefore, the estimated cost is an approximation of the optimal cost. Essentially, when all the preconditions of each action are independent of each other, the heuristic h^I reduces to the heuristic h^{add}. Otherwise, h^I will be greater or less than h^{add} depending on whether the interaction is negative or positive.

The h^I_{rp} heuristic

The h^I_{rp} heuristic is based on computing a relaxed plan with the use of cost information in the plan graph. The more sophisticated strategy is to make use of cost and interaction information in the plan graph when selecting actions for the relaxed plan. In particular, to achieve a particular subgoal at a level l, the relaxed plan construction chooses the action that minimizes the cost of achieving the goal at level l. The sum of the costs of the actions in the relaxed plan π provides an estimate of cost for the goal. It is defined as:

$$h^I_{rp} = cost(\pi) = \sum_{a_i \in \pi} Cost_{a_i} \quad (6)$$

Like the heuristic h^I, the heuristic h^I_{rp} is non-admissible.

Accuracy Evaluation

The previous section describes two inadmissible heuristics based on a cost-plan graph with interactions. To evaluate the accuracy of these heuristics, we compare *estimated cost* against the *optimal cost* for a suite of problems. The optimal cost of each problem is computed using the optimal planner HSP^*_f (Haslum, 2008), which was allowed to run for an unlimited amount of time. In addition to the h^I and h^I_{rp} heuristics, the evaluation is done for h^{add} and h^{add}_{rp}, which are the versions without interaction. In these cases, the cost-plan graph is built using traditional cost propagation. Likewise, the evaluation is done for the set-additive heuristic h^s_a. In addition, we did the evaluation using the satisficing planners LPG (Gerevini, Saetti, and Serina, 2003), SGPlan_6 (Chen, W., and Chih-Wei, 2006), and MetricFF (Hoffmann, 2003). LPG is run using LPG-speed (LPG_s) that computes the first solution, and LPG-quality (LPG_q) that computes the best solution. MetricFF is run under its three different approaches that perform cost minimization using weighted A^* (MetricFF_3), A^*_ε (MetricFF_4), and enforced hill-climbing then A^*_ε (MetricFF_5). The experiments were conducted on an Intel Xeon CPU E5-1650 processor running at 3.20GHz with 32 GB of RAM in a time limit of 1800s.

Table 1 shows the results of this evaluation on 8 planning domains with 15 problems each. For each approach in each domain, each column shows the following measures:

- *r* is the ratio of the *estimated cost* to the *optimal cost* per problem.
- *M* is the mean of the ratio among the *solved* problems.
- σ is the standard deviation of the ratio among the *solved* problems.

The symbol "-" means the approach does not solve the problem within the time limit. Bold values symbolized the closer and lower variance cost estimate. In the Blocksword and Elevator domains, h^I computes cost estimates that are considerably closer to the optimal and more consistent (lower variance) than the actual costs computed by the satisficing planners LPG_s, LPG_q, SGPlan_6, and MetricFF. In the Logistics domain, h^I computes cost estimates as accurate as MetricFF_3, which computes the closest and most consistent solution among the evaluated planners. In the rest of the domains, h^I generates better estimates than h^{add} and h^{add}_{rp}, but not as good as h^s_a and the better satisficing planners. For the h^I_{rp} heuristic, in the Intrusion and Kitchen domains, it computes cost estimates equal to the optimal cost. In the Blocks, Campus, Floortile, and Logistics domains, h^I_{rp} computes cost estimates that are better than the costs computed by some of the satisficing planners. In the Elevator domain, it computes poor cost estimates, but in Pegsol it does slightly better than h^I.

Overall, h^I computes cost estimates with lower variance and closer to the optimal cost than cost estimates computed by h^{add} in all the domains except Logistics. h^{add}_{rp} and h^I_{rp} have similar behaviors. Our hypothesis is that while constructing the cost-relaxed plan, the algorithm only considers the actions that minimize the cost, but not the interactions between/among them. Those selected actions might be the same as the ones where the interaction during cost propagation is not considered. As a result, using interaction information in relaxed plan extraction might give a more accurate estimate of cost.

h^I Heuristics in Planning

As a result of the increased accuracy and stability of the above-described h^I heuristics, it is natural to try to use them for planning purposes. The MetricFF planner (Hoffmann, 2003) was modified to incorporate these heuristics. The search strategy remains the same. For purposes of this test, the A^*_ε strategy was chosen. The only difference is the successors evaluation of the current state, which is based on the cost estimate computed by the h^I or h^I_{rp} heuristic. The best successor is the one with the lowest cost. Four variations of the MetricFF planner are compared:

- MetricFF^I: h^I as heuristic function (equation (5)), and cost propagation through the plan graph considering interaction information.
- MetricFF^{add}: h^{add} as heuristic function, and cost propagation through the plan graph not considering interaction information.
- MetricFF^I_{rp}: h^I_{rp} as heuristic function (equation (6)), and cost propagation through the plan graph considering interaction information.
- $\text{MetricFF}^{add}_{rp}$: h^I_{rp} as heuristic function, and cost propagation through the plan graph not considering interaction information.

We have tested an additional strategy, namely MetricFF^I_k, which uses the h^I heuristic function and cost propagation through the plan graph considering interaction information in only the first k levels of the search process, and then h^{add} heuristic for the rest of the search. Because the h^I heuristic is expensive to compute, and heuristics tend to be less accurate earlier in the search, we wanted to find out if the h^I heuristic could benefit the search process, but limit computation by only using it for a limited number of levels of the search.

Tables 2 and 3 show the results of an accuracy evaluation on the same 8 planning domains used in the previous section. For each approach, each row shows an average of M and σ among all the domains. The MetricFF^I_k planner has been tested with $k = 1$ up to 4. In general, the 8 planner variations reach solutions very close to the optimal. As we expected, MetricFF^I and MetricFF^I_{rp} solution qualities are slightly better than the ones generated by MetricFF^{add} and $\text{MetricFF}^{add}_{rp}$. For MetricFF^I_k, when $k = 2$ or $k = 4$ the technique generates closer and more consistent cost estimates than for any other k value tested and any MetricFF variation. However, for $k = 2$ the technique is faster than for $k = 4$. For computational time, Figure 2 shows a scatter plot, where each dot in the plot represents the relationship between $\text{MetricFF}^{add}_{rp}$ and MetricFF^I, and between $\text{MetricFF}^{add}_{rp}$ and MetricFF^I_2 for each tested problem. (We chose to compare these three planner since they show better performance in terms of accuracy.) In general, MetricFF^I and MetricFF^I_2 are

Table 1: Accuracy evaluation among different planning and heuristic techniques.

Domain	Approach	r															M	σ
		p01	p02	p03	p04	p05	p06	p07	p08	p09	p10	p11	p12	p13	p14	p15		
Blocks	LPG_s	1.33	1.18	1	1.21	1.08	1	1	1.1	1.15	1	1	1	1.16	1.34	1.53	1.14	0.155
	LPG_q	1	1	1	1.21	1.25	1	1	1	1	1	1	1	1	1.13	1	1.039	0.081
	SGPlan	1	1	1	1	1	1	1	1	1	1	1	1	2.33	1	2.68	1.201	0.517
	MetricFF_3	1	1	1	1	1.25	1	1	1	1	1	1	1	1	1.26	1.18	1.046	0.094
	MetricFF_4	1.08	1	1	1.21	1.37	1	1	1	1.15	1.8	1	1	1.16	1.39	3.5	1.311	0.623
	MetricFF_5	1.41	1.75	1.35	1.21	1.45	1.8	1.75	1.1	1.15	1.8	1.37	1.37	1.16	1.47	1.34	1.435	0.231
	h^s_a	0.91	1	1	0.89	0.83	1	1	0.95	0.7	1	1	1	0.66	0.82	0.87	0.91	0.108
	h^I	1	1.12	1	1.1	1.2	1	1	0.75	1	1	1	1	1	1.04	1.15	**1.025**	**0.099**
	h^{add}	1.16	1.37	1.35	1.21	1.2	1	1	1.7	0.7	1	1.37	1.37	0.66	1.08	1.15	1.158	0.259
	h^I_{rp}	0.91	1	1	0.89	0.83	1	1	0.95	0.7	1	1	1	0.66	0.82	0.87	0.91	0.108
	h^{add}_{rp}	0.91	1	1	0.89	0.83	1	1	0.95	0.7	1	1	1	0.66	0.82	0.87	0.91	0.108
Campus	LPG_s	1.35	1.1	1.43	1.12	1	1.35	1.18	1.07	1.28	1.25	1.07	1.03	1.21	1.14	1.25	1.193	0.124
	LPG_q	1	1.07	1.12	1	1.12	1.07	1.12	1.07	1.28	1	1.18	1.03	1	1.03	1.31	1.096	0.096
	SGPlan	1	1	1	1	1	1.14	1	1	1	1	1	1	1	1	1	1.009	0.035
	MetricFF_3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	**1**	**0**
	MetricFF_4	1.85	1.25	1.37	1.37	1.37	1.71	1.56	1.33	1.85	1.43	1.33	1.07	1.14	1.14	1.37	1.413	0.233
	MetricFF_5	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	**1**	**0**
	h^s_a	0.85	0.92	0.87	0.81	0.87	0.85	0.75	1	0.85	0.87	1	0.96	0.92	0.92	0.81	0.888	0.068
	h^I	1	0.89	1	1	1	1	1.12	0.92	1	1	0.92	0.89	0.96	0.96	1	0.979	0.055
	h^{add}	2.5	2.67	2.93	2.81	2.68	2.64	2.68	2.81	2.5	2.93	2.81	2.71	2.53	2.53	2.81	2.707	0.141
	h^I_{rp}	1	0.92	1	1	0.93	1	0.87	1	1	1	1	1.1	0.92	0.92	1	0.98	0.051
	h^{add}_{rp}	0.85	0.92	0.87	0.81	1.06	0.85	0.75	1	0.85	0.87	1	1.07	0.92	0.92	0.81	0.907	0.09
Elevator	LPG_s	1.6	1.4	2.11	1.28	1.09	1.33	-	1.53	1	1.26	2.33	2.638	1.94	2.52	2.88	1.781	0.595
	LPG_q	1.9	1.8	2.11	1.28	1	1.08	-	1.06	1	1.36	1.8	1.42	1.94	2.19	2.76	1.624	0.514
	SGPlan	1.2	1	1.33	1.28	1.09	1.08	-	1.06	1.2	1.21	1.46	1.47	1.15	2.04	2.05	1.333	0.323
	MetricFF_3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	MetricFF_4	3.2	3.6	2	2.71	1.9	3.75	-	4.2	1.1	4.21	-	-	-	-	-	2.964	1.04
	MetricFF_5	2	1.8	1.66	1.28	1.27	1.08	-	1.73	1.2	1.31	2.33	1.94	1.73	2.52	1.7	1.686	0.411
	h^s_a	0.8	0.8	1.11	0.85	0.9	0.83	-	1	1	0.78	1.06	0.84	0.94	0.95	1	0.922	0.1
	h^I	0.7	1.8	1	1.85	0.72	0.75	-	1	1.3	0.73	0.66	0.57	0.94	0.85	0.94	**0.990**	**0.384**
	h^{add}	2.2	2.8	2.55	3.14	2.45	2.16	0	2.13	2	2.26	5.2	3.52	4	4.71	4.35	3.107	1.031
	h^I_{rp}	1.7	2.2	1.77	1.71	1.72	1.5	-	1.46	1.6	1.42	1.73	1.68	1.68	1.57	1.76	1.681	0.18
	h^{add}_{rp}	1.7	2.4	1.77	1.14	1.72	1.5	0	1.46	1.6	1.42	1.73	1.68	1.68	1.57	1.7	1.651	0.263
Floortile	LPG_s	1	1.36	1.32	1.53	1.76	4.94	2	4.55	5.46	7.65	7.79	5.2	4.74	4.32	4.16	3.855	2.177
	LPG_q	1	1	2.84	1	1.58	1.5	1.16	1.4	1.55	1.44	1.18	1.64	-	1.42	1.13	1.421	0.45
	SGPlan	1	1.36	1.8	2.06	1.52	1.944	1.54	1.5	1.72	1.65	1.37	1.49	1.43	1.78	1.47	1.58	0.252
	MetricFF_3	1	1	1	1.4	1	1	1.29	-	1.09	-	-	-	-	-	-	1.097	0.148
	MetricFF_4	1	1	3.16	1.93	3.08	-	-	-	-	-	-	-	-	-	-	2.036	0.951
	MetricFF_5	1	1.72	2.4	1.53	1.64	-	-	-	-	-	-	-	-	-	-	1.661	0.448
	h^s_a	1.2	1.09	0.92	1	0.88	0.86	0.75	1.01	0.8	1.03	0.86	0.83	1.15	0.91	0.94	**0.951**	**0.125**
	h^I	1	1	1.12	0.86	1	1	0.83	0.33	0.66	0.16	0.29	0.09	0.12	0.48	0.25	0.614	0.366
	h^{add}	1.2	1.27	1.04	1.2	1.02	1.02	1	1.28	1.04	1.66	1.18	1.41	1.85	1.19	1.27	1.246	0.233
	h^I_{rp}	1	1.36	1.16	1.13	1.05	0.88	0.83	1.03	0.86	0.86	0.76	0.75	0.81	0.92	0.88	**0.954**	**0.164**
	h^{add}_{rp}	1	1.36	1.16	1.13	1.05	0.88	0.83	1.03	0.86	0.83	0.76	0.75	0.81	0.92	0.88	**0.954**	**0.164**
Intrusion	others	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	**1**	**0**
	h^I	1	1	1	1	1	1	1	1	0.9	1	1	1	1	1	1	0.993	0.024
	h^{add}	1	1.33	1.25	1.33	1.25	1.22	1.33	1.25	1.5	1.28	1.25	1.25	1.25	1.25	1.28	1.269	0.097
Kitchen	others	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	**1**	**0**
	LPG_s	1	1	1	1	1	1	1	1	1.3	1.3	1	1	1	1	1.3	1.06	0.12
	LPG_q	1.06	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1.004	0.016
	SGPlan	1.06	1.06	1.06	1.06	1.13	1.13	1.13	1.13	1.3	1.3	1.3	1.3	1.3	1.3	1.3	1.192	0.103
	MetricFF_5	1.04	1.04	1.04	1.04	1	1	1	1	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.104	0.09
	h^I	0.97	0.97	0.97	0.97	1	1	1	1	1	1	1	1	1	1	1	0.994	0.009
	h^I_{rp}	1.06	1.06	1.06	1.06	1	1	1	1	1	1	1	1	1	1	1	1.017	0.028
Logistic	LPG_s	1.14	1.25	1.41	1.58	1.28	1.36	1.08	1.2	1	1.16	2.25	1.77	1	1.4	1.07	1.333	0.321
	LPG_q	1.21	1.08	1.08	1	1	1.27	1.41	1	1.08	1.16	1.5	1.22	1	1	1.3	1.156	0.156
	SGPlan	1	1	1.16	1.16	1	1.18	1.16	1.2	1	1	1	1.22	1.16	1	1	1.084	0.091
	MetricFF_3	1.07	1	1	1.16	1.07	1	1	1	1	1	1	1	1	1	1	**1.02**	**0.045**
	MetricFF_4	1.35	1.08	2.91	1.83	1.35	1.63	1.08	1	1.75	1.08	1.08	1	1.08	1.3	2.23	1.453	0.525
	MetricFF_5	1.14	1.08	1.08	1.08	1.07	1.18	1	1	1.08	1.08	1.08	1.44	1.08	1.6	1.23	1.15	0.158
	h^s_a	0.71	0.75	1	1	0.71	0.81	1	1	1	1	0.75	1	1	0.7	0.84	0.886	0.126
	h^I	0.92	1.08	1	1.08	0.92	0.72	1	1	1	1	1.08	0.88	1	1.2	0.76	**0.979**	**0.116**
	h^{add}	0.71	1.16	1	1	0.71	0.81	1	1	1	1	1	1	1	1	0.84	0.95	0.118
	h^I_{rp}	1	1.16	1	1.16	1	1	1	1	1	1	1.16	1	1	1.2	0.84	1.036	0.092
	h^{add}_{rp}	0.71	0.75	1	1	0.71	0.81	1	1	1	1	0.75	1	1	0.7	0.84	0.886	0.126
Pegsol	LPG_s	1	1	1	1.5	1.25	2	1.66	-	1.26	2.05	-	1.29	-	-	1.79	1.438	0.374
	LPG_q	1	1	1	1	1	1.25	1	1.05	1.26	1.55	-	1.04	-	-	-	**1.106**	**0.171**
	SGPlan	1	1	1	1	1.25	1.5	1.33	-	1.2	-	1.14	-	-	-	1.5	1.192	0.19
	MetricFF_3	1	1	1	1	1.25	1.75	1	1.83	1.4	1.83	1.28	1	1.33	1.57	1.5	1.317	0.309
	MetricFF_4	1	1	1	1.5	1.25	1.75	1.33	1.83	1.6	1.66	1.28	1.25	1.33	1.42	1.37	1.373	0.254
	MetricFF_5	1	1	1	1.5	1.25	1.75	2	1.83	1.6	1.66	1.28	1.25	1.33	1.42	1.37	1.418	0.297
	h^s_a	1	0.8	1.25	1.25	1	2	1.33	1.5	1.8	1.16	1.28	1.12	1	1.28	1.25	1.269	0.299
	h^I	1.5	0.53	0.33	4.41	0.33	0.41	6.44	1.66	0.46	0.22	0.76	0.29	2.81	1.09	0.29	1.439	1.747
	h^{add}	7.5	1.2	1.75	2.25	2.75	4	7.33	1.83	3	2.5	2	2.37	1.66	2.14	2.25	2.97	1.853
	h^I_{rp}	2	0.86	1.33	1.25	1.25	1.25	2.33	1.16	1.86	1.22	1.33	0.87	0.88	1.33	1.45	1.361	0.401
	h^{add}_{rp}	2.5	1	1.25	1.25	1.75	1.75	3	1.16	2	1.33	1.14	0.87	0.88	1.42	1.37	1.514	0.58

slower than $\text{MetricFF}^{add}_{rp}$. However, there are several problems where $\text{MetricFF}^{add}_{rp}$ does not find the solution within the given time, unlike MetricFF^I and MetricFF^I_2. It is also noticeable that the use of the h^I heuristic in only the first two

Table 2: Accuracy evaluation among different MetricFF variations based on h^I, h^{add}, h^I_{rp}, and h^{add}_{rp} heuristics.

Approach	MetricFF^I	MetricFF^{add}	MetricFF$^I_{rp}$	MetricFF$^{add}_{rp}$
M	1.006	1.013	1.088	1.015
σ	0.019	0.034	0.145	0.02

levels of the search process benefits the performance. It generates equal quality results and is a bit faster than MetricFF^I.

Table 3: Accuracy evaluation among different k values.

Approach	MetricFF$^I_1$	MetricFF$^I_2$	MetricFF$^I_3$	MetricFF$^I_4$
M	1.015	1.005	1.006	1.005
σ	0.026	0.017	0.019	0.016

In general, HSP$^*_f$ takes less time to solve a problem than any MetricFF variation we evaluated. One of the reasons might be the difference in the search algorithm, IDA* versus A$^*_\epsilon$. We have not done comparisons between heuristics in HSP$^*_f$ because the code for HSP$^*_f$ is not well documented and we have not been successful at incorporating different heuristics.

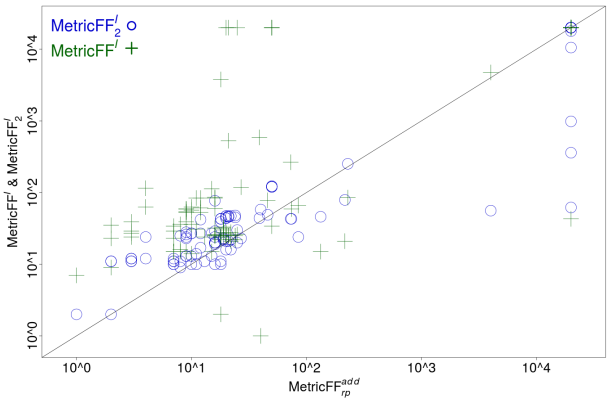


Figure 2: Time comparison among MetricFF$^{add}_{rp}$, MetricFF^I, and MetricFF$^I_2$.

While the computational overhead of full h^I is high and does not usually pay off during the actual search process, this heuristic is extremely useful for goal recognition, which we discuss in the next section.

h^I Heuristics in Goal Recognition

Ramírez (2010; 2012), defines a goal recognition problem to be a tuple $T = \langle P, \mathcal{G}, O, Pr \rangle$ where P is a planning domain and initial conditions, $\mathcal{G}$ is a set of possible goals or hypotheses, O is the observed action sequence $O = o_1, ..., o_n$, and Pr is the prior probability distribution over the goals in $\mathcal{G}$. The solution to a plan recognition problem is a probability distribution over the set of goals $G \in \mathcal{G}$ giving the relative likelihood of each goal. These posterior goal probabilities $P(G|O)$ can be characterized using Bayes Rule as:

$$Pr(G|O) = \alpha \, Pr(O|G) \, Pr(G) \tag{7}$$

where α is a normalizing constant, $Pr(G)$ is the prior distribution over $G \in \mathcal{G}$, and $Pr(O|G)$ is the likelihood of observing O when the goal is G. Ramírez goes on to characterize the likelihood $Pr(O|G)$ in terms of cost differences for achieving G under two conditions: complying with the observations O, and not complying with the observations O. More precisely, Ramírez characterizes the likelihood, $Pr(O|G)$, in terms of a Boltzman distribution:

$$Pr(O|G) = \frac{e^{[-\beta \, \Delta(G,O)]}}{1 + e^{[-\beta \, \Delta(G,O)]}} \tag{8}$$

where β is a positive constant and $\Delta(G, O)$ is the cost difference between achieving the goal with and without the observations:

$$\Delta(G, O) = Cost(G|O) - Cost(G|\overline{O}) \tag{9}$$

Putting equations (7) and (8) together yields:

$$Pr(G|O) = \alpha \frac{e^{[-\beta \, \Delta(G,O)]}}{1 + e^{[-\beta \, \Delta(G,O)]}} \, Pr(G) \tag{10}$$

By computing $\Delta(G, O)$ for each possible goal, equation 10 can be used to compute a probability distribution over those goals. The two costs necessary to compute Δ can be found by optimally solving the two planning problems $G|O$ and $G|\overline{O}$. Ramírez shows how the constraints O and $\overline{O}$ can be compiled into the goals, conditions and effects of the planning problem so that a standard planner can be used to find plans for $G|O$ and $G|\overline{O}$.

A significant drawback to the Ramírez approach is the computational cost of calling a planner twice for each possible goal. This makes the approach impractical for real-time goal recognition, such as for a robot observing a human, and trying to assist or avoid conflicts. Using h^I we show how to quickly infer a probability distribution over the possible goals using the framework of Ramirez. To compute $Cost(G|O)$, the cost-plan graph is pruned considering the sequence of observed actions. Consequently, cost estimates for goals with and without the observations are quickly computed, and a probability distribution over those goals is inferred.

Incremental Plan Recognition

Jigui and Minghao (2007) developed a framework for plan recognition that narrows the set of possible goals by incrementally pruning a plan graph as actions are observed. The approach consists of building a plan graph to determine which actions and which propositions are true (1), false (-1), or unknown (0) given the observations. For level zero, it is assumed the initial state is true: each proposition has value 1. In addition, when an action is observed at a level it gets value 1. The process incrementally builds a plan graph

and updates it level by level. The values of propositions and actions are updated according to the following rules:

1. An action in the plan graph gets value -1 when any of its preconditions or any of its effects is -1.
2. An action in the plan graph gets value 1 when it is the sole producer of an effect that has value 1, `noop` included.
3. A proposition in the plan graph gets value -1 when all of its consumers or all of its producers are -1, `noop` included.
4. A proposition in the plan graph gets value 1 when any of its consumers or any of its producers is 1, `noop` included.

The process results in a plan graph where each proposition and each action is labeled as 1, -1, or 0. Those propositions and actions identified as -1 can be ignored for plan recognition purposes, meaning that these are pruned from the resulting plan graph.

This technique assumes we know the time at which each action has been observed. To relax this assumption, we developed a modified version of this pruning technique. Like Ramírez and Geffner (2010), we assume that the sequence of actions is sequential. Initially, an *earliest time step* (*ets*) i is assigned to each action o in the observed sequence. The *ets* is given by the order of each action in the observed sequence. That is, given $(o_0, o_1, ..., o_i)$, the *ets* for each action is: *ets*(o_0)=0, *ets*(o_1)=1, *ets*(o_2)=2, etc. When the pruning process starts, we establish that an observed action o may be observed at the assigned level i if all its preconditions are true (value 1) and/or unknown (value 0), and they are not mutually exclusive at level $i-1$. Otherwise, the action cannot be executed at that level, which results in an update of the *ets* of each remaining action in the observed sequence. The result of this updating is that each observed action is assumed to occur at the earliest possible time consistent with both the observation sequence and the constraints found in constructing the plan graph, using the interaction information. To illustrate this propagation and pruning technique, consider a simple problem with three operators:

$$\begin{aligned} A &: y \rightarrow z \\ B &: y \rightarrow, \neg y, t \\ C &: t \rightarrow k, \neg t \end{aligned} \tag{11}$$

Suppose the sequence of observed actions is A and C, with initial *ets* 0 and 1 respectively. As a result of the propagation, z must be true (have value 1) at level 1 because A was observed. Since A and B are mutex, B and its effects t and $\neg y$ are false (have value -1) at level 0. C is initially assumed to be at level 1, but this cannot be the case because its precondition t is false at level 0. Therefore, the *ets* for C is updated to 2. At level 2, k and $\neg t$ must be true because C was observed. (This results in t being true at level 1.) Since A and C, and B and C are mutex, A, B, $\neg y$, and t are not possible (have value -1) at level 2. B is unknown (has value 0) at level 1 since there is not enough information to determine whether it is true or false. The proposition y is true at level 0 since the initial state is assumed to be true, and is unknown at level 1 because there is not enough information to determine whether it is true or false. However, it is false at level 2 due to the mutex relation between C and noop-y. Proposition z is true at each level since there are no operators in the domain that delete it.

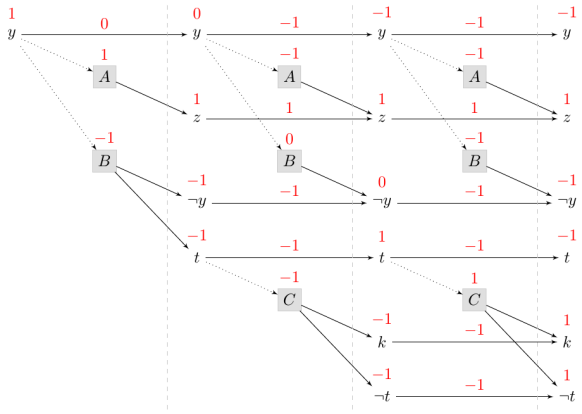


Figure 3: A plan graph with status values of propositions and actions

Fast Goal Recognition

The union of the plan graph cost estimation and the observation pruning techniques results in a heuristic approach that allows fast estimation of cost differences $\Delta(G, O)$, giving us probability estimates for the possible goals $G \in \mathcal{G}$. The steps are:

1. Build a plan graph for the problem P (domain plus initial conditions) and propagate cost and interaction information through this plan graph according to the technique early described.
2. For each (possibly conjunctive) goal $G \in \mathcal{G}$ estimate the $Cost(G)$ from the plan graph using equation (5).
3. Prune the plan graph, based on the observed actions O, using the technique early described.
4. Compute new cost and interaction estimates for this pruned plan graph, considering only those propositions and actions labeled 0, or 1.
5. For each (possibly conjunctive) goal $G \in \mathcal{G}$ estimate the $Cost(G|O)$ from the cost and interaction estimates in the pruned plan graph, again using equation (5). The pruned cost-plan graph may discard propositions or/and actions in the cost-plan graph necessary to reach the goal. This constraint provides a way to discriminate possible goals. However, it may imply that 1) the real goal is discarded, 2) the calculated costs are less accurate. Therefore, computation of $Cost(G|O)$ has been developed under two strategies:
   (a) $Cost(G|O)$ is computed using the pruned cost-plan graph.
   (b) $Cost(G|O)$ is computed after the pruned cost-plan graph is expanded to quiescence again. This will reintroduce any pruned goals that are still possible given the observations.
6. For each goal $G \in \mathcal{G}$, compute $\Delta(G, O)$, and using equation (10) compute the probability $Pr(G|O)$ for the goal given the observations.

To illustrate this computation, consider again actions A, B, and C from equation (11), and the plan graph shown in Figure 3. Suppose that A, B, and C have costs 2, 1, and 3 respectively, and that the possible goals are $g_1 = \{z, k\}$ and $g_2 = \{z, t\}$. Propagating cost and interaction information through the plan graph, we get $Cost(t) = 1$, $Cost(z) = 2$, $Cost(k) = 4$, and interaction values $I(k, t) = \infty$ and $I(k, z) = 0$ at level 3. Now consider the hypothesis $g_1 = \{z, k\}$; in order to compute $Cost(k \wedge z)$, we use the cost and interaction information propagated through the plan graph. In order to compute $Cost(k \wedge z|O)$, the cost and interaction information is propagated again only in those actions with status 1 and 0. In our example, these costs are:

$$Cost(k \wedge z) \approx Cost(z) + Cost(k) + I(k, z) = 2 + 4 + 0 = 6$$

$$Cost(k \wedge z|O) \approx Cost(k) + Cost(z) + I(k, z) = 4 + 2 - 1 = 5$$

Thus, the cost difference is:

$$\Delta(g_1, O) = Cost(g_1|O) - Cost(g_1) = 5 - 6 = -1$$

As a result:

$$Pr(O|g_1) = \frac{e^{-(-1)}}{1 + e^{-(-1)}} = 0.73$$

For the hypothesis $g_2 = \{z, t\}$, the plan graph dismisses this hypothesis as a solution because once the plan graph is pruned, propositions *t* and *z* are labeled as -1. Therefore:

$$Cost(k \wedge t|O) \approx Cost(k) + Cost(t) + I(k, t) = \infty$$

So:

$$Pr(O|g_2) = \frac{e^{-\infty}}{1 + e^{-\infty}} = \frac{0}{1} = 0$$

If we expand the pruned cost-plan graph until quiescence again, the solution is still the same because A and B are permanently mutually exclusive.

Assuming uniform priors, $Pr(G)$, after normalizing the probabilities, we get that $Pr(g_1|O) = 1$ and $Pr(g_2|O) = 0$, so the goal g_1 is certain in this simple example, given the observations of actions A and C.

Experimental Results

We conducted an experimental evaluation on planning domains used by Ramírez: BlocksWord, Intrusion, Kitchen, and Logistics. Each domain has 15 problems. The hypotheses set and actual goal for each problem were chosen at random with the priors on the goal sets assumed to be uniform. For each problem in each of the domains, we ran the LAMA planner (Richter and Westphal, 2010) to solve the problem for the actual goal. The set of observed actions for each recognition problem was taken to be a subset of this plan solution, ranging from 100% of the actions, down to 10% of the actions.

Ramírez evaluates his technique using an optimal planner HSP$^*_f$, and LAMA, a satisficing planner that is used in two modes: as a greedy planner that stops when it finds the first plan (LAMA$_G$), and as a planner that returns the best plan found in a given time limit (LAMA). For purposes of this test, Ramírez technique is also evaluated using the heuristic h^s_a, which was used in (Ramírez and Geffner, 2009). Like our technique, this requires no search since the cost is given by a heuristic function. We compare our goal recognition technique, GR, against Ramírez's technique for those three planners and h^s_a, on the aforementioned domains, using a range of time limits from 5 seconds up to 1800 seconds. We present three variations of our technique, with and without extension of the plan graph after pruning:

- GR$_I$: cost propagation in a plan graph considers interaction information.
- GR$_I$E: same as above, but the pruned cost-plan graph is expanded until quiescence.
- GR$_{add}$: traditional cost propagation in a plan graph.

Table 4 summarizes the results. For each planner, each column shows average performance over the 15 problems in each domain. The first row in the table represents the optimal solution where HSP$^*_f$ (HSP$^*_f$u) was allowed to run for an unlimited amount of time. The other rows represent different measures of quality and performance:

- T shows the average time in seconds taken for solving the problems.
- Q shows the fraction of times the actual goal was among the goals found to be the most likely.
- S shows the *spread*, that is, the average number of goals in $\mathcal{G}$ that were found to be the most likely.
- Q_{20} and Q_{50} show the fraction of times the actual goal is in the top 20% and top 50% of the ranked goals. Although Q might be less than 1 for some problem, Q_{20} or Q_{50} might be 1, indicating that the actual goal was *close* to the top.
- d is the mean distance between the probability scores produced for all the goal candidates, and the probability scores produced by gHSP$^*_f$u. More precisely, if the set of possible goals is $\{G_1, ..., G_n\}$, a method produces probabilities $\{e_1, ..., e_n\}$ for those goals, and gHSP$^*_f$u produces $\{p_1, ..., p_n\}$, d is defined as:

$$d = 1/n \sum_{i=1}^{n} |e_i - p_i| \quad (12)$$

The use of an optimal planner like HSP$^*_f$u is generally impractical for real-time goal recognition on any non-trivial domain. Surprisingly, LAMA does not perform any better on the harder domains, and the solution quality is uneven. Greedy LAMA is much faster, but still no faster than HSP$^*_f$u in the blocks world domain. The h^s_a heuristic (Ramírez and Geffner, 2009) for approximating costs is quite fast, but the cost estimates are not very accurate, leading to poor quality results for goal recognition. The GR$_I$ heuristic is also quite fast, but yields much better results. On the harder domains, it is two orders of magnitude faster than HSP$^*_f$u, LAMA, or Greedy LAMA, and yields results of comparable quality to both LAMA and Greedy LAMA for the higher observation percentages.

Conclusions and Future Work

This paper presents a heuristic estimator h^I based on a cost-plan graph and interaction information to compute more accurate cost estimates. This heuristic provides cost estimates

Table 4: Goal recognition with random observations

Domain		Blocks					Intrusion					Kitchen					Logistics				
Approach	%O	100	70	50	30	10	100	70	50	30	10	100	70	50	30	10	100	70	50	30	10
HSP^*_fu	T	558.08	419.45	379.81	357.94	357.94	447.41	281.12	151.37	3.58	3.55	480.51	171.08	49.62	37.93	37.92	36.26	32.46	14.08	7.04	7.04
	Q	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0.93	0.66	0.8	0.8
	S	1.06	1.13	4.06	11.46	11.46	1	1	1.06	4.46	4.6	1	1	1.33	1.4	1.4	1	1.13	2.26	3.6	3.6
LAMA	T	1603.24	1522.96	1260.6	1077.62	1082.15	92.85	89.01	64.97	17.57	17.48	26.33	26.2	20.45	20.3	20.3	45.17	41.71	26.53	11.38	11.39
	Q	0.33	0.8	0.8	0.93	1	0.93	1	1	1	1	0.73	1	1	1	1	1	0.93	0.66	0.8	0.8
	S	1	1.13	3.86	10.4	10.93	0.93	1	1.06	4.46	4.6	0.73	1	1.33	1.4	1.4	1	1.13	2.26	3.6	3.6
	Q_{20}	1	1	1	1	1	0.93	1	1	1	1	0.73	1	1	1	1	1	1	0.93	1	1
	Q_{50}	1	1	1	1	1	0.93	1	1	1	1	0.73	1	1	1	1	1	1	1	1	1
	d	0.24	0.316	0.192	0.048	0.068	1.739	1.12	0.095	7×10^{-6}	7×10^{-6}	0.358	3×10^{-3}	0.016	0.012	0.012	0.019	0.673	1.051	0.956	0.956
$LAMA_G$	T	849.08	840.76	814.95	803.04	809.01	3.32	2.63	2.21	2.08	2.08	0.42	0.36	0.33	0.32	0.32	5.47	4.95	4.5	4.33	4.36
	Q	0.93	0.8	0.73	0.66	0.46	0	0.4	1	1	1	0.73	1	1	1	1	1	0.8	0.4	0.46	0.46
	S	1	1.2	3	6.2	4.2	0	0.4	1.13	4.46	4.6	0.73	1	1.33	1.4	1.4	1	1.2	1.8	2.93	2.93
	Q_{20}	0.93	1	1	1	1	0	0.4	1	1	1	0.73	1	1	1	1	1	1	0.66	0.6	0.6
	Q_{50}	0.93	1	1	1	1	0	0.4	1	1	1	0.73	1	1	1	1	1	1	0.93	0.86	0.86
	d	0.068	0.34	0.404	0.322	0.341	1.86	1.12	0.108	7×10^{-6}	7×10^{-6}	0.358	3×10^{-3}	0.016	0.012	0.012	0.019	0.675	1.179	1.063	1.063
h^*_a	T	1.04	0.89	0.81	0.81	0.8	0.7	0.46	0.39	0.35	0.36	0.066	0.049	0.044	0.045	0.046	0.61	0.55	0.51	0.51	0.51
	Q	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0.13	0.06	0.13	0.06	0.06
	S	20.26	20.26	20.26	20.26	20.26	16.66	16.66	16.66	16.66	16.66	3	3	3	3	3	2.8	1.86	1.93	1	1
	Q_{20}	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0.93	0.93	0.86	0.86	0.86
	Q_{50}	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0.93	0.93	0.86	0.86	0.86
	d	0.092	0.087	0.062	0.035	0.035	0.123	0.124	0.119	0.075	0.073	0.443	0.436	0.284	0.226	0.226	0.123	0.117	0.099	0.074	0.074
GR_I	T	1.007	1.029	1.265	1.452	1.452	0.885	0.493	0.212	0.209	0.21	0.261	0.195	0.140	0.135	0.135	0.886	1.015	1.19	1.266	1.271
	Q	1	0.66	0.4	0.13	0.13	1	1	0.93	0.93	0.93	1	1	1	1	1	1	0.86	0.53	0.6	0.6
	S	1.06	0.8	1.06	1.73	1.73	1	1	1	4.4	4.53	1	1	1.2	1.26	1.26	1	1.26	1.6	2.46	2.46
	Q_{20}	1	0.66	0.53	0.46	0.46	1	1	1	0.93	0.93	1	1	1	1	1	1	0.93	0.66	0.73	0.73
	Q_{50}	1	0.73	0.73	0.8	0.8	1	1	1	1	1	1	1	1	1	1	1	0.93	0.8	0.86	0.86
	d	0.149	0.962	0.751	0.336	0.336	3.2×10^{-4}	0.055	0.872	0.241	0.241	3.98×10^{-4}	0.036	0.107	0.192	0.192	0.264	0.786	1.163	0.718	0.718
GR_IE	T	9.936	8.211	4.542	3.696	3.687	1.293	1.191	0.996	0.743	0.738	0.287	0.266	0.230	0.193	0.193	7.535	4.24	3.016	2.842	2.834
	Q	0.46	0.53	0.46	0.13	0.13	1	1	0.93	0.93	0.93	1	1	1	1	1	0.86	0.66	0.66	0.6	0.6
	S	1.26	1.13	1.86	1.73	1.73	1	1	1	4.4	4.53	1	1	1.2	1.26	1.26	1.13	1.4	1.8	2.8	2.8
	Q_{20}	0.46	0.73	0.66	0.4	0.4	1	1	1	0.93	0.93	1	1	1	1	1	0.86	0.8	0.8	0.73	0.73
	Q_{50}	0.46	0.8	0.86	0.8	0.8	1	1	1	1	1	1	1	1	1	1	0.93	1	0.86	0.86	0.86
	d	1.094	1.025	0.742	0.358	0.358	3.2×10^{-4}	0.055	0.872	0.241	0.241	3.98×10^{-4}	0.036	0.107	0.192	0.192	0.387	0.943	1.107	0.771	0.771
GR_{add}	T	0.761	0.609	0.643	0.782	0.783	0.886	0.491	0.196	0.192	0.193	0.258	0.191	0.136	0.128	0.129	0.806	0.405	0.448	0.508	0.51
	Q	1	0.46	0.46	0.46	0.46	1	1	1	1	0.93	1	1	1	1	1	1	0.4	0.46	0.6	0.6
	S	1	0.53	1.2	2.06	2.06	1	1.13	1.13	4.06	3.93	1	1	1.33	1.4	1.4	1	1	2.13	2.93	2.93
	Q_{20}	1	0.46	0.6	0.6	0.6	1	1	1	1	0.93	1	1	1	1	1	1	0.46	0.6	0.66	0.66
	Q_{50}	1	0.46	0.6	0.73	0.73	1	1	1	1	1	1	1	1	1	1	1	0.53	0.73	0.8	0.8
	d	0.088	1.084	1.036	1.004	1.004	0.151	0.764	1.25	0.311	0.285	2×10^{-6}	0.038	5×10^{-6}	0.022	0.022	0.011	1.196	1.29	0.899	0.899

that are substantially closer to the optimal and more consistent. Regardless of the quality estimates generated by h^I, its use in classical planning does not pay off because of the computational overhead. However, it is very useful for goal recognition to infer probability estimates for the possible goals.

Acknowledgments

This work was funded by the JCCM project PEII-2014-015A, USRA, and NASA Ames Research Center.

References

Blum, A., and Furst, M. 1997. Fast planning through planning graph analysis. *AI* 90:281—300.

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *JAIR* 129:5—33.

Bonet, B.; Loerincs, G.; and Geffner, H. 1997. A robust and fast action selection mechanism for planning. In *AAAI'97*.

Bryce, D., and Smith, D. E. 2006. Using interaction to compute better probability estimates in plan graphs. In *ICAPS'06 Workshop on Planning Under Uncertainty and Execution Control for Autonomous Systems*.

Chen, Y.; W., W. B.; and Chih-Wei, H. 2006. Temporal planning using subgoal partitioning and resolution in SGPlan. *JAIR* 26:323—369.

Do, M., and Kambhampati, S. 2002. Planning graph-based heuristics for cost-sensitive temporal planning. In *AIPS'02*. Toulouse, France.

Gerevini, A.; Saetti, A.; and Serina, I. 2003. Planning through stochastic local search and temporal action graphs in LPG. *JAIR* 20:239—290.

Haslum, P., and Geffner, H. 2000. Admissible heuristic for optimal planning. In *AIPS'00*.

Haslum, P. 2008. Additive and reversed relaxed reachability heuristics revisited. In *IPC-08*.

Hoffmann, J. 2003. The Metric-FF planning system: Translating "ignoring delete lists" to numeric state variables. *JAIR* 20:291—341.

Jigui, S., and Minghao, Y. 2007. Recognizing the agent's goals incrementally: planning graph as a basis. *In Frontiers of Computer Science in China* 1(1):26—36.

Keyder, E., and Geffner, H. 2007. Heuristics for planning with action costs. In *CAEPIA'07*.

Nguyen, X.; Kambhampati, S.; and Nigenda, R. S. 2002. Planning graph as the basis for deriving heuristics for plan synthesis by state space and csp search. *AI* 135(1-2):73—123.

Ramírez, M., and Geffner, H. 2009. Plan recognition as planning. In *IJCAI'09*.

Ramírez, M., and Geffner, H. 2010. Probabilistic plan recognition using off-the-shelf classical planners. In *AAAI'10*.

Ramírez, M. 2012. *Plan Recognition as Planning*. Ph.D. Dissertation, Universitat Pompeu Fabra, Barcelona, Spain.

Richter, S., and Westphal, M. 2010. The LAMA planner: guiding cost-based anytime planning with landmarks. *JAIR* 39:127—177.

Red-Black Planning: A New Tractability Analysis and Heuristic Function

Daniel Gnad and **Jörg Hoffmann**
Saarland University
Saarbrücken, Germany
{gnad, hoffmann}@cs.uni-saarland.de

Abstract

Red-black planning is a recent approach to partial delete relaxation, where red variables take the relaxed semantics (accumulating their values), while black variables take the regular semantics. Practical heuristic functions can be generated from tractable sub-classes of red-black planning. Prior work has identified such sub-classes based on the black causal graph, i. e., the projection of the causal graph onto the black variables. Here, we consider cross-dependencies between black and red variables instead. We show that, if no red variable relies on black preconditions, then red-black plan generation is tractable in the size of the black state space, i. e., the product of the black variables. We employ this insight to devise a new red-black plan heuristic in which variables are painted black starting from the causal graph leaves. We evaluate this heuristic on the planning competition benchmarks. Compared to a standard delete relaxation heuristic, while the increased runtime overhead often is detrimental, in some cases the search space reduction is strong enough to result in improved performance overall.

Introduction

In classical AI planning, we have a set of finite-domain state variables, an initial state, a goal, and actions described in terms of preconditions and effects over the state variables. We need to find a sequence of actions leading from the initial state to a goal state. One prominent way of addressing this is heuristic forward state space search, and one major question in doing so is how to generate the heuristic function automatically, i. e., just from the problem description without any further human user input. We are concerned with that question here, in satisficing planning where no guarantee on plan quality needs to be provided. The most prominent class of heuristic functions for satisficing planning are *relaxed plan* heuristics (e. g. (McDermott 1999; Bonet and Geffner 2001; Hoffmann and Nebel 2001; Gerevini, Saetti, and Serina 2003; Richter and Westphal 2010)).

Relaxed plan heuristics are based on the *delete* (or *monotonic*) *relaxation*, which assumes that state variables accumulate their values, rather than switching between them. Optimal delete-relaxed planning still is **NP**-hard, but satisficing delete-relaxed planning is polynomial-time (Bylander 1994). Given a search state s, relaxed plan heuristics generate a (not necessarily optimal) delete-relaxed plan for s, resulting in an inadmissible heuristic function which tends to be very informative on many planning benchmarks (an explicit analysis has been conducted by Hoffmann (2005)).

Yet, like any heuristic, relaxed plan heuristics also have significant pitfalls. A striking example (see, e. g., (Coles et al. 2008; Nakhost, Hoffmann, and Müller 2012; Coles et al. 2013)) is "resource persistence", that is, the inability to account for the consumption of non-replenishable resources. As variables never lose any "old" values, the relaxation pretends that resources are never actually consumed. For this and related reasons, the design of heuristics that take *some* deletes into account has been an active research area from the outset (e. g. (Fox and Long 2001; Gerevini, Saetti, and Serina 2003; Helmert 2004; van den Briel et al. 2007; Helmert and Geffner 2008; Coles et al. 2008; Keyder and Geffner 2008; Baier and Botea 2009; Keyder, Hoffmann, and Haslum 2012; Coles et al. 2013; Keyder, Hoffmann, and Haslum 2014). We herein continue the most recent approach along these lines, *red-black planning* as introduced by Katz et al. (2013b).

Red-black planning delete-relaxes only a subset of the state variables, called "red", which accumulate their values; the remaining variables, called "black", retain the regular value-switching semantics. The idea is to obtain an inadmissible yet informative heuristic in a manner similar to relaxed plan heuristics, i. e. by generating some (not necessarily optimal) red-black plan for any given search state s. For this to make sense, such red-black plan generation must be sufficiently fast. Therefore, after introducing the red-black planning framework, Katz et al. embarked on a line of work generating red-black plan heuristics based on tractable fragments. These are characterized by properties of the projection of the causal graph – a standard structure capturing state variable dependencies – onto the black variables (Katz, Hoffmann, and Domshlak 2013a; Katz and Hoffmann 2013; Domshlak, Hoffmann, and Katz 2015). *Cross-dependencies between black and red variables were not considered at all yet.* We fill that gap, approaching "from the other side" in that we analyze *only* such cross-dependencies. We ignore the structure inside the black part, assuming that there is a single black variable only; in practice, that "single variable" will correspond to the cross-product of the black variables.

Distinguishing between (i) black-precondition-to-red-effect, (ii) red-precondition-to-black-effect, and (iii) mixed-red-black-effect dependencies, and assuming there is a sin-

gle black variable, we establish that (i) alone governs the borderline between **P** and **NP**: If we allow type (i) dependencies, deciding red-black plan existence is **NP**-complete, and if we disallow them, red-black plan generation is polynomial-time. Katz et al. also considered the single-black-variable case. Our hardness result strengthens theirs in that it shows only type (i) dependencies are needed. Our tractability result is a major step forward in that *it allows to scale the size of the black variable*, in contrast to Katz et al.'s algorithm whose runtime is exponential in that parameter. Hence, in contrast to Katz et al.'s algorithm, ours is practical. It leads us to a new red-black plan heuristic, whose painting strategy draws a "horizontal line" through the causal graph viewed as a DAG of strongly connected components (SCC), with the roots at the top and the leaves at the bottom. The part above the line gets painted red, the part below the line gets painted black, so type (i) dependencies are avoided.

Note that, by design, the black variables must be "close to the causal graph leaves". This is in contrast with Katz et al.'s red-black plan heuristics, which attempt to paint black the variables "close to the causal graph root", to account for the to-and-fro of these variables when servicing other variables (e. g., a truck moving around to service packages). Indeed, if the black variables are causal graph leaves, then provably no information is gained over a standard relaxed plan heuristic (Katz, Hoffmann, and Domshlak 2013b). However, in our new heuristic we paint black *leaf SCCs*, as opposed to *leaf variables*. As we point out using an illustrative example, this can result in better heuristic estimates than a standard relaxed plan, and even than a red-black plan when painting the causal graph roots black. That said, in the International Planning Competition (IPC) benchmarks, this kind of structure seems to be rare. Our new heuristic often does not yield a search space reduction so its runtime overhead ends up being detrimental. Katz et al.'s heuristic almost universally performs better. In some cases though, our heuristic does reduce the search space dramatically relative to standard relaxed plans, resulting in improved performance.

Preliminaries

Our approach is placed in the *finite-domain representation (FDR)* framework. To save space, we introduce FDR and its delete relaxation as special cases of red-black planning. A *red-black (RB)* planning task is a tuple $\Pi = \langle V^{\mathsf{B}}, V^{\mathsf{R}}, A, I, G\rangle$. V^{B} is a set of *black state variables* and V^{R} is a set of *red state variables*, where $V^{\mathsf{B}} \cap V^{\mathsf{R}} = \emptyset$ and each $v \in V := V^{\mathsf{B}} \cup V^{\mathsf{R}}$ is associated with a finite domain $\mathcal{D}(v)$. The *initial state* I is a complete assignment to V, the *goal* G is a partial assignment to V. Each action a is a pair $\langle \mathsf{pre}(a), \mathsf{eff}(a)\rangle$ of partial assignments to V called *precondition* and *effect*. We often refer to (partial) assignments as sets of *facts*, i. e., variable-value pairs $v = d$. For a partial assignment p, $\mathcal{V}(p)$ denotes the subset of V instantiated by p. For $V' \subseteq \mathcal{V}(p)$, $p[V']$ denotes the value of V' in p.

A state s assigns each $v \in V$ a non-empty subset $s[v] \subseteq \mathcal{D}(v)$, where $|s[v]| = 1$ for all $v \in V^{\mathsf{B}}$. An action a is applicable in state s if $\mathsf{pre}(a)[v] \in s[v]$ for all $v \in \mathcal{V}(\mathsf{pre}(a))$. Applying a in s changes the value of $v \in \mathcal{V}(\mathsf{eff}(a)) \cap V^{\mathsf{B}}$ to $\{\mathsf{eff}(a)[v]\}$, and changes the value of $v \in \mathcal{V}(\mathsf{eff}(a)) \cap V^{\mathsf{R}}$ to $s[v] \cup \{\mathsf{eff}(a)[v]\}$. The resulting state is denoted $s[\![a]\!]$. By $s[\![\langle a_1, \ldots, a_k\rangle]\!]$ we denote the state obtained from sequential application of $a_1, \ldots, a_k$. An action sequence $\langle a_1, \ldots, a_k\rangle$ is a *plan* if $G[v] \in I[\![\langle a_1, \ldots, a_k\rangle]\!][v]$ for all $v \in \mathcal{V}(G)$.

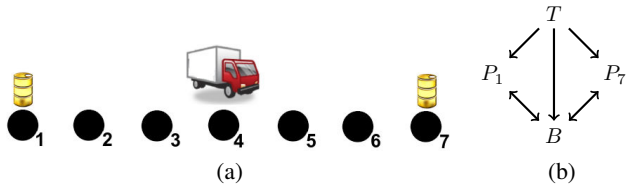
Π is a *finite-domain representation (FDR)* planning task if $V = V^{\mathsf{B}}$, and is a *monotonic finite-domain representation (MFDR)* planning task if $V = V^{\mathsf{R}}$. Optimal planning for MFDR tasks is **NP**-complete, but satisficing planning is polynomial-time. The latter can be exploited for deriving (inadmissible) *relaxed plan* heuristics, denoted h^{FF} here. Generalizing this to red-black planning, the *red-black relaxation* of an FDR task Π relative to a *variable painting*, i. e. a subset V^{R} to be painted red, is the RB task $\Pi^{\mathsf{RB}}_{V^{\mathsf{R}}} = \langle V \setminus V^{\mathsf{R}}, V^{\mathsf{R}}, A, I, G\rangle$. A plan for $\Pi^{\mathsf{RB}}_{V^{\mathsf{R}}}$ is a *red-black plan* for Π. Generating optimal red-black plans is **NP**-hard regardless of the painting simply because we always generalize MFDR. The idea is to generate satisficing red-black plans and thus obtain a *red-black plan heuristic* h^{RB} similarly as for h^{FF}. That approach is practical if the variable painting is chosen so that satisficing red-black plan generation is tractable (or sufficiently fast, anyway).

A standard means to identify structure, and therewith tractable fragments, in planning is to capture dependencies between state variables in terms of the *causal graph*. This is a digraph with vertices V. An arc (v, v') is in CG_Π if $v \neq v'$ and there exists an action $a \in A$ such that $(v, v') \in [\mathcal{V}(\mathsf{eff}(a)) \cup \mathcal{V}(\mathsf{pre}(a))] \times \mathcal{V}(\mathsf{eff}(a))$.

Prior work on tractability in red-black planning (Katz, Hoffmann, and Domshlak 2013b; 2013a; Domshlak, Hoffmann, and Katz 2015) considered (a) the "black causal graph" i. e. the sub-graph induced by the black variables only, and (b) the case of a single black variable. Of these, only (a) was employed for the design of heuristic functions. Herein, we improve upon (b). Method (a) is not of immediate relevance to our technical contribution, but we compare to it empirically, specifically to the most competitive heuristic h^{Mercury} as used in the Mercury system that participated in IPC'14 (Katz and Hoffmann 2014). That heuristic exploits the tractable fragment of red-black planning where the black causal graph is acyclic and every black variable is "invertible" in a particular sense. The painting strategy is geared at painting black the "most influential" variables, close to the causal graph roots.

Example 1 *As an illustrative example, we use a simplified version of the IPC benchmark TPP. Consider Figure 1. There is a truck moving along a line $l_1, \ldots, l_7$ of locations. The truck starts in the middle; the goal is to buy two units of a product, depicted in Figure 1 (a) by the barrels, where one unit is on sale at each extreme end of the road map.*

Concretely, say the encoding in FDR is as follows. The state variables are T with domain $\{l_1, \ldots, l_7\}$ for the truck position; B with domain $\{0, 1, 2\}$ for the amount of product bought already; P_1 with domain $\{0, 1\}$ for the amount of product still on sale at l_1; and P_7 with domain $\{0, 1\}$ for the amount of product still on sale at l_7. The initial state is as shown in the figure, i. e., $T = l_3$, $B = 0$, $P_1 = 1$, $P_7 = 1$. The goal is $B = 2$. The actions are:



(a) (b)

Figure 1: Our running example (a), and its causal graph (b).

- *move*(x, y)*: precondition* $\{T = l_x\}$ *and effect* $\{T = l_y\}$*, where* $x, y \in \{1, \ldots, 7\}$ *such that* $|x - y| = 1$*.*
- *buy*(x, y, z)*: precondition* $\{T = l_x, P_x = 1, B = y\}$ *and effect* $\{P_x = 0, B = z\}$*, where* $x \in \{1, 7\}$ *and* $y, z \in \{0, 1, 2\}$ *such that* $z = y + 1$*.*

The causal graph is shown in Figure 1 (b). Note that the variables pertaining to the product, i. e. B, P_1, P_7*, form a strongly connected component because of the "buy" actions.*

Consider first the painting where all variables are red, i. e., a full delete relaxation. A relaxed plan then ignores that, after buying the product at one of the two locations l_1 *or* l_7*, the product is no longer available so we have to move to the other end of the line. Instead, we can buy the product again at the same location, ending up with a relaxed plan of length* 5 *instead of the* 11 *steps needed in a real plan.*

Exactly the same problem arises in h^{Mercury}*: The only "invertible" variable here is* T*. But if we paint only* T *black, then the red-black plan still is the same as the fully delete-relaxed plan (the truck does not have to move back and forth anyhow), and we still get the same goal distance estimate* 5*.*

Now say that we paint T *red, and paint all other variables black. This is the painting our new heuristic function will use. We can no longer cheat when buying the product, i. e., we do need to buy at each of* l_1 *and* l_7*. Variable* T *is relaxed so we require* 6 *moves to reach both these locations, resulting in a red-black plan of length* 8*.*

Tractability Analysis

We focus on the case of a single black variable. This has been previously investigated by Katz et al. (2013b), but scantly only. We will discuss details below; our contribution regards a kind of dependency hitherto ignored, namely *cross-dependencies* between red and black variables:

Definition 1 *Let* $\Pi = \langle V^{\mathsf{B}}, V^{\mathsf{R}}, A, I, G \rangle$ *be a RB planning task. We say that* $v, v' \in V^{\mathsf{B}} \cup V^{\mathsf{R}}$ have different colors *if either* $v \in V^{\mathsf{B}}$ *and* $v' \in V^{\mathsf{R}}$ *or vice versa. The* red-black causal graph CG^{RB}_{Π} *of* Π *is the digraph with vertices* V *and those arcs* (v, v') *from* CG_{Π} *where* v *and* v' *have different colors. We say that* (v, v') is of type*:*

(i) BtoR *if* $v \in V^{\mathsf{B}}$*,* $v' \in V^{\mathsf{R}}$*, and there exists an action* $a \in A$ *such that* $(v, v') \in \mathcal{V}(\mathsf{pre}(a)) \times \mathcal{V}(\mathsf{eff}(a))$*.*

(ii) RtoB *if* $v \in V^{\mathsf{R}}$*,* $v' \in V^{\mathsf{B}}$*, and there exists an action* $a \in A$ *such that* $(v, v') \in \mathcal{V}(\mathsf{pre}(a)) \times \mathcal{V}(\mathsf{eff}(a))$*.*

(iii) EFF *else.*

We investigate the complexity of satisficing red-black planning as a function of allowing vs. disallowing each of the types (i) – (iii) of cross-dependencies individually. We completely disregard the inner structure of the black part of Π, i. e., the subset V^{B} of black variables may be arbitrary. The underlying assumption is that these variables will be pre-composed into a single black variable. Such "pre-composition" essentially means to build the cross-product of the respective variable domains (Seipp and Helmert 2011). We will refer to that cross-product as the *black state space*, and state our complexity results relative to the assumption that $|V^{\mathsf{B}}| = 1$, denoting the single black variable with v^{B}. In other words, our complexity analysis is relative to the size $|\mathcal{D}(v^{\mathsf{B}})|$ of the black state space, as opposed to the size of the input task. From a practical perspective, which we elaborate on in the next section, this makes sense provided the variable painting is chosen so that the black state space is "small".

Katz et al. (2013b) show in their Theorem 1, henceforth called "KatzP", that satisficing red-black plan generation is polynomial-time in case $|\mathcal{D}(v^{\mathsf{B}})|$ is fixed, via an algorithm that is exponential only in that parameter. They show in their Theorem 2, henceforth called "KatzNP", that deciding red-black plan existence is **NP**-complete if $|\mathcal{D}(v^{\mathsf{B}})|$ is allowed to scale. They do not investigate any structural criteria distinguishing sub-classes of the single black variable case. We close that gap here, considering the dependency types (i) – (iii) of Definition 1. The major benefit of doing so will be a polynomial-time algorithm for scaling $|\mathcal{D}(v^{\mathsf{B}})|$.

Switching each of (i) – (iii) on or off individually yields a lattice of eight sub-classes of red-black planning. It turns out that, as far as the complexity of satisficing red-black plan generation is concerned, this lattice collapses into just two classes, characterized by the presence or absence of dependencies (i): If arcs of type BtoR are allowed, then the problem is **NP**-complete even if arcs of types RtoB and EFF are disallowed. If arcs of type BtoR are disallowed, then the problem is polynomial-time even if arcs of types RtoB and EFF are allowed. We start with the negative result:

Theorem 1 *Deciding red-black plan existence for RB planning tasks with a single black variable, and without* CG^{RB}_{Π} *arcs of types RtoB and EFF, is* ***NP****-complete.*

Proof: Membership follows from KatzNP. (Plan length with a single black variable is polynomially bounded, so this holds by guess-and-check.)

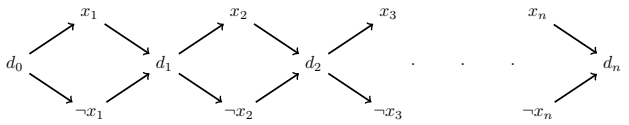


Figure 2: Illustration of the black variable v^{B} in the SAT reduction in the proof of Theorem 1.

We prove hardness by a reduction from SAT. Consider a CNF formula ϕ with propositional variables $x_1, \ldots, x_n$ and clauses $c_1, \ldots, c_m$. Our RB planning task has m Boolean red variables v^{R}_i, and the single black variable v^{B} has domain $\{d_0, \ldots, d_n\} \cup \{x_i, \neg x_i \mid 1 \leq i \leq n\}$. In the initial state, all v^{R}_j are set to false and v^{B} has value d_0. The goal is for all v^{R}_j to be set to true. The actions moving v^{B} have

Algorithm NoBtoR-Planning:
$R := I[V^{\mathsf{R}}] \cup$ RedFixedPoint(A^{R})
if $G[V^{\mathsf{R}}] \subseteq R$ and BlackReachable$(R, I[v^{\mathsf{B}}], G[v^{\mathsf{B}}])$ **then**
  return "solvable" /* case (a) */
endif
$R := I[V^{\mathsf{R}}] \cup$ RedFixedPoint$(A^{\mathsf{R}} \cup A^{\mathsf{RB}})$
if $G[V^{\mathsf{R}}] \subseteq R$ **then**
  for $a \in A^{\mathsf{RB}}$ s.t. $\mathsf{pre}(a) \subseteq R$ **do**
    if BlackReachable$(R, \mathsf{eff}(a)[v^{\mathsf{B}}], G[v^{\mathsf{B}}])$ **then**
      return "solvable" /* case (b) */
    endif
  endfor
endif
return "unsolvable"

Figure 3: Algorithm used in the proof of Theorem 2.

preconditions and effects only on v^{B}, and are such that we can move as shown in Figure 2, i. e., for $1 \leq i \leq n$: from d_{i-1} to x_i; from d_{i-1} to $\neg x_i$; from x_i to d_i; and from $\neg x_i$ to d_i. For each literal $l \in c_j$ there is an action allowing to set v^{R}_j to true provided v^{B} has the correct value, i. e., for $l = x_i$ the precondition is $v^{\mathsf{B}} = x_i$, and for $l = \neg x_i$ the precondition is $v^{\mathsf{B}} = \neg x_i$. This construction does not incur any RtoB or EFF dependencies. The paths v^{B} can take correspond exactly to all possible truth value assignments. We can achieve the red goal iff one of these paths visits at least one literal from every clause, which is the case iff ϕ is satisfiable. □

The hardness part of KatzNP relies on EFF dependencies. Theorem 1 strengthens this in showing that these dependencies are not actually required for hardness.

Theorem 2 *Satisficing plan generation for RB planning tasks with a single black variable, and without CG^{RB}_Π arcs of type BtoR, is polynomial-time.*

Proof: Let $\Pi = \langle \{v^{\mathsf{B}}\}, V^{\mathsf{R}}, A, I, G \rangle$ as specified. We can partition A into the following subsets:

- $A^{\mathsf{B}} := \{a \in A \mid \mathcal{V}(\mathsf{eff}(a)) \cap V^{\mathsf{B}} = \{v^{\mathsf{B}}\}, \mathcal{V}(\mathsf{eff}(a)) \cap V^{\mathsf{R}} = \emptyset\}$ are the actions affecting only the black variable. These actions may have red preconditions.
- $A^{\mathsf{R}} := \{a \in A \mid \mathcal{V}(\mathsf{eff}(a)) \cap V^{\mathsf{B}} = \emptyset, \mathcal{V}(\mathsf{eff}(a)) \cap V^{\mathsf{R}} \neq \emptyset\}$ are the actions affecting only red variables. As there are no CG^{RB}_Π arcs of type BtoR, the actions in A^{R} have no black preconditions.
- $A^{\mathsf{RB}} := \{a \in A \mid \mathcal{V}(\mathsf{eff}(a)) \cap V^{\mathsf{B}} = \{v^{\mathsf{B}}\}, \mathcal{V}(\mathsf{eff}(a)) \cap V^{\mathsf{R}} \neq \emptyset\}$ are the actions affecting both red variables and the black variable. As there are no CG^{RB}_Π arcs of type BtoR, the actions in A^{RB} have no black preconditions.

Consider Figure 3. By *RedFixedPoint*(A') for a subset $A' \subseteq A$ of actions without black preconditions, we mean all red facts reachable using only A', ignoring any black effects. This can be computed by building a relaxed planning graph over A'. By *BlackReachable*(R, d, d') we mean the question whether there exists an A^{B} path moving v^{B} from d to d', using only red preconditions from R.

Clearly, NoBtoR-Planning runs in polynomial time. If it returns "solvable", we can construct a plan π^{RB} for Π as follows. In case (a), we obtain π^{RB} by any sequence of A^{R} actions establishing RedFixedPoint(A^{R}) (there are neither black preconditions nor black effects), and attaching a sequence of A^{B} actions leading from $I[v^{\mathsf{B}}]$ to $G[v^{\mathsf{B}}]$. In case (b), we obtain π^{RB} by: any sequence of A^{R} actions establishing RedFixedPoint$(A^{\mathsf{R}} \cup A^{\mathsf{RB}})$ (there are no black preconditions); attaching the A^{RB} action a successful in the for-loop (which is applicable due to $\mathsf{pre}(a) \subseteq R$ and $\mathcal{V}(\mathsf{pre}(a)) \cap V^{\mathsf{B}} = \emptyset$); and attaching a sequence of A^{B} actions leading from $\mathsf{eff}(a)[v^{\mathsf{B}}]$ to $G[v^{\mathsf{B}}]$. Note that, after RedFixedPoint$(A^{\mathsf{R}} \cup A^{\mathsf{RB}})$, only a single A^{RB} action a is necessary, enabling the black value $\mathsf{eff}(a)[v^{\mathsf{B}}]$ from which the black goal is A^{B}-reachable.

If there is a plan π^{RB} for Π, then NoBtoR-Planning returns "solvable". First, if π^{RB} does not use any A^{RB} action, i. e. π^{RB} consists entirely of A^{R} and A^{B} actions, then case (a) will apply because RedFixedPoint(A^{R}) contains all we can do with the former, and BlackReachable$(R, I[v^{\mathsf{B}}], G[v^{\mathsf{B}}])$ examines all we can do with the latter. Second, say π^{RB} does use at least one A^{RB} action. RedFixedPoint$(A^{\mathsf{R}} \cup A^{\mathsf{RB}})$ contains all red facts that can be achieved in Π, so in particular *(*) RedFixedPoint$(A^{\mathsf{R}} \cup A^{\mathsf{RB}})$ contains all red facts true along π^{RB}*. Let a be the last A^{RB} action applied in π^{RB}. Then π^{RB} contains a path from $\mathsf{eff}(a)[v^{\mathsf{B}}]$ to $G[v^{\mathsf{B}}]$ behind a. With (*), $\mathsf{pre}(a) \subseteq R$ and BlackReachable$(R, \mathsf{eff}(a)[v^{\mathsf{B}}], G[v^{\mathsf{B}}])$ succeeds, so case (b) will apply. □

In other words, if (a) no A^{RB} action is needed to solve Π, then we simply execute a relaxed planning fixed point prior to moving v^{B}. If (b) such an action is needed, then we mix A^{RB} with the fully-red ones in the relaxed planning fixed point, which works because, having no black preconditions, once an A^{RB} action has become applicable, it remains applicable. Note that the case distinction (a) vs. (b) is needed: When making use of the "large" fixed point RedFixedPoint$(A^{\mathsf{R}} \cup A^{\mathsf{RB}})$, there is no guarantee we can get v^{B} back into its initial value afterwards.

Example 2 *Consider again our illustrative example (cf. Figure 1), painting T red and painting all other variables black. Then v^{B} corresponds to the cross-product of variables B, P_1, and P_7; A^{B} contains the "buy" actions, A^{R} contains the "move" actions, and A^{RB} is empty.*

The call to RedFixedPoint(A^{R}) in Figure 3 results in R containing all truck positions, $R = \{T = l_1, \ldots, T = l_7\}$. The call to BlackReachable$(R, I[v^{\mathsf{B}}], G[v^{\mathsf{B}}])$ then succeeds as, given we have both truck preconditions $T = l_1$ and $T = l_7$ required for the "buy" actions, indeed the black goal $B = 2$ is reachable. The red-black plan extracted will contain a sequence of moves reaching all of $\{T = l_1, \ldots, T = l_7\}$, followed by a sequence of two "buy" actions leading from $I[v^{\mathsf{B}}] = \{B = 0, P_1 = 1, P_2 = 1\}$ to $G[v^{\mathsf{B}}] = \{B = 2\}$.

Theorem 2 is a substantial improvement over KatzP in terms of the scaling behavior in $|\mathcal{D}(v^{\mathsf{B}})|$. KatzP is based on an algorithm with runtime exponential in $|\mathcal{D}(v^{\mathsf{B}})|$. Our NoBtoR-Planning has low-order polynomial runtime in that

parameter, in fact all we need to do is find paths in a graph of size $|\mathcal{D}(v^{\mathsf{B}})|$. This dramatic complexity reduction is obtained at the price of disallowing BtoR dependencies.

Heuristic Function

Assume an input FDR planning task Π. As indicated, we will choose a painting (a subset V^{R} of red variables) so that BtoR dependencies do not exist, and for each search state s generate a heuristic value by running NoBtoR-Planning with s as the initial state. We describe our painting strategy in the next section. Some words are in order regarding the heuristic function itself, which diverges from our previous theoretical discussion – Figure 3 and Theorem 2 – in several important aspects.

While the previous section assumed that the entire black state space is pre-composed into a single black variable v^{B}, that assumption was only made for convenience. In practice there is no need for such pre-composition. We instead run NoBtoR-Planning with the BlackReachable(R, d, d') calls implemented as a forward state space search within the projection onto the black variables, using only those black-affecting actions whose red preconditions are contained in the current set of red facts R. This is straightforward, and avoids having to generate the entire black state space up front – instead, we will only generate those parts actually needed during red-black plan generation as requested by the surrounding search. Still, of course for this to be feasible we need to keep the size of the black state space at bay.

That said, actually what we need to keep at bay is not the black state space itself, but its *weakly connected components*. As the red variables are taken out of this part of the problem, chances are that the remaining part will contain separate components.

Example 3 *In our running example, say there are several different kinds of products, i. e. the truck needs to buy a goal amount of several products. (This is indeed the case in the TPP benchmark suite as used in the IPC.) The state variables for each product then form an SCC like the variables B, P_1, P_7 in Figure 1 (b), mutually separated from each other by taking out (painting red) the central variable T.*

We can *decompose* the black state space, handling each connected component of variables $V_c^{\mathsf{B}} \subseteq V^{\mathsf{B}}$ separately. When calling BlackReachable(R, d, d'), we do not call a single state space search within the projection onto V^{B}, but call one state space search within the projection onto V_c^{B}, for every component V_c^{B}. The overall search is successful if all its components are, and in that case the overall solution path results from simple concatenation.

We finally employ several simple optimizations: *black state space results caching*, *stop search*, and *optimized red-black plan extraction*. The first of these is important as the heuristic function will be called on the same black state space many times during search, and within each call there may be several questions about paths from d to d' through that state space. The same pairs d and d' may reappear many times in the calls to BlackReachable(R, d, d'), so we can avoid duplicate effort simply by caching these results. Precisely, our cache consists of pairs (d, d') along with a black path $\pi(d, d')$ from d to d'. (In preliminary experiments, caching the actual triples (R, d, d') led to high memory consumption.) Whenever a call to BlackReachable(R, d, d') is made, we check whether (d, d') is in the cache, and if so check whether $\pi(d, d')$ works given R, i. e., contains only actions whose red preconditions are contained in R. If that is not so, or if (d, d') is not in the cache at all yet, we run the (decomposed) state space search, and in case of success add its result to the cache.

Stop search is the same as already used in (and found to be important in) Katz et al.'s previous work on red-black plan heuristics. If the red-black plan π^{RB} generated for a search state s is actually executable in the original FDR input planning task, then we terminate search immediately and output the path to s, followed by π^{RB}, as the solution.

Finally, the red-black plans π^{RB} described in the proof of Theorem 2 are of course highly redundant in that they execute the entire red fixed points, as opposed to establishing only those red facts $R^g \subseteq R$ required by the red goal $G[V^{\mathsf{R}}]$, and required as red preconditions on the solution black path found by BlackReachable(R, d, d'). We address this straightforwardly following the usual relaxed planning approach. The forward red fixed point phase is followed by a backward red plan extraction phase, in which we select supporters for R^g and the red subgoals it generates.

Painting Strategy

Given an input FDR planning task Π, we need to choose our painting V^{R} such that the red-black causal graph CG_Π^{RB} has no BtoR dependencies. A convenient view for doing so is to perceive the causal graph CG_Π as a DAG D of SCCs in which the root SCCs are at the top and the leaf SCCs at the bottom: Our task is then equivalent to drawing a "horizontal line" anywhere through D, painting the top part red, and painting the bottom part black. We say that such a painting is *non-trivial* if the bottom part is non-empty.

Example 4 *In our running example, the only non-trivial painting is the one illustrated in Figure 4.*

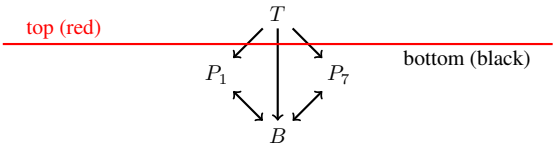


Figure 4: The painting in our running example.

If there are several different kinds of products as described in Example 3, then the state variables for each product form a separate component in the bottom part. If there are several trucks, then the "horizontal line" may put any non-empty subset of trucks into the top part.

We implemented a simple painting strategy accommodating the above. The strategy has an input parameter N imposing an upper bound on the (conservatively) estimated size of the decomposed black state space. Starting with the DAG D

		without preferred operators								with preferred operators							
				$N =$								$N =$					
domain	#	h^{Mercury}	h^{FF}	0	$1k$	$10k$	$100k$	$1m$	$10m$	h^{Mercury}	h^{FF}	0	$1k$	$10k$	$100k$	$1m$	$10m$
Logistics00	28	28	28	28	28	28	28	28	28	28	28	28	28	28	28	28	28
Logistics98	35	35	26	23	23	23	23	23	23	35	35	32	32	32	32	32	32
Miconic	150	150	150	150	150	150	150	150	150	150	150	150	150	150	150	150	150
ParcPrinter08	30	30	26	30	30	30	30	30	30	30	26	30	30	30	30	30	30
ParcPrinter11	20	20	12	20	20	20	20	20	20	20	12	20	20	20	20	20	20
Pathways	30	11	11	8	10	10	10	10	10	30	20	23	23	23	23	23	23
Rovers	40	27	23	23	23	24	26	25	24	40	40	40	40	40	40	40	40
Satellite	36	36	30	26	26	26	26	25	26	36	36	35	35	35	35	35	35
TPP	30	23	22	18	18	18	18	18	19	30	30	30	30	30	30	30	30
Woodworking08	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30
Woodworking11	20	20	19	19	19	20	20	20	20	20	20	20	20	20	20	20	20
Zenotravel	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
$\sum$	469	430	397	395	397	399	401	399	400	469	447	458	458	458	458	458	458

Table 1: Coverage results. All heuristics are run with FD's greedy best-first search, single-queue for configurations without preferred operators, double-queue for configurations with preferred operators. The preferred operators are taken from h^{FF} in all cases (see text).

of SCCs over the original causal graph, and with the empty set V^{B} of black variables, iterate the following steps:

1. Set the candidates for inclusion to be all leaf SCCs $V_l \subseteq V$ in D.
2. Select a V_l where $\prod_{v \in V'} |\mathcal{D}(v)|$ is minimal.
3. Set $V' := V^{\mathsf{B}} \cup V_l$ and find the weakly connected components $V_c^{\mathsf{B}} \subseteq V'$.
4. If $\sum_{V_c^{\mathsf{B}}} \prod_{v \in V_c^{\mathsf{B}}} |\mathcal{D}(v)| \leq N$, set $V^{\mathsf{B}} := V'$, remove V_l from D, and iterate; else, terminate.

Example 5 *In our running example, this strategy will result in exactly the painting displayed in Figure 4, provided N is choosen large enough to accommodate the variable subset $\{B, P_1, P_7\}$, but not large enough to accommodate the entire set of variables.*

If there are several different kinds of products, as in the IPC TPP domain, then N does not have to be large to accommodate all products (as each is a separate component), but would have to be huge to accommodate any truck (which would reconnect all these components). Hence, for a broad range of settings of N, we end up painting the products black and the trucks red, as desired.

Note that our painting strategy may terminate with the trivial painting ($V^{\mathsf{B}} = \emptyset$), namely if even the smallest candidate V_l breaks the size bound N. This will happen, in particular, on all input tasks Π whose causal graph is a single SCC, unless N is large enough to accommodate the entire state space. Therefore, in practice, we exclude input tasks whose causal graph is strongly connected.

Experiments

Our techniques are implemented in Fast Downward (FD) (Helmert 2006). For our painting strategy, we experiment with the size bounds $N \in \{1k, 10k, 100k, 1m, 10m\}$ ("m" meaning "million"). We run all IPC STRIPS benchmarks, precisely their satisficing-planning test suites, where we obtain non-trivial paintings. This excludes domains whose causal graphs are strongly connected, and it excludes domains where even the smallest leaf SCCs break our size bounds. It turns out that, given this, only 9 benchmark domains qualify, 3 of which have been used in two IPC editions so that we end up with 12 test suites.

As our contribution consists in a new heuristic function, we fix the search algorithm, namely FD's greedy best-first search with lazy evaluation, and evaluate the heuristic function against its closest relatives. Foremost, we compare to the standard relaxed plan heuristic h^{FF}, which we set out to improve upon. More specifically, we compare to two implementations of h^{FF}: the one from the FD distribution, and our own heuristic with size bound $N = 0$. The former is more "standard", but differs from our heuristic even in the case $N = 0$ because these are separate implementations that do not coincide exactly in terms of tie breaking. As we shall see, this seemingly small difference can significantly affect performance. To obtain a more precise picture of which differences are due to the black variables rather than other details, we use $N = 0$, i. e. "our own" h^{FF} implementation, as the main baseline. We also run h^{Mercury}, as a representative of the state of the art in alternate red-black plan heuristics.

A few words are in order regarding preferred operators. As was previously observed by Katz et al., h^{Mercury} yields best performance when using the standard preferred operators extracted from h^{FF}: The latter is computed as part of computing h^{Mercury} anyhow, and the standard preferred operators tend to work better than variants Katz et al. tried trying to exploit the handling of black variables in h^{Mercury}. For our own heuristics, we made a similar observation, in that we experimented with variants of preferred operators specific to these, but found that using the standard preferred operators from h^{FF} gave better results. This is true despite the fact that our heuristics do *not* compute h^{FF} as part of the process. The preferred operators are obtained by a separate call to FD's standard implementation of h^{FF}, on every search state. Hence, in what follows, all heuristics reported use the exact

same method to generate preferred operators.

Table 1 shows coverage results. Observe first that, in terms of this most basic performance parameter, h^{Mercury} dominates all other heuristics, across all domains and regardless whether or not preferred operators are being used. Recall here that, in contrast to our heuristics which paint black the variables "close to the causal graph leaves", h^{Mercury} uses paintings that paint black the variables "close to the causal graph roots". Although in principle the former kind of painting can be of advantage as illustrated in Example 1, as previously indicated the latter kind of painting tends to work better on the IPC benchmarks. We provide a per-instance comparison of h^{Mercury} against our heuristics, in Rovers and TPP which turn out to be the most interesting domains for these heuristics, further below (Table 3). For now, let's focus on the comparison to the baseline, h^{FF}.

Note first the influence of tie breaking: Without preferred operators, $N = 0$ has a dramatic advantage over h^{FF} in ParcPrinter, and smaller but significant disadvantages in Logistics98, Pathways, Satellite, and TPP. With preferred operators, the coverage differences get smoothed out, because with the pruning the instances become much easier to solve so the performance differences due to the different heuristics do not affect coverage as much anymore. The upshot is that only the advantage in ParcPrinter, but none of the disadvantages, remain. As these differences have nothing to do with our contribution, we will from now on not discuss h^{FF} as implemented in FD, and instead use the baseline $N = 0$.

Considering coverage as a function of N, observe that, with preferred operators, there are no changes whatsoever, again because with the pruning the instances become much easier to solve. Without preferred operators, increasing N and thus the black part of our heuristic function affects coverage in Pathways, Rovers, Satellite, TPP, and Woodworking11. With the single exception of Satellite for $N = 1m$, the coverage change relative to the baseline $N = 0$ is positive. However, the extent of the coverage increase is small in almost all cases. We now examine this more closely, considering more fine-grained performance parameters.

Table 2 considers the number of evaluated states during search, and search runtime, in terms of improvement factors i. e. the factor by which evaluations/search time reduce relative to the baseline $N = 0$. As we can see in the top half of the table, the (geo)mean improvement factors are almost consistently greater than 1 (the most notable exception being Pathways), i. e., there typically is an improvement on average (although: see below). The actual search time, on the other hand, almost consistently gets worse, with a very pronounced tendency for the "improvement factor" to be < 1, and to decrease as a function of N. The exceptions in this regard are Rovers, and especially TPP where, quite contrary to the common trend, the search time improvement factor *grows* as a function of N. This makes sense as Rovers and TPP clearly stand out as the two domains with the highest evaluations improvement factors.

Per-instance data sheds additional light on this. In Logistics, Miconic, ParcPrinter, Pathways, and Zenotravel, almost all search space reductions obtained are on the smallest instances, where N is large enough to accommodate the entire state space and hence, trivially, the number of evaluations is 1. On almost all larger instances of these domains, the search spaces are identical, explaining the bad search time results previously observed. In Satellite and Woodworking, the results are mixed. There are substantial improvements also on some large instances, but the evaluations improvement factor is always smaller than 6, with the single exception of Woodworking08 instance p24 where for $N \geq 10k$ it is 17.23. In contrast, in Rovers the largest evaluations improvement factor is 4612, and in TPP it is 17317.

domain	#	$1k$	$10k$	$100k$	$1m$	$10m$
evaluations						
Logistics00	28	1.00	1.05	1.43	3.71	3.71
Logistics98	23	1.00	0.98	1.01	1.22	1.35
Miconic	150	1.24	1.41	1.86	2.18	3.12
ParcPrinter08	30	1.07	1.38	1.52	1.52	1.71
ParcPrinter11	20	1.00	1.03	1.08	1.08	1.09
Pathways	8	0.71	0.71	0.71	0.88	0.88
Rovers	19	1.60	1.95	5.10	5.84	5.16
Satellite	25	1.04	1.83	1.61	2.18	2.40
TPP	17	1.83	3.76	5.90	20.89	27.54
Woodworking08	30	1.54	2.06	2.06	2.06	2.06
Woodworking11	19	1.08	1.68	1.76	1.76	1.76
Zenotravel	20	1.14	1.14	0.87	1.13	2.27
search time						
Logistics00	28	1.00	1.00	0.94	0.59	0.59
Logistics98	23	1.00	1.00	0.93	0.80	0.58
Miconic	150	0.73	0.73	0.70	0.63	0.44
ParcPrinter08	30	1.00	1.00	1.00	1.00	0.44
ParcPrinter11	20	1.00	0.95	0.96	0.96	0.23
Pathways	8	0.97	0.97	0.96	0.96	0.93
Rovers	19	1.44	1.74	2.01	1.56	0.87
Satellite	25	0.87	1.07	0.90	0.63	0.24
TPP	17	0.98	1.39	1.78	3.75	4.67
Woodworking08	30	0.94	0.86	0.86	0.86	0.86
Woodworking11	19	0.87	0.65	0.67	0.67	0.67
Zenotravel	20	1.00	1.00	0.95	0.78	0.43

Table 2: Improvement factors relative to $N = 0$. Per-domain geometric mean over the set of instances commonly solved for all values of N. Without preferred operators.

Table 3 shows per-instance data on Rovers and TPP, where our techniques are most interesting. We also include h^{Mercury} here for a detailed comparison. $N = 1k$ and $N = 100k$ are left out of the table for lack of space, and as these configurations are always dominated by at least one other value of N here. With respect to the behavior against the baseline $N = 0$, clearly in both domains drastic evaluations and search time improvements can be obtained. It should be said though that there is an unfortunate tendency for our red-black heuristics to have advantages in the smaller instances, rather than the larger ones. This is presumably because, in smaller instances (even disregarding the pathological case where the entire state space fits into the black part of our heuristic) we have a better chance to capture complex variable interactions inside the black part, and hence obtain substantially better heuristic values.

With respect to h^{Mercury}, the conclusion can only be that the previous approach to red-black plan heuristics – painting

	h^{Mercury}		$N=0$		$N=10k$		$N=1m$		$N=10m$	
	E	T	E	T	E	T	E	T	E	T
Rovers										
p01	5	0.1	35	0.1	31	0.1	1	0.1	1	0.1
p02	6	0.1	6	0.1	1	0.1	1	0.1	1	0.1
p03	1	0.1	62	0.1	112	0.1	1	0.1	1	0.1
p04	1	0.1	17	0.1	21	0.1	1	0.1	1	0.1
p05	119	0.1	114	0.1	170	0.1	117	0.1	117	0.1
p06	304	0.1	543	0.1	485	0.1	485	0.1	285	0.6
p07	70	0.1	331	0.1	334	0.1	162	1.5	175	7.1
p08	116	0.1	1742189	46.3	451078	15.4	603	0.1	1929	28.2
p09	358	0.1	2773	0.1	1792	0.1	2120	0.1	2120	0.1
p10	578	0.1	441	0.1	441	0.1	244	0.4	244	0.4
p11	1047	0.1	85832	2.7	85787	3.1	85787	3.1	85787	3.1
p12	6	0.1	606	0.1	958	0.1	301	0.3	698	0.4
p13	25037	2.13	1944	0.1	2578	0.1	2882	0.3	2882	0.3
p14	294	0.1	5161720	208.5	1467	0.1	732	0.2	1119	0.8
p15	1035	0.1	–	–	5024	0.3	3520	1.5	4782	3.3
p16	358	0.1	–	–	11895	0.6	11895	0.6	11895	0.6
p17	1139	0.1	–	–	–	–	3340	0.3	3340	0.3
p18	2156	0.19	93372	6.6	47472	3.6	47472	3.6	48620	6.0
p19	180979	22.08	370650	38.3	452905	47.7	470758	47.7	470758	47.3
p20	–	–	1782100	233.1	–	–	1828671	248.3	1828671	248.0
p22	3674	0.63	2478919	339.2	1707251	237.1	1707251	232.4	1707251	236.0
p23	24347	5.77	–	–	–	–	–	–	–	–
p25	1	0.1	2677	0.3	31890	4.5	31248	4.2	44688	5.8
p26	7338129	1418.16	117583	13.7	83724	11.7	7263721	974.8	–	–
p27	61575	13.73	–	–	6737329	1158.0	6737329	1138.6	6737329	1148.3
p28	6346	1.95	1066434	223.8	15321	3.8	18002	4.3	18002	4.3
p29	8409	2.71	1298473	251.4	–	–	–	–	–	–
p30	–	–	5940371	1134.0	333303	108.0	–	–	–	–
p34	60144	38.59	–	–	–	–	–	–	–	–
TPP										
p01	1	0.1	5	0.1	1	0.1	1	0.1	1	0.1
p02	1	0.1	9	0.1	1	0.1	1	0.1	1	0.1
p03	1	0.1	13	0.1	1	0.1	1	0.1	1	0.1
p04	1	0.1	17	0.1	17	0.1	1	0.1	1	0.1
p05	1	0.1	22	0.1	25	0.1	25	0.1	25	0.1
p06	38	0.1	107	0.1	46	0.1	46	0.1	46	0.1
p07	1672	0.1	1756	0.1	68	0.1	68	0.1	68	0.1
p08	2462	0.1	2534	0.1	71	0.1	71	0.1	71	0.1
p09	6753	0.28	2963	0.1	299	0.1	121	0.1	121	0.1
p10	24370	1.24	10712	0.5	1061	0.1	147	0.1	147	0.1
p11	15519	1.3	1610504	99.8	56090	4.0	93	0.1	93	0.1
p12	54852	4.37	1340734	91.3	699377	55.7	109	0.1	109	0.1
p13	38205	3.5	40291	3.5	40291	3.5	472	0.1	475	0.1
p14	57981	7.37	35089	3.6	35089	3.6	552	0.1	555	0.1
p15	52722	7.32	22842	2.4	22842	2.5	70467	8.1	283	0.1
p16	298618	77.47	247304	49.6	247304	48.4	112610	19.6	253727	38.0
p17	2660716	774.05	–	–	–	–	–	–	1611051	671.4
p18	264855	77.25	–	–	–	–	–	–	–	–
p19	1957381	639.02	1710323	509.3	1710323	508.8	1710323	505.6	–	–
p20	–	–	–	–	–	–	–	–	4161990	1272.4
p21	811226	578.23	–	–	–	–	–	–	–	–
p22	652741	372.74	–	–	–	–	–	–	–	–
p23	1329626	902.09	2432228	1362.0	2432228	1365.3	2432228	1367.6	2432228	1370.7
p24	1253699	801.71	–	–	–	–	–	–	–	–

Table 3: Evaluations and search time in Rovers and TPP. "E" evaluations, "T" search time. Without preferred operators.

variables "close to the root" black, as opposed to painting variables "close to the leaves" black as we do here – works better in practice. There are rare cases where our new heuristics have an advantage, most notably in Rovers p20, p26, p30, and TPP p5–p17, p19, p20. But overall, especially on the largest instances, h^{Mercury} tends to be better. We remark that, with preferred operators switched on, the advantage of h^{Mercury} tends to be even more pronounced because the few cases that are hard for it in Table 3 become easy.

A few words are in order regarding plan quality, by which, since we only consider uniform action costs in the experiments, we mean plan length. Comparing our most informed configuration, $N = 10m$, to our pure delete relaxed baseline, i. e. our heuristic with $N = 0$, it turns out that the value of N hardly influences the quality of the plans found. Without using preferred operators, the average per-domain gain/loss of one configuration over the other is always $< 3\%$. The only domain where solution quality differs more significantly is TPP, where the generated plans for $N = 10m$ are 23.3% shorter on average than those with $N = 0$. This reduces to 10% when preferred operators are switched on. In the other domains, not much changes when enabling preferred operators; the average gain/loss per domain is less than 4.4%.

Comparing our $N = 10m$ configuration to h^{Mercury}, having preferred operators disabled, the plan quality is only slightly different in most domains ($< 3.1\%$ gain/loss on average). Results differ more significantly in Miconic and TPP. In the former, our plans are 25% longer than those found using h^{Mercury}; in the latter, our plans are 25% shorter. Enabling preferred operators does not change much, except in Woodworking, where our plans are on average 19.1% (16.5%) shorter in the IPC'08 (IPC'11) instance suites.

Conclusion

Our investigation has brought new insights into the interaction between red and black variables in red-black planning. The practical heuristic function resulting from this can, in principle, improve over standard relaxed plan heuristics as well as known red-black plan heuristics. In practice – as far as captured by IPC benchmarks – unfortunately such improvements are rare. We believe this is a valuable insight for further research on red-black planning. It remains to be seen whether our tractability analysis can be extended and/or exploited in some other, more practically fruitful, way. The most promising option seems to be to seek tractable special cases of black-to-red (BtoR) dependencies, potentially by restrictions onto the DTG (the variable-value transitions) of the black variable weaker than the "invertibility" criterion imposed by Katz et al.

Acknowledgments. We thank Carmel Domshlak for discussions. We thank the anonymous reviewers, whose comments helped to improve the paper. This work was partially supported by the German Research Foundation (DFG), under grant HO 2169/5-1, and by the EU FP7 Programme under grant agreement 295261 (MEALS).

References

Baier, J. A., and Botea, A. 2009. Improving planning performance using low-conflict relaxed plans. In Gerevini, A.; Howe, A.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*, 10–17. AAAI Press.

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence* 129(1–2):5–33.

Bonet, B.; McCluskey, L.; Silva, J. R.; and Williams, B., eds. 2012. *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*. AAAI Press.

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

Coles, A.; Fox, M.; Long, D.; and Smith, A. 2008. A hybrid relaxed planning graph'lp heuristic for numeric planning domains. In Rintanen et al. (2008), 52–59.

Coles, A. J.; Coles, A.; Fox, M.; and Long, D. 2013. A hybrid LP-RPG heuristic for modelling numeric resource flows in planning. *Journal of Artificial Intelligence Research* 46:343–412.

Domshlak, C.; Hoffmann, J.; and Katz, M. 2015. Red-black planning: A new systematic approach to partial delete relaxation. *Artificial Intelligence* 221:73–114.

Fox, M., and Long, D. 2001. Hybrid STAN: Identifying and managing combinatorial optimisation sub-problems in planning. In Nebel, B., ed., *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI-01)*, 445–450. Seattle, Washington, USA: Morgan Kaufmann.

Gerevini, A.; Saetti, A.; and Serina, I. 2003. Planning through stochastic local search and temporal action graphs. *Journal of Artificial Intelligence Research* 20:239–290.

Helmert, M., and Geffner, H. 2008. Unifying the causal graph and additive heuristics. In Rintanen et al. (2008), 140–147.

Helmert, M. 2004. A planning heuristic based on causal graph analysis. In Koenig, S.; Zilberstein, S.; and Koehler, J., eds., *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS'04)*, 161–170. Whistler, Canada: Morgan Kaufmann.

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

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

Hoffmann, J. 2005. Where 'ignoring delete lists' works: Local search topology in planning benchmarks. *Journal of Artificial Intelligence Research* 24:685–758.

Katz, M., and Hoffmann, J. 2013. Red-black relaxed plan heuristics reloaded. In Helmert, M., and Röger, G., eds., *Proceedings of the 6th Annual Symposium on Combinatorial Search (SOCS'13)*, 105–113. AAAI Press.

Katz, M., and Hoffmann, J. 2014. Mercury planner: Pushing the limits of partial delete relaxation. In *IPC 2014 planner abstracts*, 43–47.

Katz, M.; Hoffmann, J.; and Domshlak, C. 2013a. Red-black relaxed plan heuristics. In desJardins, M., and Littman, M., eds., *Proceedings of the 27th AAAI Conference on Artificial Intelligence (AAAI'13)*, 489–495. Bellevue, WA, USA: AAAI Press.

Katz, M.; Hoffmann, J.; and Domshlak, C. 2013b. Who said we need to relax *all* variables? In Borrajo, D.; Fratini, S.; Kambhampati, S.; and Oddi, A., eds., *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS'13)*, 126–134. Rome, Italy: AAAI Press.

Keyder, E., and Geffner, H. 2008. Heuristics for planning with action costs revisited. In Ghallab, M., ed., *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI-08)*, 588–592. Patras, Greece: Wiley.

Keyder, E.; Hoffmann, J.; and Haslum, P. 2012. Semi-relaxed plan heuristics. In Bonet et al. (2012), 128–136.

Keyder, E.; Hoffmann, J.; and Haslum, P. 2014. Improving delete relaxation heuristics through explicitly represented conjunctions. *Journal of Artificial Intelligence Research* 50:487–533.

McDermott, D. V. 1999. Using regression-match graphs to control search in planning. *Artificial Intelligence* 109(1-2):111–159.

Nakhost, H.; Hoffmann, J.; and Müller, M. 2012. Resource-constrained planning: A monte carlo random walk approach. In Bonet et al. (2012), 181–189.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research* 39:127–177.

Rintanen, J.; Nebel, B.; Beck, J. C.; and Hansen, E., eds. 2008. *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS'08)*. AAAI Press.

Seipp, J., and Helmert, M. 2011. Fluent merging for classical planning problems. In *ICAPS 2011 Workshop on Knowledge Engineering for Planning and Scheduling*, 47–53.

van den Briel, M.; Benton, J.; Kambhampati, S.; and Vossen, T. 2007. An LP-based heuristic for optimal planning. In Bessiere, C., ed., *Proceedings of the Thirteenth International Conference on Principles and Practice of Constraint Programming (CP'07)*, volume 4741 of *Lecture Notes in Computer Science*, 651–665. Springer-Verlag.

From Fork Decoupling to Star-Topology Decoupling

Daniel Gnad and **Jörg Hoffmann**
Saarland University
Saarbrücken, Germany
{gnad, hoffmann}@cs.uni-saarland.de

Carmel Domshlak
Technion
Haifa, Israel
dcarmel@ie.technion.ac.il

Abstract

Fork decoupling is a recent approach to exploiting problem structure in state space search. The problem is assumed to take the form of a fork, where a single (large) *center* component provides preconditions for several (small) *leaf* components. The leaves are then conditionally independent in the sense that, given a fixed center path π^C, the *compliant* leaf moves – those leaf moves enabled by the preconditions supplied along π^C – can be scheduled independently for each leaf. *Fork-decoupled state space search* exploits this through conducting a regular search over center paths, augmented with maintenance of the compliant paths for each leaf individually. We herein show that the same ideas apply to much more general *star-topology* structures, where leaves may supply preconditions for the center, and actions may affect several leaves simultaneously as long as they also affect the center. Our empirical evaluation in planning, super-imposing star topologies by automatically grouping the state variables into suitable components, shows the merits of the approach.

Introduction

In classical AI planning, large deterministic transition systems are described in terms of a set of finite-domain state variables, and actions specified via preconditions and effects over these state variables. The task is to find a sequence of actions leading from a given initial state to a state that satisfies a given goal condition. *Factored planning* is one traditional approach towards doing so effectively. The idea is to decouple the planning task into subsets, *factors*, of state variables. In *localized* factored planning (Amir and Engelhardt 2003; Brafman and Domshlak 2006; 2008; 2013; Fabre et al. 2010), two factors interact if they are affected by common actions, and a global plan needs to comply with these *cross-factor interactions*. In *hierarchical* factored planning (e. g. (Knoblock 1994; Kelareva et al. 2007; Wang and Williams 2015)), the factors are used within a hierarchy of increasingly more detailed abstraction levels, search refining abstract plans as it proceeds down the hierarchy, and backtracking if an abstract plan has no refinement.

Both localized and hierarchical factored planning have traditionally been viewed as the resolution of complex interactions between (relatively) small factors. In recent work, Gnad and Hoffmann (2015) (henceforth: GH) proposed to turn this upside down, fixing a *simple* interaction profile the factoring should induce, at the cost of potentially very *large* factors. They baptize this approach *target-profile factoring*, and develop a concrete instance where the target profile is a *fork*: a single (large) *center* factor provides preconditions for several (small) *leaf* factors, and no other cross-factor interactions are present. They introduce a simple and quick *factoring strategy* which, given an arbitrary input planning task, analyzes the state variable dependencies and either outputs a *fork factoring*, or *abstains* if the relevant stucture is not there (the task can then be handed over to other methods).

Say the factoring strategy does not abstain. We then face, not a general planning problem, but a fork planning problem. GH's key observation is that these can be solved via *fork-decoupled state space search*: The leaves are conditionally independent in the sense that, given a fixed center path π^C, the *compliant* leaf moves – those leaf moves enabled by the preconditions supplied along π^C – can be scheduled independently for each leaf. This can be exploited by searching only over center paths, and maintaining the compliant paths separately for each leaf, thus avoiding the enumeration of state combinations across leaves. GH show that this substantially reduces the number of reachable states in (non-abstained) planning benchmarks, almost always by at least $1-2$ orders of magnitude, up to 6 orders of magnitude in one domain (TPP). They show how to connect to classical planning heuristics, and to standard search methods, guaranteeing plan optimality under the same conditions as before.

We herein extend GH's ideas to *star-topology* structures, where the center still supplies preconditions for the leaves, but also the leaves may supply preconditions for the center, and actions may affect several leaves simultaneously as long as they also affect the center. The connection to standard heuristics and search methods remains valid. We run experiments on the planning competition benchmarks.

We place our work in the planning context for the direct connection to GH. However, note that trying to factorize a general input problem, and having to abstain in case the saught structure is not present, really is an artefact of the general-planning context. Star-topology decoupling applies, in principle, to the state space of any system that naturally has a star topology. As star topology is a classical design paradigm in many areas of CS, such systems abound.

Preliminaries

We use a finite-domain state variable formalization of planning (e. g. (Bäckström and Nebel 1995; Helmert 2006)). A *finite-domain representation* planning task, short FDR task, is a quadruple $\Pi = \langle V, A, I, G\rangle$. V is a set of *state variables*, where each $v \in V$ is associated with a finite domain $\mathcal{D}(v)$. We identify (partial) variable assignments with sets of variable/value pairs. A complete assignment to V is a *state*. I is the *initial state*, and the *goal* G is a partial assignment to V. A is a finite set of *actions*. Each action $a \in A$ is a triple $\langle \mathsf{pre}(a), \mathsf{eff}(a), \mathsf{cost}(a)\rangle$ where the *precondition* $\mathsf{pre}(a)$ and *effect* $\mathsf{eff}(a)$ are partial assignments to V, and $\mathsf{cost}(a) \in \mathbb{R}^{0+}$ is the action's non-negative *cost*.

For a partial assignment p, $\mathcal{V}(p) \subseteq V$ denotes the subset of state variables instantiated by p. For any $V' \subseteq \mathcal{V}(p)$, by $p[V']$ we denote the assignment to V' made by p. An action a is *applicable* in a state s if $\mathsf{pre}(a) \subseteq s$, i. e., if $s[v] = \mathsf{pre}(a)[v]$ for all $v \in \mathcal{V}(\mathsf{pre}(a))$. Applying a in s changes the value of each $v \in \mathcal{V}(\mathsf{eff}(a))$ to $\mathsf{eff}(a)[v]$, and leaves s unchanged elsewhere; the outcome state is denoted $s[\![a]\!]$. We also use this notation for partial states p: by $p[\![a]\!]$ we denote the assignment over-writing p with $\mathsf{eff}(a)$ where both p and $\mathsf{eff}(a)$ are defined. The outcome state of applying a sequence of (respectively applicable) actions is denoted $s[\![\langle a_1, \ldots, a_n\rangle]\!]$. A *plan* for Π is an action sequence s.t. $G \subseteq I[\![\langle a_1, \ldots, a_n\rangle]\!]$. The plan is *optimal* if its summed-up cost is minimal among all plans for Π.

The *causal graph* of a planning task captures state variable dependencies (e. g. (Knoblock 1994; Jonsson and Bäckström 1995; Brafman and Domshlak 2003; Helmert 2006)). We use the commonly employed definition in the FDR context, where the causal graph *CG* is a directed graph over vertices V, with an arc from v to v', which we denote $(v \to v')$, if $v \neq v'$ and there exists an action $a \in A$ such that $(v, v') \in [\mathcal{V}(\mathsf{eff}(a)) \cup \mathcal{V}(\mathsf{pre}(a))] \times \mathcal{V}(\mathsf{eff}(a))$. In words, the causal graph captures precondition-effect as well as effect-effect dependencies, as result from the action descriptions. A simple intuition is that, whenever $(v \to v')$ is an arc in *CG*, changing the value of v' may involve changing that of v as well. We assume for simplicity that *CG* is weakly connected (this is wlog: else, the task can be equivalently split into several independent tasks).

We will also need the notion of a *support graph*, *SuppG*, similarly as used e. g. by Hoffmann (2011). *SuppG* is like *CG* except its arcs are only those $(v \to v')$ where there exists an action $a \in A$ such that $(v, v') \in \mathcal{V}(\mathsf{pre}(a)) \times \mathcal{V}(\mathsf{eff}(a))$. In words, the support graph captures only the precondition-effect dependencies, not effect-effect dependencies. This more restricted concept will be needed to conveniently describe our notion of star topologies, for which purpose the effect-effect arcs in *CG* are not suitable.

As an illustrative example, we will consider a simple transportation-like FDR planning task $\Pi = \langle V, A, I, G\rangle$ with one package p and two trucks t_A, t_B, defined as follows. $V = \{p, t_A, t_B\}$ where $\mathcal{D}(p) = \{A, B, l_1, l_2, l_3\}$ and $\mathcal{D}(t_A) = \mathcal{D}(t_B) = \{l_1, l_2, l_3\}$. The initial state is $I = \{p = l_1, t_A = l_1, t_B = l_3\}$, i. e., p and t_A start at l_1, and t_B starts at l_3. The goal is $G = \{p = l_3\}$. The actions (all with cost 1) are truck moves and load/unload:

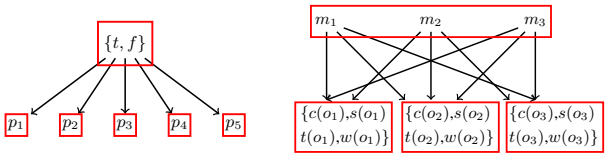


Figure 1: (Gnad and Hoffmann 2015) Possible fork factorings in transportation with fuel consumption (left), and job-planning problems (right).

- move(x, y, z): precondition $\{t_x = y\}$ and effect $\{t_x = z\}$, where $x \in \{A, B\}$ and $\{y, z\} \in \{\{l_1, l_2\}, \{l_2, l_3\}\}$.
- load(x, y): precondition $\{t_x = y, p = y\}$ and effect $\{p = x\}$, where $x \in \{A, B\}$ and $y \in \{l_1, l_2, l_3\}$.
- unload(x, y): precondition $\{t_x = y, p = x\}$ and effect $\{p = y\}$, where $x \in \{A, B\}$ and $y \in \{l_1, l_2, l_3\}$.

The causal graph and support graph of this task are identical. Their arcs are $(t_A \to p)$ and $(t_B \to p)$.

Fork Decoupling

We give a brief summary of fork decoupling, in a form suitable for describing our extension to star topologies.

Definition 1 (Fork Factoring (GH)) *Let Π be an FDR task with variables V. A* factoring *$\mathcal{F}$ is a partition of V into non-empty subsets F, called* factors. *The* interaction graph *IG($\mathcal{F}$) of $\mathcal{F}$ is the directed graph whose vertices are the factors, with an arc $(F \to F')$ if $F \neq F'$ and there exist $v \in F$ and $v' \in F'$ such that $(v \to v')$ is an arc in CG.*

$\mathcal{F}$ is a fork factoring *if $|\mathcal{F}| > 1$ and there exists $F^C \in \mathcal{F}$ s.t. the arcs in IG($\mathcal{F}$) are exactly $\{(F^C \to F^L) \mid F^L \in \mathcal{F} \setminus \{F^C\}\}$. F^C is the* center *of $\mathcal{F}$, and all other factors $F^L \in \mathcal{F}^L := \mathcal{F} \setminus \{F^C\}$ are* leaves. *We also consider the* trivial factoring *where $F^C = V$ and $\mathcal{F}^L = \emptyset$, and the* pathological factoring *where $F^C = \emptyset$ and $\mathcal{F}^L = \{V\}$.*

The only cross-factor interactions in a fork factoring consist in the center factor establishing preconditions for actions moving individual leaf factors. We use the word "center", instead of GH's "root", to align the terminology with star topologies. Regarding the trivial and pathological cases, in the former decoupled search simplifies to standard search, and in the latter the entire problem is pushed into the single "leaf". Note that, when we say that $\mathcal{F}$ is a fork factoring, we explicitly exclude these cases.

In our example, we will consider the fork factoring where $F^C = \{t_A, t_B\}$ and the single leaf is $F^L = \{p\}$.

Given an arbitrary FDR task Π as input, as pointed out by GH, a fork factoring – if one exists, which is the case iff the causal graph has more than one strongly connected component (SCC) – can be found automatically based on a simple causal graph analysis. We describe GH's strategy later, in the experiments, along with our own generalized strategies. For illustration, Figure 1 already shows factorings that GH's strategy may find, on practical problems akin to planning benchmarks. On the left, a truck t with fuel supply f transports packages $p_1, \ldots, p_n$. On the right, objects o_i are independent except for sharing the machines.

We need some terminology, that we will use also for star topologies later on. Assume an FDR task $\Pi = \langle V, A, I, G\rangle$ and a fork factoring $\mathcal{F}$ with center F^C and leaves $F^L \in \mathcal{F}^L$.

We refer to the actions A^C affecting the center as *center actions*, notation convention a^C, and to all other actions as *leaf actions*, notation convention a^L. For the set of actions affecting one particular $F^L \in \mathcal{F}^L$, we write $A^L|_{F^L}$. As $\mathcal{F}$ is a fork factoring, the center actions A^C have preconditions and effects only on F^C. The leaf actions $A^L|_{F^L}$ have preconditions only on $F^C \cup F^L$, and effects only on F^L. The sets A^C and $A^L|_{F^L}$ form a partition of the original action set A.

A *center path* is a sequence of center actions applicable to I; a *leaf path* is a sequence of leaf actions applicable to I when ignoring preconditions on the center. Value assignments to F^C are *center states*, notated s^C, and value assignments to any $F^L \in \mathcal{F}^L$ are *leaf states*, notated s^L. For the leaf states of one particular $F^L \in \mathcal{F}^L$, we write $S^L|_{F^L}$, and for the set of all leaf states we write S^L. A center state s^C is a *goal center state* if $s^C \supseteq G[F^C]$, and a leaf state $s^L \in S^L|_{F^L}$ is a *goal leaf state* if $s^L \supseteq G[F^L]$.

The idea in fork-decoupling is to augment a regular search over center paths with maintenance of cheapest compliant leaf paths for each leaf. A leaf path $\pi^L = \langle a^L_1, \ldots, a^L_n \rangle$ complies with center path π^C if we can schedule the a^L_i at monotonically increasing points alongside π^C so that each a^L_i is enabled in the respective center state. Formally:

Definition 2 (Fork-Compliant Path (GH)) *Let Π be an FDR task, $\mathcal{F}$ a fork factoring with center F^C, and π^C a center path traversing center states $\langle s^C_0, \ldots, s^C_n \rangle$. For a leaf path $\pi^L = \langle a^L_1, \ldots, a^L_m \rangle$, an* embedding *into π^C is a monotonically increasing function $t : \{1, \ldots, m\} \mapsto \{0, \ldots, n\}$ so that, for every $i \in \{1, \ldots, m\}$, $\mathsf{pre}(a^L_i)[F^C] \subseteq s^C_{t(i)}$. We say that π^L* fork-complies *with π^C, also π^L is π^C-*fork-compliant*, if an embedding exists.*

Where the center path in question is clear from context, or when discussing compliant paths in general, we will omit "π^C" and simply talk about *compliant* leaf paths.

In our example, $\pi^L = \langle \mathrm{load}(A, l_1) \rangle$ complies with the empty center path, and $\pi^L = \langle \mathrm{load}(A, l_1), \mathrm{unload}(A, l_2) \rangle$ complies with the center path $\pi^C = \langle \mathrm{move}(A, l_1, l_2) \rangle$. But $\pi^L = \langle \mathrm{load}(A, l_1), \mathrm{unload}(A, l_3) \rangle$ does not comply with π^C as the required precondition $t_A = l_3$ is not established on π^C. And $\pi^L = \langle \mathrm{load}(A, l_1), \mathrm{unload}(A, l_2), \mathrm{load}(A, l_2), \mathrm{unload}(A, l_1) \rangle$ does not comply with π^C as the last precondition $t_A = l_1$ does not appear behind $t_A = l_2$ on π^C.

The notion of compliant paths is a reformulation of plans for the original input planning task Π, in the following sense. Say π is a plan for Π, and say π^C is the sub-sequence of center actions in π. Then π^C is a center path. For each leaf $F^L \in \mathcal{F}^L$, say π^L is the sub-sequence of $A^L|_{F^L}$ actions in π. Then π^L is a leaf path, and is π^C-fork-compliant because π schedules π^L along with π^C in a way so that its center preconditions are fulfilled. Vice versa, if a center path π^C reaches a goal center state, and can be augmented with π^C-fork-compliant leaf paths π^L reaching goal leaf states, then the embedding of the π^L into π^C yields a plan for Π. Hence *the plans for Π are in one-to-one correspondence with center paths augmented with compliant leaf paths.*

GH define the *fork-decoupled state space* Θ^ϕ, in which each *fork-decoupled state* s is a pair $\langle \mathsf{center}(s), \mathsf{prices}(s) \rangle$ of a center state $\mathsf{center}(s)$ along with a *pricing function* $\mathsf{prices}(s) : S^L \mapsto \mathbb{R}^{0+} \cup \{\infty\}$. The paths in Θ^ϕ correspond to center paths, i. e., the *fork-decoupled initial state* I^ϕ has $\mathsf{center}(I^\phi) = I[F^C]$, and the transitions over center states are exactly those induced by the center actions. The pricing functions are maintained so that, for every center path π^C ending in fork-decoupled state s, and for every leaf state s^L, $\mathsf{prices}(s)[s^L]$ equals the cost of a cheapest π^C-fork-compliant leaf path π^L ending in s^L. The *fork-decoupled goal states* are those s where $\mathsf{center}(s)$ is a goal center state, and, for every $F^L \in \mathcal{F}^L$, at least one goal leaf state $s^L \in S^L|_{F^L}$ has a finite price $\mathsf{prices}(s)[s^L] < \infty$. Once a fork-decoupled goal state s is reached, a plan for Π can be extracted by augmenting the center path π^C leading to s with cheapest π^C-fork-compliant *goal leaf paths*, i. e., leaf paths ending in goal leaf states. Observe that this plan is optimal subject to fixing π^C, i. e., the cheapest possible plan for Π when comitting to exactly the center moves π^C.

Say $\pi^C = \langle \mathrm{move}(A, l_1, l_2), \mathrm{move}(B, l_3, l_2), \mathrm{move}(B, l_2, l_3) \rangle$ in our example, traversing the fork-decoupled states s_0, s_1, s_2, s_3. Then $\mathsf{prices}(s_0)[p = l_1] = 0$, $\mathsf{prices}(s_0)[p = A] = 1$, $\mathsf{prices}(s_1)[p = l_2] = 2$, $\mathsf{prices}(s_2)[p = B] = 3$, and $\mathsf{prices}(s_3)[p = l_3] = 4$. To extract a plan for Π from the fork-decoupled goal state s_3, we trace back the compliant leaf path supporting $p = l_3$ and embed it into π^C. The resulting plan loads p onto t_A, moves t_A to l_2, unloads p, moves t_B to l_2, loads p onto t_2, moves t_B to l_3, and unloads p.

The core of GH's construction is the maintenance of pricing functions. For forks, this is simple enough to be described in a few lines within the definition of Θ^ϕ. For star topologies, we need to substantially extend this construction, so we hone in on it in more detail here. We reformulate it in terms of *compliant path graphs*, which capture all possible compliant graphs for a leaf F^L given a center path π^C:

Definition 3 (Fork-Compliant Path Graph) *Let Π be an FDR task, $\mathcal{F}$ a fork factoring with center F^C and leaves $\mathcal{F}^L$, and π^C a center path traversing center states $\langle s^C_0, \ldots, s^C_n \rangle$. The π^C-*fork-compliant path graph *for a leaf $F^L \in \mathcal{F}^L$, denoted $\mathit{CompG}^\phi(\pi^C, F^L)$, is the arc-labeled weighted directed graph whose vertices are the* time-stamped leaf states $\{s^L_t \mid s^L \in S^L|_{F^L}, 0 \leq t \leq n\}$*, and whose arcs are:*

- *(i)* $s^L_t \xrightarrow{a^L} s'^L_t$ *with weight $c(a^L)$ whenever $s^L, s'^L \in S^L|_{F^L}$ and $a^L \in A^L|_{F^L}$ such that $\mathsf{pre}(a^L)[F^C] \subseteq s^C_t$, $\mathsf{pre}(a^L)[F^L] \subseteq s^L$, and $s^L[\![a^L]\!] = s'^L$.*
- *(ii)* $s^L_t \xrightarrow{0} s^L_{t+1}$ *with weight 0 for all $s^L \in S^L|_{F^L}$ and $0 \leq t < n$.*

In words, the π^C-fork-compliant path graph includes a copy of the leaf states at every time step $0 \leq t \leq n$ along the center path π^C. Within each t, the graph includes all leaf-state transitions enabled in the respective center state. From each t to $t + 1$, the graph has a 0-cost transition for each leaf state. Consider again the example, and the center path $\pi^C = \langle \mathrm{move}(A, l_1, l_2) \rangle$. The π^C-fork-compliant path graph for the package is shown in Figure 2.[1]

[1]Note that $\mathit{CompG}^\phi(\pi^C, F^L)$ contains redundant parts, not reachable from the initial leaf state $I[F^L]$, i. e., $(p = l_1)_0$ in the

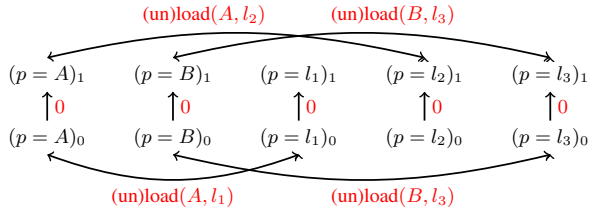


Figure 2: The fork-compliant path graph for $\pi^C = \langle \text{move}(A, l_1, l_2) \rangle$ in our illustrative example.

The π^C-fork-compliant leaf path $\pi^L = \langle \text{load}(A, l_1), \text{unload}(A, l_2) \rangle$ can be embedded by starting at $(p = l_1)_0$, following the arc labeled load(A, l_1) to $(p = A)_0$, following the 0-arc to $(p = A)_1$, and following the arc labeled unload(A, l_2) to $(p = l_2)_1$. The non-compliant leaf path $\pi^L = \langle \text{load}(A, l_1), \text{unload}(A, l_2), \text{load}(A, l_2), \text{load}(A, l_1) \rangle$ cannot be embedded, as (un)load(A, l_2) appears only at $t = 1$, while load(A, l_1) is not available anymore at $t \geq 1$.

Lemma 1 *Let Π be an FDR task, $\mathcal{F}$ a fork factoring with center F^C and leaves $\mathcal{F}^L$, and π^C a center path. Let $F^L \in \mathcal{F}^L$, and $s^L \in S^L|_{F^L}$. Then the cost of a cheapest π^C-fork-compliant leaf path π^L ending in s^L equals the cost of a cheapest path from $I[F^L]_0$ to s^L_n in CompG$^\phi(\pi^C, F^L)$.*

Proof: Given a π^C-fork-compliant leaf path $\pi^L = \langle a^L_1, \ldots, a^L_m \rangle$, we can schedule each a^L_i as a (i) arc in CompG$^\phi(\pi^C, F^L)$ at the time step $t(i)$ assigned by the embedding. Connecting the resulting partial paths across time steps using the (ii) arcs, we get a path π from $I[F^L]_0$ to s^L_n in CompG$^\phi(\pi^C, F^L)$, whose cost equals that of π^L. Vice versa, given a path π from $I[F^L]_0$ to s^L_n in CompG$^\phi(\pi^C, F^L)$, remove the time indices of the vertices on π, and remove the arcs crossing time steps. This yields a π^C-fork-compliant leaf path π^L ending in s^L, whose cost equals that of π. So the π^C-fork-compliant leaf paths ending in s^L are in one-to-one correspondence with the paths from $I[F^L]_0$ to s^L_n in CompG$^\phi(\pi^C, F^L)$, showing the claim. □

To prepare our extension in the next section, we now reformulate the fork-decoupled state space Θ^ϕ. Each fork-decoupled state s is a center path $\pi^C(s)$, associated for every leaf $F^L \in \mathcal{F}^L$ with the π^C-fork-compliant path graph CompG$^\phi(\pi^C(s), F^L)$. The fork-decoupled initial state I^ϕ is the empty center path $\pi^C(I^\phi) = \langle\rangle$. The successor states s' of s are exactly the center paths extending $\pi^C(s)$ by one more center action. The fork-decoupled goal states are those s where $\pi^C(s)$ ends in a center goal state, and for every $F^L \in \mathcal{F}^L$ there exists a goal leaf state $s^L \in S^L|_{F^L}$ s.t. $s^L_{|\pi^C(s)|}$ is reachable from $I[F^L]_0$ in CompG$^\phi(\pi^C(s), F^L)$.

This formulation is different from GH's, but is equivalent given Lemma 1: Instead of maintaining the pricing functions prices(s) explicitly listing the costs of cheapest π^C-fork-compliant paths, we maintain the fork-compliant path graphs CompG$^\phi(\pi^C(s), F^L)$, from which these same costs can be obtained in terms of standard graph distance.

figure. This is just to keep the definition simple, in practice one can maintain only the reachable part of CompG$^\phi(\pi^C, F^L)$.

Star-Topology Decoupling

We now extend the concepts of compliant paths, and compliant path graphs, to handle star topologies instead of forks. A star topology is one where the center may interact arbitrarily with each leaf, and even effect-effect dependencies across leaves are allowed so long as they also affect the center:

Definition 4 (Star Factoring) *Let Π be an FDR task, and let $\mathcal{F}$ be a factoring. The* support-interaction graph *SuppIG$(\mathcal{F})$ of $\mathcal{F}$ is the directed graph whose vertices are the factors, with an arc $(F \to F')$ if $F \neq F'$ and there exist $v \in F$ and $v' \in F'$ such that $(v \to v')$ is an arc in SuppG. $\mathcal{F}$ is a* star factoring *if $|\mathcal{F}| > 1$ and there exists $F^C \in \mathcal{F}$ s.t. the following two conditions hold:*

(1) The arcs in SuppIG$(\mathcal{F})$ are contained in $\{(F^C \to F^L), (F^L \to F^C) \mid F^L \in \mathcal{F} \setminus \{F^C\}\}$.

(2) For every action a, if there exist $F^L_1, F^L_2 \in \mathcal{F} \setminus \{F^C\}$ such that $F^L_1 \neq F^L_2$ and $\mathcal{V}(\mathsf{eff}(a)) \cap F^L_1 \neq \emptyset$ as well as $\mathcal{V}(\mathsf{eff}(a)) \cap F^L_2 \neq \emptyset$, then $\mathcal{V}(\mathsf{eff}(a)) \cap F^C \neq \emptyset$.

F^C is the center *of $\mathcal{F}$, and all other factors $F^L \in \mathcal{F}^L := \mathcal{F} \setminus \{F^C\}$ are* leaves.

A star factoring $\mathcal{F}$ is strict *if the arcs in IG$(\mathcal{F})$ are contained in $\{(F^C \to F^L), (F^L \to F^C) \mid F^L \in \mathcal{F} \setminus \{F^C\}\}$.*

We cannot characterize star factorings in terms of just the causal graph, because the effect-effect arcs in that graph are inserted for all variable pairs in the effect: If there is an arc between two leaves, we cannot distinguish whether or not the same action also affects the center. In contrast, in a strict star factoring every action affects at most one leaf, which *can* be characterized in terms of just the causal graph. Hence strict star factorings are interesting from a practical perspective, allowing different/simpler factoring strategies.

Obviously, Definition 4 generalizes Definition 1. Perhaps less obvious is how far this generalization carries. As pointed out, an FDR task has a fork factoring iff its causal graph has more than one SCC. In contrast, *every* FDR task has a star factoring. In fact, any partition of the variables into two non-empty subsets is a star factoring: Calling one half of the variables the "center", and the other the "leaf", we have a (strict) star factoring, as Definition 4 does not apply any restrictions if there is a single leaf only.[2] That said, it is not clear whether single-leaf factorings are useful in practice. We get back to this when discussing our experiments.

To illustrate, consider what we will refer to as the *no-empty example*, where we forbid "empty truck moves". This is as before, except the precondition of move(x, y, z) is no longer $\{t_x = y\}$, but is $\{t_x = y, p = x\}$: A truck can only move if the package is currently inside it. The causal graph arcs now are not only $(t_A \to p)$ and $(t_B \to p)$ as before, but also $(p \to t_A)$ and $(p \to t_B)$. Hence there exists no fork factoring. But our previous factoring with $F^C = \{t_A, t_B\}$ and the single leaf $F^L = \{p\}$ is now a strict star factoring.

[2] Observe also that Definition 4 (2) can always be enforced wlog, simply by introducing redundant effects on F^C. However, actions affecting F^C are those our search must branch over, so this is just another way of saying "cross-leaf effects can be tackled by centrally branching over the respective actions". Doing so weakens the decoupling obtained, and for now we do not consider it.

To define compliance, we will be using the same terminology as before, i. e. center actions/paths and leaf actions/paths. These concepts are (mostly) defined as before, however their behavior is more complicated now.

The notions of center/leaf states, and of goal center/leaf states, remain the same. The center actions A^C are still all those actions affecting the center, and the leaf actions $A^L|_{F^L}$ for $F^L \in \mathcal{F}^L$ are still all those actions affecting F^L. However, *A^C and $A^L|_{F^L}$ are no longer disjoint*, as the same action may affect both A^C and $A^L|_{F^L}$.

A leaf path still is a sequence of leaf actions applicable to I when ignoring all center preconditions. The notion of center path changes, as now there may be leaf preconditions; we ignore these, i. e., *a center path is now a sequence of center actions applicable to I when ignoring all leaf preconditions*. As center and leaf paths may overlap, we need to clarify where to account for the cost of the shared actions. Our search, as we explain in a moment, views all A^C actions as part of the center, so we account for their costs there. To that end, *the cost of a leaf path π^L now is the summed-up cost of its $(A^L|_{F^L} \setminus A^C)$ actions*. By construction, these actions do not affect any factor other than F^L itself.

We will define star-compliant paths, and star-compliant path graphs $\mathit{CompG}^\sigma(\pi^C(s), F^L)$, below. For an overview before delving into these details, consider first the *star-decoupled state space* Θ^σ. A *star-decoupled state* s is a center path $\pi^C(s)$ associated for every leaf $F^L \in \mathcal{F}^L$ with the π^C-star-compliant path graph $\mathit{CompG}^\sigma(\pi^C(s), F^L)$. The *star-decoupled initial state* I^σ is the empty center path $\pi^C(I^\sigma) = \langle\rangle$. The *star-decoupled goal states* are those s where $\pi^C(s)$ ends in a center goal state, and for every $F^L \in \mathcal{F}^L$ there exists a goal leaf state $s^L \in S^L|_{F^L}$ s.t. $s^L_{|\pi^C(s)|}$ is reachable from $I[F^L]_0$ in $\mathit{CompG}^\sigma(\pi^C(s), F^L)$.

The definition of successor states s' of a star-decoupled state s changes more substantially. For forks, these simply were all center paths extending $\pi^C(s)$ by one more center action a^C. This worked due to the absence of leaf preconditions: by definition of "center path", $\mathsf{pre}(a^C)$ was satisfied at the end of $\pi^C(s)$. Given a star factoring instead, the successor states s' still result from extending $\pi^C(s)$ by one more center action a^C, but restricted to those a^C whose leaf preconditions can be satisfied at the end of $\pi^C(s)$. Namely, we require that, for every $F^L \in \mathcal{F}^L$, there exists $s^L \in S^L|_{F^L}$ such that $\mathsf{pre}(a^C)[F^L] \subseteq s^L$ and $s^L_{|\pi^C(s)|}$ is reachable from $I[F^L]_0$ in $\mathit{CompG}^\sigma(\pi^C(s), F^L)$.

In our original example, the star-decoupled initial state has two successors, from $\mathrm{move}(A, l_1, l_2)$ and $\mathrm{move}(B, l_3, l_2)$. In the no-empty example, only $\mathrm{move}(A, l_1, l_2)$ is present: Its precondition $p = A$ is reachable from $I[F^L]_0$ given the empty center path. But that is not so for the precondition $p = B$ of $\mathrm{move}(B, l_3, l_2)$.

Like for forks, we first identify a notion of compliant paths which captures how plans for the input task Π can be understood as center paths augmented with compliant leaf paths; and we then capture compliant paths in terms of compliant path graphs. However, the notion of "compliance" is now quite a bit more complicated. Given center path π^C and leaf path π^L, we require that (1) the sub-sequences of shared actions in π_L and π_C coincide, and (2) in between, we can schedule π^L at monotonically increasing points alongside π^C s.t. (2a) the center precondition of each leaf action holds in the respective center state and (2b) the F^L precondition of each center action holds in the respective leaf state.

Definition 5 (Star-Compliant Path) *Let Π be an FDR task, $\mathcal{F}$ a star factoring with center F^C and leaves $\mathcal{F}^L$, and π^C a center path. Let π^L be a leaf path for $F^L \in \mathcal{F}^L$. We say that π^L* star-complies *with π^C, also π^L is π^C-*star-compliant*, if the following two conditions hold:*

(1) The sub-sequence of A^C actions in π^L coincides with the sub-sequence of $A^L|_{F^L}$ actions in π^C.

(2) Assume, for ease of notation, dummy $A^C \cap A^L|_{F^L}$ actions added to start and end in each of π^C and π^L. For every pair $\langle a, a'\rangle$ of subsequent $A^C \cap A^L|_{F^L}$ actions in π^L and π^C, there exists an embedding at $\langle a, a'\rangle$.

Here, denote the sub-sequence of π^C between a and a' (not including a and a' themselves) by $\langle a^C_1, \dots, a^C_n\rangle$, and the F^C states it traverses by $\langle s^C_0, \dots, s^C_n\rangle$. Denote the sub-sequence of π^L between a and a' by $\langle a^L_1, \dots, a^L_m\rangle$, and the F^L states it traverses by $\langle s^L_0, \dots, s^L_m\rangle$. An embedding at $\langle a, a'\rangle$ *then is a monotonically increasing function $t : \{1, \dots, m\} \mapsto \{0, \dots, n\}$ so that both:*

(a) For every $i \in \{1, \dots, m\}$, $\mathsf{pre}(a^L_i)[F^C] \subseteq s^C_{t(i)}$.

(b) For every $t \in \{1, \dots, n\}$, $\mathsf{pre}(a^C_t)[F^L] \subseteq s^L_{i(t)}$ where $i(t) := \max\{i \mid t(i) < t\}$ (with $\max \emptyset := 0$).

To illustrate this, consider our no-empty example, the center path $\pi^C = \langle \mathrm{move}(A, l_1, l_2)\rangle$, and the leaf path $\pi^L = \langle \mathrm{load}(A, l_1), \mathrm{unload}(A, l_2)\rangle$. Definition 5 (1) is trivially fulfilled because the sub-sequences it refers to are both empty. For Definition 5 (2), we assume dummy shared actions, $\pi^C = \langle a, \mathrm{move}(A, l_1, l_2), a'\rangle$ and $\pi^L = \langle a, \mathrm{load}(A, l_1), \mathrm{unload}(A, l_2), a'\rangle$. The only pair of subsequent shared actions then is $\langle a, a'\rangle$. We need to find an embedding $t : \{1, 2\} \mapsto \{0, 1\}$ of $\langle a^L_1 = \mathrm{load}(A, l_1), a^L_2 = \mathrm{unload}(A, l_2)\rangle$ traversing F^L states $\langle s^L_0 = \{p = l_1\}, s^L_1 = \{p = A\}, s^L_2 = \{p = l_2\}\rangle$, into $\langle a^C_1 = \mathrm{move}(A, l_1, l_2)\rangle$ traversing F^C states $\langle s^C_0 = \{t_A = l_1\}, s^C_1 = \{t_A = l_2\}\rangle$. Given their center preconditions, we must schedule $\mathrm{load}(A, l_1)$ before $\mathrm{move}(A, l_1, l_2)$ and $\mathrm{unload}(A, l_2)$ behind $\mathrm{move}(A, l_1, l_2)$. So the only possibility is $t(1) := 0, t(2) := 1$. Indeed, that is an embedding: For Definition 5 (2a), $\mathsf{pre}(a^L_1)[F^C] = \{t_A = l_1\} \subseteq s^C_{t(1)} = s^C_0$, and $\mathsf{pre}(a^L_2)[F^C] = \{t_A = l_2\} \subseteq s^C_{t(2)} = s^C_1$. For Definition 5 (2b), $i(1) = \max\{i \mid t(i) < 1\} = 1$ because $t(1) = 0$ i. e. $a^L_1 = \mathrm{load}(A, l_1)$ is scheduled before $a^C_1 = \mathrm{move}(A, l_1, l_2)$. So $\mathsf{pre}(a^C_1)[F^L] = \{p = A\} \subseteq s^L_{i(1)} = s^L_1$ as required.

Despite the much more complex definition, the correspondence of compliant paths to plans for the original input planning task Π is as easily seen as for fork factorings. Say π is a plan for Π. The sub-sequence π^C of center actions in π is a center path. For a leaf $F^L \in \mathcal{F}^L$, the sub-sequence π^L of $A^L|_{F^L}$ actions in π is a leaf path. The sub-sequence of $A^C \cap A^L|_{F^L}$ actions in π^L coincides by construction with the sub-sequence of $A^C \cap A^L|_{F^L}$ actions in π^C, so we fulfill Definition 5 (1). Furthermore, between any pair of subsequent shared actions, all F^C preconditions of π^L, and all F^L

preconditions of π^C, must be satisfied because π is a plan, so we can read off an embedding fulfilling Definition 5 (2), and π^L is π^C-star-compliant. Vice versa, say center path π^C ends in a goal center state, and can be augmented for every $F^L \in \mathcal{F}^L$ with a π^C-star-compliant leaf path π^L ending in a goal leaf state. Note that, if an action a affects more than one leaf, by the definition of star factorings a must also affect the center, so by Definition 5 (1) the sub-sequences of such actions are synchronized via π^C: They must be identical for every leaf involved, and correspond to the same action occurences in π^C. Hence, sequencing all actions in π^C and every π^L according to the embeddings, we get an executable action sequence π achieving the overall goal in Π. Recall, finally, that we defined the cost of leaf paths to account only for those actions affecting just the leaf in question and nothing else. So, in both directions above, the cost of π equals the summed-up cost of the center path and leaf paths. We get that *the plans for* Π *are in one-to-one correspondence with center paths augmented with compliant leaf paths.*

We finally show how to capture π^C-star-compliant paths in terms of the weighted graphs $\mathit{CompG}^\sigma(\pi^C(s), F^L)$ we maintain alongside search over center paths in Θ^σ:

Definition 6 (Star-Compliant Path Graph) *Let* Π *be an FDR task,* $\mathcal{F}$ *a star factoring with center* F^C *and leaves* $\mathcal{F}^L$*, and* $\pi^C = \langle a_1^C, \dots, a_n^C \rangle$ *a center path traversing center states* $\langle s_0^C, \dots, s_n^C \rangle$*. The* π^C-star-compliant path graph *for a leaf* $F^L \in \mathcal{F}^L$*, denoted* $\mathit{CompG}^\sigma(\pi^C, F^L)$*, is the arc-labeled weigthed directed graph whose vertices are* $\{s_t^L \mid s^L \in S^L|_{F^L}, 0 \leq t \leq n\}$*, and whose arcs are as follows:*

- *(i)* $s_t^L \xrightarrow{a^L} s'^L_t$ *with weight* $c(a^L)$ *whenever* $s^L, s'^L \in S^L|_{F^L}$ *and* $a^L \in A^L|_{F^L} \setminus A^C$ *s.t.* $\mathsf{pre}(a^L)[F^C] \subseteq s_t^C$*,* $\mathsf{pre}(a^L)[F^L] \subseteq s^L$*, and* $s^L[\![a^L]\!] = s'^L$*.*
- *(ii)* $s_t^L \xrightarrow{0} s'^L_{t+1}$ *with weight* 0 *whenever* $s^L, s'^L \in S^L|_{F^L}$ *s.t.* $\mathsf{pre}(a_t^C)[F^L] \subseteq s^L$ *and* $s^L[\![a_t^C]\!] = s'^L$*.*

Item (i) is a benign change of Definition 3. Exactly as before, within each time step t the arcs correspond to those leaf-only actions whose center precondition is enabled at t. The only difference is that we need to explicitly exclude actions a^L affecting also the center (which for fork factorings cannot happen anyway). Item (ii) differs more substantially. Intuitively, whereas for fork factorings the $t \to t+1$ arcs simply stated that whichever leaf state we achieved before will survive the center action a_t^C (which could neither rely on, nor affect, the leaf), these arcs now state that the surviving leaf states are only those which comply with a_t^C's precondition, and will be mapped to possibly different leaf states by a_t^C's effect. Note that, if a_t^C has no precondition on F^L, then all leaf states survive, and if a_t^C has no effect on F^L, then all leaf states remain the same at $t+1$. If both is the case, then we are back to exactly the arcs (ii) in Definition 3.

For our no-empty task and $\pi^C = \langle \mathrm{move}(A, l_1, l_2) \rangle$, the π^C-star-compliant path graph is as shown in Figure 3.

Note the (only) difference to Figure 2: From time 0 to time 1, the only (ii) arc we have now is that from $(p = A)_0$ to $(p = A)_1$. This is because $\mathrm{move}(A, l_1, l_2)$ now has precondition $p = A$, so all other values of p do not comply with the center action being applied at this time step.

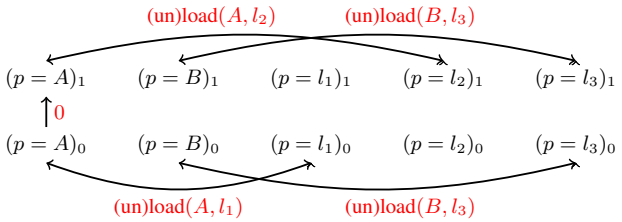


Figure 3: The star-compliant path graph for $\pi^C = \langle \mathrm{move}(A, l_1, l_2) \rangle$ in our no-empty example.

Lemma 2 *Let* Π *be an FDR task,* $\mathcal{F}$ *a star factoring with center* F^C *and leaves* $\mathcal{F}^L$*, and* π^C *a center path. Let* $F^L \in \mathcal{F}^L$*, and* $s^L \in S^L|_{F^L}$*. Then the cost of a cheapest* π^C*-star-compliant leaf path* π^L *ending in* s^L *equals the cost of a cheapest path from* $I[F^L]_0$ *to* s_n^L *in* $\mathit{CompG}^\sigma(\pi^C, F^L)$*.*

Proof: Consider first a π^C-star-compliant leaf path $\pi^L = \langle a_1^L, \dots, a_m^L \rangle$ for leaf $F^L \in \mathcal{F}^L$. By Definition 5 (1), the $A^C \cap A^L|_{F^L}$ sub-sequences in π^C and π^L coincide. Scheduling these at the respective time steps $t \to t+1$ in $\mathit{CompG}^\sigma(\pi^C, F^L)$, corresponding (ii) arcs must be present by construction. In between each pair of such actions, by Definition 5 we have embeddings t mapping the respective sub-sequence of π^L to that of π^C. Schedule each π^L action at its time step assigned by t. Then corresponding (i) arcs must be present by Definition 5 (2a). By Definition 5 (2b), if a π^C action here relies on an F^L precondition, then the corresponding leaf state satisfies that precondition so we have the necessary (ii) arc. Overall, we obtain a path π from $I[F^L]_0$ to s_n^L in $\mathit{CompG}^\sigma(\pi^C, F^L)$, and clearly the cost of π accounts exactly for the F^L-only actions on π^L, as needed.

Vice versa, consider any path π from $I[F^L]_0$ to s_n^L in $\mathit{CompG}^\sigma(\pi^C, F^L)$. Removing the time indices of the vertices on π, and removing those (ii) arcs $s_t^L \xrightarrow{0} s'^L_{t+1}$ where $s_t^L = s'^L_{t+1}$, clearly we obtain a π^C-star-compliant leaf path π^L ending in s^L, whose cost equals that of π.

So the π^C-star-compliant leaf paths ending in s^L are in one-to-one correspondence with the paths from $I[F^L]_0$ to s_n^L in $\mathit{CompG}^\sigma(\pi^C, F^L)$, showing the claim. □

Overall, goal paths in the star-decoupled state space Θ^σ correspond to center goal paths augmented with star-compliant leaf goal paths, which correspond to plans for the original planning task Π, of the same cost. So (optimal) search in Θ^σ is a form of (optimal) planning for Π.

Heuristic Search

GH show how standard classical planning heuristics, and standard search algorithms, can be applied to fork-decoupled search. All these concepts remain intact for star topologies; one issue requires non-trivial attention. (For space reasons, we omit details and give a summary only.)

A *heuristic* for Θ^σ is a function from star-decoupled states into $\mathbb{R}^{0+} \cup \{\infty\}$. The *star-perfect* heuristic, $h^{\sigma *}$, assigns to any s the minimum cost for completing s, i. e., reaching a star-decoupled goal state plus embedding compliant goal leaf paths. A heuristic h is *star-admissible* if $h \leq h^{\sigma *}$.

Given an FDR task Π and a star-decoupled state s, one can construct an FDR task $\Pi^\sigma(s)$ so that computing any admissible heuristic h on $\Pi^\sigma(s)$ delivers a star-admissible heuristic

value for s. $\Pi^\sigma(s)$ is like Π except for the initial state (center state of s, initial state for the leaves), and that new actions are added allowing to achieve each leaf state at its price in s.

Star-decoupled goal states are, as GH put it, *goal states with price tags*: Their path cost accounts only for the center moves, and we still have to pay the price for the goal leaf paths. In particular, $h^{\sigma *}$ is *not* 0 on star-decoupled goal states. We can obtain a standard structure Θ' for search as follows. Introduce a new goal state G. Give every star-decoupled goal state s an outgoing transition to G whose cost equals the summed-up cost of cheapest compliant goal leaf paths in s. Given a heuristic h for Θ^σ, set $h(G) := 0$.

The generalization to star-decoupling incurs one important issue, not present in the special case of fork factorings. If center moves require preconditions on leaves, then we should "buy" these preconditions immediately, putting their price into the path cost g, because otherwise we lose information during the search. For illustration, in our no-empty example, say the goal is $t_A = l_2$ instead of $p = l_3$, and consider the star-decoupled state s after applying move(A, l_1, l_2). Then $t_A = l_2$ is true, $g = 1$, and h^* on the compiled FDR task $\Pi^\sigma(s)$ returns 0 because the goal is already true. But $h^{\sigma *}(s) = 1$ and the actual cost of the plan is 2: We still need to pay the price for the precondition $p = A$ of move(A, l_1, l_2). This is not captured in $\Pi^\sigma(s)$ because it is needed *prior* to s only. The solution is to perceive this "price" as a "cost" already committed to. In our modified structure Θ', when applying a center action a^C to star-decoupled state s, we set the local cost of a^C (its cost specifically at this particular position in Θ') to $\mathsf{cost}(a_t^C) + \sum_{F^L} g(F^L)$. Here, $g(F^L)$ is the minimum over the price in s of those $s^L \in S^L|_{F^L}$ that satisfy a^C's precondition. Intuitively, to apply a^C, we must first buy its leaf preconditions. To reflect that $g(F^L)$ has already been paid, the respective (ii) arcs in $\mathit{CompG}^\sigma(\pi^C(s), F^L)$ are assigned weight $-g(F^L)$. In our example above, the path cost in s is $g = 2$ giving us the correct $g + h = 2$. The "0" arc in Figure 3 is assigned weight -1, so that the overall cost of the compliant path for p will be 0 (as the only action we need to use has already been paid for by the center move).

Any (optimal) standard heuristic search algorithm X on Θ' yields an optimal heuristic search algorithm for Θ^σ, which we denote *Star-Decoupled X (SDX)*.

Experiments

Our implementation is in FD (Helmert 2006), extending that for fork decoupling by GH. We ran all international planning competition (IPC) STRIPS benchmarks ('98–'14), on a cluster of Intel E5-2660 machines running at 2.20 GHz, with time (memory) cut-offs of 30 minutes (4 GB).

Our experiments are preliminary in that we perform only a very limited exploration of factoring strategies. Factoring strategy design for star topologies is, in contrast to fork topologies, quite challenging. The space of star factorings includes arbitrary two-subset partitions of single-SCC causal graphs, where fork factorings do not exist at all. Even for the simplest possible optimization criterion, maximizing the number of leaves in a strict star factoring, finding an optimal factoring is **NP**-complete (this follows by a straightforward reduction from Maximum Independent Set (Garey and Johnson 1979)). An additional complication is that leaves may be "frozen": As we need to branch over all actions affecting the center, for a leaf F^L to yield a state space size reduction there must be at least one action affecting *only* F^L (not affecting the center). For example, in IPC Visit-All, while the robot position may naturally be viewed as the "center" and each "visited" variable as a leaf, every leaf-moving action also affects the center so nothing is gained.

We shun this complexity here, leaving its comprehensive exploration to future work, and instead design only two simple strict-star factoring strategies by direct extension of GH's fork factoring strategy. That strategy works as follows.

Denote by $\mathcal{F}^{\mathrm{SCC}}$ the factoring whose factors are the SCCs of *CG*. View the interaction graph $\mathit{IG}(\mathcal{F}^{\mathrm{SCC}})$ over these SCCs as a DAG where the root SCCs are at the top and the leaf SCCs at the bottom. Consider the "horizontal lines" $\{T, B\}$ (top, bottom) through that DAG, i. e., the partitions of V where every $F \in \mathcal{F}^{\mathrm{SCC}}$ is fully contained in either of T or B, and where the only arc in $\mathit{IG}(\{T, B\})$ is $(T \to B)$. Let $\mathcal{W}$ be the set of weakly connected components of $\mathcal{F}^{\mathrm{SCC}}$ within B. Then a fork factoring $\mathcal{F}$ is obtained by setting $F^C := T$ and $\mathcal{F}^L := \mathcal{W}$. Any fork factoring can be obtained in this manner, except the *redundant* ones where some $F^L \in \mathcal{F}^L$ contains several weakly connected components.

GH's strategy moves the horizontal line upwards, from leaves to roots in $\mathit{IG}(\mathcal{F}^{\mathrm{SCC}})$, in a greedy fashion, thereby generating a sequence $\mathcal{F}_1, \ldots, \mathcal{F}_k$ of fork factorings. They select the factoring $\mathcal{F}_i$ whose number of leaf factors is maximal, and whose index i is minimal among these factorings. The rationale behind this is to maximize the number of leaves (the amount of conditional independence) while keeping these as small as possible (reducing the runtime overhead). If $k = 0$ (no horizontal line exists i. e. *CG* is a single SCC), or $\mathcal{F}_i$ has a single leaf only, then GH *abstain* from solving the input task. The rationale is that, in GH's experiments, single-leaf factorings hardly ever payed off.

We design two new (non-fork) strategies, *inverted forks* and *X-shape*. The former is exactly GH's strategy but inverting the direction of the arcs in the causal graph. The latter runs GH's fork factoring first, and thereafter runs inverted forks on the fork center component F^C. If an inverted-fork leaf F^L has an outgoing arc into a fork leaf, then F^L is included into F^C. We abstain if no factoring exists or if the selected factoring has a single leaf only. Note that this still abstains on single-SCC causal graphs, and that frozen leaves cannot occur as neither forks nor inverted forks allow leaf-affecting actions to affect the center. The strategies take negligible runtime (rounded to 0.00 in most cases, much faster than FD's pre-processes in the few other cases).

For optimal planning (with LM-cut (Helmert and Domshlak 2009)), while GH reported dramatic gains using fork factoring, our new factoring strategies do not improve much upon these gains. Inverted forks do sometimes help, most notably in Satellite where Star-Decoupled A* (SDA*) with inverted fork factoring solves 3 more instances than each of A* and fork-factoring SDA*, reducing evaluations on commonly solved instances by up to two orders of magnitude.

	(A): Coverage												(B): Evaluations: Improvement factor relative to GBFS												(C): Runtime: Improvement factor relative to GBFS											
	X-shape						inverted fork			fork			X-shape																							
													no preferred operators					with preferred operators							no preferred operators					with preferred operators					
														SDGBFS				SDGBFS				LAMA				SDGBFS				SDGBFS				LAMA		
		npo		po				npo	po		npo	po																								
	#	base	sd	base	sd	LAMA	#	sd	sd	#	sd	sd	#	∑D	GM	max	#	∑D	GM	max	∑D	GM	max	#	∑D	GM	max	#	∑D	GM	max	∑D	GM	max
Childsna	20	0	0	3	+1	-3	20	0	+1	0			0				0							0				0							
Depots	22	14	-1	18	-1	+3	22	-1	-1	0			12	0.5	1.1	14.0	17	1.5	2.1	54.5	3.4	2.3	299	10	0.3	0.3	5.7	14	1.0	0.7	22.5	0.8	1.3	91.0
Driver	20	18	+1	20	-2	0	0			20	+1	-2	17	1.2	2.2	8.3	18	0.1	0.9	4.0	0.7	0.9	21.0	5	0.3	0.4	3.3	12	0.0	0.2	1.0	0.5	0.6	2.1
Elev08	30	30	0	30	0	0	30	0	0	0			30	5.5	5.4	18.1	30	4.2	3.5	12.1	1.0	1.1	3.8	24	2.0	1.5	4.0	20	1.4	1.1	2.0	0.6	0.6	1.4
Elev11	20	18	+2	20	0	0	20	+2	0	0			18	12.6	7.5	46.8	20	11.5	5.6	37.3	1.8	1.1	4.3	18	4.8	2.7	18.5	20	4.7	1.8	13.0	1.3	0.7	2.8
Floor11	20	6	-5	6	-4	0	20	-5	-4	0			1	0.4	0.4	0.4	2	0.7	0.7	0.7	0.8	0.8	1.0	1	0.1	0.1	0.1	2	0.1	0.1	0.1	0.2	0.2	0.3
Floor14	20	2	-2	2	0	0	20	-2	0	0			0				2	0.6	0.6	0.7	0.9	0.9	1.1	0				2	0.1	0.1	0.1	0.2	0.2	0.3
Log00	28	28	0	28	0	0	28	0	0	28	0	0	28	11.8	9.4	18.8	28	5.2	4.7	6.8	0.8	0.9	1.0	0				0							
Log98	35	26	+9	35	0	0	35	+9	0	35	+9	0	26	37.3	12.7	60.0	35	49.4	3.5	177	9.0	1.1	28.1	16	19.0	6.7	34.9	24	19.1	1.5	65.8	4.3	0.8	18.0
Mico	145	145	0	145	0	0	0			145	0	0	145	2.4	2.2	4.4	145	4.3	3.6	8.4	1.0	1.0	1.8	26	1.1	1.1	1.6	45	1.0	1.0	1.2	0.5	0.6	0.9
Mystery	2	0	0	1	0	0	2	0	0	0			0				1	0.9	0.9	0.9	1.4	1.3	1.3	0				1	0.6	0.6	0.6	1.0	1.0	1.0
NoMy	20	9	+10	10	+9	+3	0			20	+10	+9	9	141	34.0	1250	10	43K	33.5	15M	1697	4.5	135K	5	2.6	2.2	5.9	8	1033	3.8	12K	468	2.1	10K
Pathw	30	11	+2	20	0	+4	0			29	+2	0	11	12.7	3.8	26.5	20	1.4	1.5	1.8	0.6	0.7	1.2	7	3.5	1.3	10.7	10	0.4	0.6	1.0	0.3	0.4	1.0
PSR	3	3	0	3	0	0	0			3	0	0	3	1.6	1.7	1.9	3	1.3	1.2	1.6	1.0	1.0	1.0	0				0						
Rovers	40	23	-1	40	0	0	38	+5	-1	40	-1	0	22	1.2	1.7	12.3	40	1.3	1.7	3.1	0.7	0.8	1.8	12	0.4	0.5	1.7	22	0.6	0.7	1.0	0.3	0.4	0.8
Satell	36	30	+3	36	0	0	34	+1	0	36	+3	0	30	2.7	2.0	38.7	36	1.8	1.3	6.8	0.4	0.5	1.5	19	1.1	1.8	21.9	25	1.9	1.0	3.9	0.1	0.3	0.8
TPP	29	22	+1	29	0	0	26	-5	0	27	+1	0	22	3183	454	31K	29	0.1	3.4	41.9	0.8	0.8	1.8	14	95.8	52.7	987	17	0.0	0.2	1.3	0.5	0.6	0.9
Transp08	30	16	+14	28	+2	+2	30	+14	+2	0			16	693	35.7	2226	28	305	21.0	4360	1.0	1.0	3.3	10	135	17.6	262	22	18.0	2.0	156	0.6	0.7	1.7
Transp11	20	0	+20	11	+9	+7	20	+20	+9	0			0				11	500	114	4360	1.0	1.1	8.9	0				11	19.0	5.2	141	0.7	0.7	4.7
Transp14	20	0	+20	6	+14	+9	20	+20	+14	0			0				6	168	130	407	1.1	1.1	6.9	0				6	8.7	7.4	23.9	0.6	0.8	4.4
Wood08	26	26	0	26	0	0	25	0	0	25	0	0	26	0.4	1.2	121	26	1.2	2.3	111	7.8	14.8	145	20	0.1	0.2	2.1	22	0.2	0.3	7.9	2.1	2.4	6.3
Wood11	19	18	0	19	0	0	18	+1	0	18	+1	0	2	0.4	0.5	4.3	19	1.2	1.2	3.0	11.8	15.8	53.8	16	0.0	0.1	0.5	18	0.0	0.1	0.2	2.1	2.1	4.9
Zeno	20	20	0	20	0	0	18	0	0	20	0	0	20	29.3	12.1	103	20	4.6	3.0	6.3	0.8	0.8	1.3	7	4.2	3.8	10.6	7	0.9	0.8	1.1	0.4	0.4	0.7
∑	655	465	+73	556	+28	+25	426	+59	+20	446	+26	+7																							

Table 1: Results in satisficing planning with h^{FF}. Each table section fixes a factoring strategy (2nd row from top), and uses *only the instances which that strategy does not abstain on*; of these, (B) uses the subset of commonly solved ones (with vs. without preferred operators), (C) the same but excluding ones commonly solved in ≤ 0.1 seconds. The respective numbers of underlying instances are shown in columns "#". GBFS: Greedy Best-First Search; npo: no preferred ops; po: with preferred ops (FD's dual-queue search); base: baseline (GBFS on standard state space); sd: star-decoupled base; $\sum$D: factor over the per-domain sums; GM/max: Geometric mean/maximum over the per-instance factors; K: thousand; M: million.

But such cases are rare. It remains future work to explore more sophisticated star-factoring strategies for optimal planning. Here, we focus on satisficing planning where even our current simple factoring strategies yield good results.

Consider Table 1. X-shape factoring abstains much less than each of the "base strategies", forks and inverted forks, on its own. X-shape factoring substantially improves coverage, overall and in several domains, without preferred operators, and does so in NoMystery and Transport with preferred operators. It even beats LAMA (Richter and Westphal 2010) overall, thanks to excelling in Transport. Indeed, with preferred operators, Transport is solved almost instantly, with a maximum (average) of 69 (36.1) state evaluations and 14.7 (2.7) seconds runtime. Without preferred operators, the average is 1236.8 evaluations and 127.1 seconds. Substantial reductions of evaluations, with corresponding smaller but still significant reductions of runtime, are obtained also in Elevators, Logistics, NoMystery, Pathways, TPP, and Zenotravel, plus smaller improvements in various other cases.

The strength of X-shape factoring is inherited from fork factoring in NoMystery and Pathways, where inverted fork factoring abstains. It stems from inverted fork factoring, and is thus entirely thanks to the new techniques introduced herein, in Elevators and Transport where fork factoring abstains. For Logistics, TPP, and Zenotravel, where neither base strategy abstains, our current configuration of X-shape factoring uses exactly the fork factorings, because the fork strategy is run first and thus given a "preference". To illustrate, in Logistics, the fork makes each package a leaf. The inverted fork would make the trucks leaves, but they have outgoing arcs to the packages so are re-included into the center. Symmetrically, if the X-shape ran the inverted fork strategy first, it would end up with exactly the inverted fork factorings. In Logistics98, these have a consistent advantage ($\sum$D evaluations improvement factor over forks is 6.4 with preferred operators). In Logistics00 they have a consistent small disadvantage ($\sum$D factor 0.9). In TPP and Zenotravel, there is significant per-instance variance, up to 14 times worse respectively 537 times better in TPP, and up to 2 times worse respectively 5 times better in Zenotravel. So there may be room for improvement by combining forks and inverted forks in a less simplistic manner.

As we are mainly interested in speed, the above considers uniform action costs. The results for non-uniform action costs are very similar, except in Elevators where both the baseline and X-shape factoring get worse, in Transport where only the baseline gets worse, and in Woodworking where only X-shape factoring without preferred operators gets worse. The overall coverage gain is then 49 without preferred operators, and 35 with preferred operators.

Conclusion

For our current simple factoring strategies, the upshot from our experiments is that the main empirical advantage of star topologies over forks (as far as IPC benchmarks are concerned) stems from the Transport domain, which this technique "kills" completely. The most important research question in planning remains the exploration of factoring strategies. We need methods able to identify sophisticated star topologies, and we need to understand the strengths of different factorings as a function of problem structure.

Beyond planning, an interesting opportunity is to apply star-topology decoupling to star-topology systems. Verification & synthesis appear especially relevant, possibly with adversarial or probabilistic extensions to star-decoupling. Some classical puzzles may also be amenable; e. g., in the Peg Solitaire game, one can take the "center" to be the middle of the board and the "leaves" to be its peripheral parts.

Acknowledgments. We thank the anonymous reviewers, whose comments helped to improve the paper.

References

Amir, E., and Engelhardt, B. 2003. Factored planning. In Gottlob, G., ed., *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)*, 929–935. Acapulco, Mexico: Morgan Kaufmann.

Bäckström, C., and Nebel, B. 1995. Complexity results for SAS^+ planning. *Computational Intelligence* 11(4):625–655.

Brafman, R., and Domshlak, C. 2003. Structure and complexity in planning with unary operators. *Journal of Artificial Intelligence Research* 18:315–349.

Brafman, R. I., and Domshlak, C. 2006. Factored planning: How, when, and when not. In Gil, Y., and Mooney, R. J., eds., *Proceedings of the 21st National Conference of the American Association for Artificial Intelligence (AAAI-06)*, 809–814. Boston, Massachusetts, USA: AAAI Press.

Brafman, R. I., and Domshlak, C. 2008. From one to many: Planning for loosely coupled multi-agent systems. In Rintanen, J.; Nebel, B.; Beck, J. C.; and Hansen, E., eds., *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS'08)*, 28–35. AAAI Press.

Brafman, R., and Domshlak, C. 2013. On the complexity of planning for agent teams and its implications for single agent planning. *Artificial Intelligence* 198:52–71.

Fabre, E.; Jezequel, L.; Haslum, P.; and Thiébaux, S. 2010. Cost-optimal factored planning: Promises and pitfalls. In Brafman, R. I.; Geffner, H.; Hoffmann, J.; and Kautz, H. A., eds., *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS'10)*, 65–72. AAAI Press.

Garey, M. R., and Johnson, D. S. 1979. *Computers and Intractability—A Guide to the Theory of NP-Completeness*. San Francisco, CA: Freeman.

Gnad, D., and Hoffmann, J. 2015. Beating lm-cut with h^{max} (sometimes): Fork-decoupled state space search. In Brafman, R.; Domshlak, C.; Haslum, P.; and Zilberstein, S., eds., *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS'15)*. AAAI Press.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In Gerevini, A.; Howe, A.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*, 162–169. AAAI Press.

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

Hoffmann, J. 2011. Analyzing search topology without running any search: On the connection between causal graphs and h^+. *Journal of Artificial Intelligence Research* 41:155–229.

Jonsson, P., and Bäckström, C. 1995. Incremental planning. In *European Workshop on Planning*.

Kelareva, E.; Buffet, O.; Huang, J.; and Thiébaux, S. 2007. Factored planning using decomposition trees. In Veloso, M., ed., *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, 1942–1947. Hyderabad, India: Morgan Kaufmann.

Knoblock, C. 1994. Automatically generating abstractions for planning. *Artificial Intelligence* 68(2):243–302.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research* 39:127–177.

Wang, D., and Williams, B. C. 2015. tburton: A divide and conquer temporal planner. In Bonet, B., and Koenig, S., eds., *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI'15)*, 3409–3417. AAAI Press.

Goal Recognition Design With Non-Observable Actions

Sarah Keren and **Avigdor Gal**
{sarahn@tx,avigal@ie}.technion.ac.il
Technion — Israel Institute of Technology

Erez Karpas
karpase@csail.mit.edu
Massachusetts Institute of Technology

Abstract

Goal recognition design involves the offline analysis of goal recognition models by formulating measures that assess the ability to perform goal recognition within a model and finding efficient ways to compute and optimize them. In this work we extend earlier work in goal recognition design to partially observable settings. Partial observability is relevant to goal recognition applications such as assisted cognition and security that suffer from reduced observability due to sensor malfunction, deliberate sabotage, or lack of sufficient budget. We relax the full observability assumption by offering a new generalized model for goal recognition design with non-observable actions. In particular we redefine the *worst case distinctiveness* (*wcd*) measure to represent the maximal number of steps an agent can take in a system before the observed portion of his trajectory reveals his objective. We present a method for calculating the *wcd* based on novel compilations to classical planning and propose a method to improve the design. Our empirical evaluation shows the proposed solutions to be effective in computing and improving the *wcd*.

Introduction

Goal recognition design (*grd*) (Keren, Gal, and Karpas 2014; 2015) involves the offline analysis of goal recognition models, interchangeably called in the literature plan recognition (Pattison and Long 2011; Kautz and Allen 1986; Cohen, Perrault, and Allen 1981; Lesh and Etzioni 1995; Ramirez and Geffner 2009; Agotnes 2010; Hong 2001), by formulating measures that assess the ability to perform goal recognition within a model and finding efficient ways to compute and optimize them.

Goal recognition design is applicable to any domain for which quickly performing goal recognition is essential and in which the model design can be controlled. In particular goal recognition design is relevant to goal and plan recognition applications such as assisted cognition (Kautz et al. 2003) and security (Jarvis, Lunt, and Myers 2004; Kaluza, Kaminka, and Tambe 2011; Boddy et al. 2005) that suffer from reduced observability due to sensor malfunction, deliberate sabotage, or lack of sufficient budget. In a safe home setting, for example, reduced coverage means less control over access to sensitive areas such as a hot oven. For intrusion detection applications, malfunctioning sensors may result in an unobserved path to sensitive zones.

Earlier works on goal recognition design (Keren, Gal, and Karpas 2014; 2015) fail to provide support for goal recognition with reduced observability due to its reliance on the assumption that the model being analyzed is fully observable. In this work we relax the full observability assumption and offer innovative tools for a goal recognition design analysis that accounts for a model with non-observable actions and a design process that involves sensor placement.

The set of actions in the *partially observable* setting is partitioned into observable and non-observable actions, reflecting for example a partial sensor coverage. The proposed analysis of observations relies on the partial incoming stream of observations. A key feature of this setting is that it supports a scenario where the system has no information regarding the actions of the agents beyond what is observed. Therefore, in the absence of an observation, the system cannot differentiate unobserved actions from idleness of an agent. An example of such a scenario can be found in Real Time Location Systems (RTLS) where the last known location of an agent is taken as its current position.

The *partially observable* setting provides three extensions to the goal recognition design state-of-the-art. First, we support a partially observable model by defining an observation sequence that contains only the observable actions performed by an agent. This means that each such sequence may be generated by more than one execution sequence. Accordingly, a non-distinctive observation sequence is one that *satisfies* (formally defined in this paper) paths to more than one goal. The *worst case distinctiveness*(*wcd*) is then the length of the maximal execution that produces a non-distinctive observation sequence. The *wcd* serves as an upper bound on the number of observations that need to be collected for each agent before guaranteeing his objective is recognized.

As a second extension, we provide a compilation of the *partially observable* goal recognition design problem into a classical planning problem, which allows us to exploit existing tools for calculating the *wcd*. Our empirical analysis shows that the compilation allows efficient computation of the *wcd*.

After calculating the *wcd* of a model, it may be desired to minimize it. The third extension we present therefore in-

volves finding the optimal set of modifications that can be introduced to the model in order to reduce the *wcd*. We introduce a new design-time modification method that involves exposing non-observable actions, *e.g.*, by (re)placing sensors. This modification method is used in addition to removing actions from the model (Keren, Gal, and Karpas 2014) in order to minimize the *wcd* while respecting the specified restrictions on the number of allowed modifications. The empirical analysis reveals that the combination of observation exposure and activity restriction allows bigger improvements than each of the measures separately.

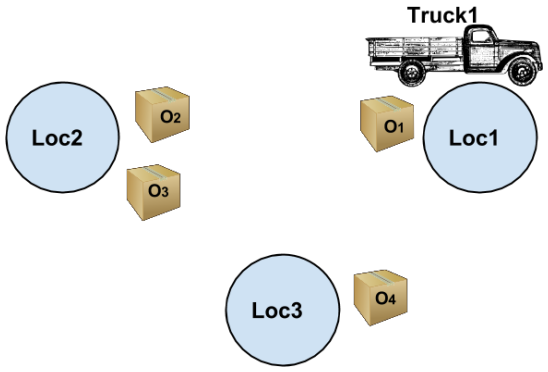


Figure 1: An example of a goal recognition design problem

Example 1 *To illustrate the objective of calculating and optimizing the* wcd *of a goal recognition design model, consider the example depicted in Figure 1, which demonstrates a simplified setting from the logistics domain. There are* 3 *locations, a single truck that is initially located at position* $Loc1$*, and* 4 *objects that are initially placed such that* O_1 *is at location* $Loc1$*,* O_2 *and* O_3 *are at* $Loc2$ *and* O_4 *is at* $Loc3$*. Objects can be moved by loading them onto the truck and unloading them in their destination after the truck reaches it. There are two goals: 1)* O_1 *and* O_2 *at* $Loc3$ *and* O_4 *at* $Loc1$ *(*g_0*) and 2)* O_3 *at* $Loc3$ *(*g_1*). Optimal plans are the only valid plans in this example. In the fully observable setting* wcd $= 0$*, since the goal is revealed by the first action, which can either be* $LoadO_1$ *for* g_0 *or* $Drive\text{-}Loc1\text{-}Loc2$ *for* g_1*. Assume that the non-observable actions are all the load and unload actions, and the observable actions are only the movement of the truck. Here, because the truck needs to travel from* $Loc1$ *to* $Loc2$ *and then to* $Loc3$ *for achieving both goals the goal is revealed only if* $Drive\text{-}Loc3\text{-}Loc1$ *is performed. This means that* g_1 *can be achieved without the system being aware of it. Exposing* $LoadO_1$*, by placing a sensor on the object, changes the situation dramatically.*

The rest of the paper is organized as follows. We start by providing background on classical planning, followed by introducing a model of *grd* problems and the *wcd* value for the *partially observable* setting. We continue by presenting methods for calculating and reducing the *wcd* value of a *grd* problem, respectively. We conclude with an empirical evaluation, a discussion of related work, and concluding remarks.

Background

The basic form of automated planning, referred to as *classical planning*, is a model in which the actions of agents are fully observable and deterministic. A common way to represent classical planning problems is the STRIPS formalism (Fikes and Nilsson 1972): $P = \langle F, I, A, G, C \rangle$ where F is the set of fluents, $I \subseteq F$ is the initial state, $G \subseteq F$ represents the set of goal states, and A is a set of actions. Each action is a triple $a = \langle pre(a), add(a), del(a) \rangle$, that represents the precondition, add, and delete lists respectively, and are all subsets of F. An action a is applicable in state s if $pre(a) \subseteq s$. If action a is applied in state s, it results in a new state $s' = (s \setminus del(a)) \cup add(a)$. $C : A \rightarrow \mathbb{R}_0^+$ is a cost function that assigns each action a non-negative cost.

The objective of a planning problem is to find a plan $\pi = \langle a_1, \ldots, a_n \rangle$, a sequence of actions that brings an agent from I to a goal state. The cost $c(\pi)$ of a plan π is $\Sigma_{i=1}^{n}(C(a_i))$. Often, the objective is to find an optimal solution for P, an optimal plan, π^*, that minimizes the cost. We assume the input of the problem includes actions with a uniform cost equal to 1. Therefore, plan cost is equivalent to plan length, and the optimal plans are the shortest ones.

Model

A model for partially observable goal recognition design (*pogrd*) is given as $D = \langle P_D, \mathcal{G}_D, \Pi_{leg}(\mathcal{G}_D) \rangle$ where:

- $P_D = \langle F_D, I_D, A_D \rangle$ is a planning domain where $A = A^o \cup A^{no}$ is a partition of A into observable and non-observable actions, respectively.
- $\mathcal{G}_D$ is a set of possible goals, where each possible goal $g \in \mathcal{G}_D$ is a subset of F_D.
- $\Pi_{leg}(\mathcal{G}_D) = \bigcup_{g \in \mathcal{G}_D} \Pi_{leg}(g)$ is the set of *legal* plans to each of the goals. A plan is an execution of actions that take the agent from I to a goal in $\mathcal{G}_D$. A legal plan is one that is allowed under the assumptions made on the behavior of the agent.

It is worth noting that the model includes an initial state, common to all agents acting in the system. In case there are multiple initial states, there is a simple compilation, which adds a zero cost transition between a dummy common initial state and each initial state, making the model and the methods we propose applicable.

The *pogrd* model divides the system description into three elements, namely *system dynamics*, defined by F_D, I_D, A_D and $\mathcal{G}_D$, *agent strategy* defined by $\Pi_{leg}(\mathcal{G}_D)$, and *observability* defined by the partition of A into A^o and A^{no}. Whenever D is clear from the context we shall use P, $\mathcal{G}$, and $\Pi_{leg}(\mathcal{G})$.

A *path* is a prefix of a legal *plan*. We denote the set of paths in D as $\Pi_{pref}(\mathcal{G}_D)$ and the set of paths to goal $g \in \mathcal{G}_D$ as $\Pi_{pref}(g)$. An *observation sequence* $\vec{o} = \langle a_1, \cdots, a_n \rangle$ is a sequence of actions $a_j \in A^o$. For any two action sequences $\langle a_1, \cdots, a_n \rangle$ and $\langle a_1^{'}, \cdots, a_m^{'} \rangle$ the concatenation of the action sequences is denoted by $\langle a_1, \cdots, a_n \rangle \cdot \langle a_1^{'}, \cdots, a_m^{'} \rangle$.

In the *partially observable* setting the observations sequence that is produced by a path includes only the observable actions that are performed. The relationship between a path and an observation sequence is formally defined next.

Definition 1 *Given a path π, the* observable projection *of π in D, denoted $op_D(\pi)$ ($op(\pi)$ when clear from the context), is recursively defined as follows:*

$$op(\pi) = \begin{cases} \langle\rangle & \text{if } \pi = \langle\rangle \\ \langle a_1\rangle \cdot op(\langle a_2 \cdots a_n\rangle) & \text{if } \pi = \langle a_1, \cdots, a_n\rangle \text{ and } a_1 \in A^o \\ op(\langle a_2, \cdots, a_n\rangle) & \text{otherwise} \end{cases}$$

It is worth noting that the fully observable settings presented by (Keren, Gal, and Karpas 2014; 2015) is a special case of the model presented here, in which the entire action set is observable. In this case, $A^{no} = \emptyset$, $A^o = A$, and the observable projection of any action sequence is equivalent to the action sequence itself.

The relation between an observation sequence and a goal is defined as follows.

Definition 2 *An observation sequence $\vec{o}$ satisfies a goal g if $\exists \pi \in \Pi_{pref}(g)$ s.t. $\vec{o} = op(\pi)$.*

We now define non-distinctive observation sequences and paths, as follows.

Definition 3 *$\vec{o}$ is a* non-distinctive observation sequence *if it satisfies more than one goal. Otherwise, it is* distinctive*.*
π is a non-distinctive path *if its observable projection $\vec{o}$ is non-distinctive. Otherwise, it is* distinctive*.*

The following basic observation sets the relationship between a path and its prefixes.

Lemma 1 *Any prefix of a non-distinctive path is non-distinctive.*

Proof: Let π be a non-distinctive path. According to Definition 3, $\exists g, g' \in \mathcal{G}, \pi \in \Pi_{pref}(g), \pi' \in \Pi_{pref}(g')$ s.t. $g' \neq g$ and $op(\pi) = op(\pi')$. Let π_{pre} be a prefix of π. Then, by Definition 1, $op(\pi_{pre})$ is a prefix of $op(\pi) = op(\pi')$. According to Definition 2, $\pi_{pre} \in \Pi_{pref}(g')$ and therefore, according to Definition 3, π_{pre} is non-distinctive. ∎

Given Lemma 1, we define a maximal non-distinctive prefix of a path π (dubbed $pre_{nd}(\pi)$) to be a prefix of π such that there is no other non-distinctive prefix of π that contains it.

We now examine the effect of moving an action from A^o to A^{no} (dubbed *concealment*) on the number of goals a path satisfies. Relying on Definition 2, we mark the set of goals a path π satisfies in D as $\mathcal{G}_D(\pi)$ and we use $\Pi_{op_D(\pi)}$ to represent the paths π' in D s.t. $op_D(\pi) = op_D(\pi')$.

Theorem 1 *Let D and D' be two* pogrd *models that are identical except that $A^{no}_D \subseteq A^{no}_{D'}$. For any $\pi \in \Pi_{pref}(\mathcal{G}_D)$, $\mathcal{G}_D(\pi) \subseteq \mathcal{G}_{D'}(\pi)$. If, in addition, $\forall a \in A^{no}_{D'} \setminus A^{no}_D$, $\hat{\pi}$ is distinctive in D' for any prefix $\hat{\pi} \cdot \langle a\rangle$ of π, then $\mathcal{G}_D(\pi) = \mathcal{G}_{D'}(\pi)$ (strict equivalence).*

Proof: In general, if $\forall a \in \pi$, $a \notin A^{no}_{D'} \setminus A^{no}_D$ then $op_D(\pi) = op_{D'}(\pi)$ since none of the actions in π changed their observability property. Therefore, $\mathcal{G}_D(\pi) = \mathcal{G}_{D'}(\pi)$.

Otherwise, since $\pi \in \Pi_{pref}(\mathcal{G}_D)$ there exists a goal $g \in \mathcal{G}_D$ and $\pi_g \in \Pi_{leg}(g)$ s.t. $op_D(\pi)$ is a prefix to $op_D(\pi_g)$ (Definition 2). By eliminating any action $a \in A^{no}_{D'} \setminus A^{no}_D$ both in $op_D(\pi)$ and in $op_D(\pi_g)$ we maintain the prefix property. Therefore, $op_{D'}(\pi)$ is a prefix to $op_D(\pi_g)$ and thus $\pi \in \Pi_{pref}(\mathcal{G}_{D'})$.

The only thing left to show now is that if $\hat{\pi}$ is distinctive in D' then $\mathcal{G}_D(\pi) = \mathcal{G}_{D'}(\pi)$. $\hat{\pi}$ is distinctive in D' and therefore $op_{D'}(\hat{\pi})$ satisfies a single goal g (Definition 3). π achieves the goal g and $\hat{\pi}$ is a prefix of π. Therefore, $op_D(\hat{\pi})$ satisfies g as well. $op_{D'}(\hat{\pi} \cdot \langle a\rangle) = op_{D'}(\hat{\pi})$ and therefore also satisfies g.

Assume to the contrary that $\hat{\pi} \cdot \langle a\rangle$ is non-distinctive. Therefore, there must be at least one more goal $g' \neq g$ s.t. $op_{D'}(\hat{\pi} \cdot \langle a\rangle)$ satisfies g'. Therefore, there are two plans π_1 to goal g and π_2 to goal g' such that $op_{D'}(\hat{\pi} \cdot \langle a\rangle)$ is a prefix to both $op_{D'}(\pi_1)$ and $op_{D'}(\pi_2)$. Since $op_{D'}(\hat{\pi} \cdot \langle a\rangle) = op_{D'}(\hat{\pi})$, $op_{D'}(\hat{\pi})$ is also a prefix to both $op_{D'}(\pi_1)$ and $op_{D'}(\pi_2)$, which serves to contradict the assumption that $op_{D'}(\hat{\pi})$ satisfies a single goal and $\mathcal{G}_D(\pi) = \mathcal{G}_{D'}(\pi)$. ∎

Next, we define the worst case distinctiveness (*wcd*) measure of a *pogrd* model. We mark the set of non-distinctive paths of a model D as $\Pi_{nd}(D)$ and define the *wcd* as maximal length of a non-distinctive path in the model.

Definition 4 *The* worst case distinctiveness *of a model D, denoted by* $\text{wcd}(D)$ *is:*

$$wcd(D) = \max_{\pi \in \Pi_{nd}(D)} |\pi|$$

An immediate implication of Theorem 1 is that concealed actions may impact the *wcd* only if they immediately follow a non-distinctive prefix. In particular, if a prefix π is non-distinctive and $\pi \cdot \langle a\rangle$ is distinctive, then by concealing a, $op(\pi \cdot \langle a\rangle)$ may increase the number of goals it satisfies, making it non-distinctive.

Calculating *wcd*

To calculate the *wcd* value of the *pogrd* model we compile it into a classical planning problem, where *wcd* is computed for each pair of goals. Individual results are then combined for computing the *wcd* of the entire model. Note that the following description is restricted to the setting where the set of legal plans is the set of optimal plans. The same approach can be applied to the non-optimal setting present by Keren, Gal, and Karpas (2015) with minor modifications, but is omitted here for lack of space.

The *pogrd* problem with two goals is compiled as a single planning problem involving two agents each aiming at a separate goal. The solution to the problem is a plan for each agent and is divided into two parts by a common *exposure point*. The prefix of a plan up to the exposure point represents a non-distinctive path, one that does not reveal the goal of the agent and may include actions performed by

both agents together in addition to non-observable actions performed by a single agent. After the exposure the goal of the agent is recognized. Since our objective is to reveal the *wcd* of the model we discount the cost of actions that belong to the unexposed prefix of the plan thus encouraging the agents to extend the unexposed prefix as much as possible.

Given a *pogrd* problem D and goals $\mathcal{G} = \{g_0, g_1\}$, the *latest-expose* compilation includes two agents: $agent_0$ aiming at g_0 and $agent_1$ aiming at g_1. The model contains three types of actions: *together-actions* (denoted $A^{0,1}$), actions that both agents perform simultaneously, *non-exposed actions* (A^i_{ne}), actions in A_{no} that are executed by a single agent before the *exposure point*, and *exposed actions* (A^i_e), actions (either observed or non-observed) that are performed by a single agent after the *exposure point*. This setting is achieved by adding to the model the *DoExpose* no-cost operation that represents the exposure point by adding to the current state the *exposed* predicate. Actions in $A^{0,1}$ and A^i_{ne} are applicable only before *DoExpose* occurs while actions in A^i_e can be applied only after.

The use of the exposure point is similar to the use of *split* (Keren, Gal, and Karpas 2014; 2015), where agents are encouraged to act together. However, the addition of non-observable actions to the unexposed extends prefix breaks the symmetry that existed in the fully observable setting. The objective is no longer to find a path that maximizes the number of steps both agents share. Rather, one of the agents seeks a maximum path that keeps the agent unrecognized by combining non-observable actions and actions that are on legal paths to a different goal. To reflect this asymmetry we change the objective to allow only one agent (arbitrarily chosen as $agent_0$) to benefit from performing non-observable actions. We let $\Pi_{nd}(g_i)$ represent non-distinctive paths that are prefixes to plans to g_i and define the *wcd-g_i* to be the maximal *wcd* shared by goal g_i and any other goal.

Definition 5 *The* worst case distinctiveness *of a goal g_i in model D, denoted by* wcd-$g_i(D)$ *is:*

$$\text{wcd-g}_i(D) = \max_{\pi \in \Pi_{nd}(g_i)} |\pi|$$

We now present the *latest-expose* compilation for optimal agents.

Definition 6 *For a* pogrd *problem $D = \langle P, \mathcal{G} = \{g_0, g_1\}, \Pi_{leg}(\mathcal{G})\rangle$ where $P = \langle F, I, A = A_o \cup A_{no}\rangle$ we create a planning problem $P' = \langle F', I', A', G'\rangle$, with action costs C', where:*

- $F' = \{f_0, f_1 \mid f \in F\} \cup \{\textit{exposed}\} \cup \{\textit{done}_0\}$
- $I' = \{f_0, f_1 \mid f \in I\}\}$
- $A' = A^{0,1} \cup A^i_{ne} \cup A^i_e \cup \{\textit{DoExpose}\} \cup \{Done_i\}$
  - $A^{0,1} = \{\langle\{f_0, f_1 \mid f \in pre(a)\} \cup \{\neg\textit{exposed}\}, \{f_0, f_1 \mid f \in add(a)\}, \{f_0, f_1 \mid f \in del(a)\}\rangle \mid a \in A\}$
  - $A^i_{ne} = \{\langle\{f_i \mid f \in pre(a)\} \cup \{\neg\textit{exposed}\}, \{f_i \mid f \in add(a)\}, \{f_i \mid f \in del(a)\}\rangle \mid a \in A_{no}\}$
  - $A^0_e = \{\langle\{f_0 \mid f \in pre(a)\} \cup \{\textit{exposed}\} \cup \{\neg\textit{done}_0\}, \{f_0 \mid f \in add(a)\}, \{f_0 \mid f \in del(a)\}\rangle \mid a \in A\}$
  - $A^1_e = \{\langle\{f_1 \mid f \in pre(a)\} \cup \{\textit{exposed}\} \cup \{\textit{done}_0\}, \{f_1 \mid f \in add(a)\}, \{f_1 \mid f \in del(a)\}\rangle \mid a \in A\}$
  - $\textit{Done}_0 = \langle exposed, \textit{done}_0, \emptyset\rangle$
  - $DoExpose = \langle\emptyset, exposed, \emptyset\rangle$
- $G' = \{f_0 | f \in g_0\} \cup \{f_1 | f \in g_1\}$
- $C'(a) = \begin{cases} 2-\epsilon & \textit{if } a \in A^{0,1} \\ 1-\epsilon & \textit{if } a \in A^i_{ne} \\ 1 & \textit{if } a \in A^i_e \\ 0 & \textit{if } a \in \{\textit{DoExpose}\} \cup \{Done_0\} \end{cases}$

f_i is a copy of F for agent i, $exposed$ is a fluent representing the no-cost action *DoExpose* has occurred, and $done_0$ is a fluent indicating the no-cost $Done_0$ has occurred. The initial state is common to both agents and does not include the *exposed* and $done_0$ fluents. Until a *DoExpose* action is performed, the only actions that can be applied are the actions in $A_{0,1}$ and A^i_{ne}. The *DoExpose* action adds $exposed$ to the current state thus allowing the actions in A^0_e to be applied. After agent 0 accomplishes its goal, $Done_0$ is performed, allowing the application of actions in A^1_e until g_1 is achieved. We enforce agent 1 to wait until agent 0 reaches its goal before starting to act in order to make the search for a solution to P' more efficient by removing symmetries between different interleaving of agent plans after $DoExpose$ occurs.

Having described the compilation we now describe its use. In order to find the *wcd* of the model, we solve a different planning problem for each pair of the goals, each time discounting a single agent. The *wcd-g_0* value is found by counting the number of actions performed by $agent_0$ before *DoExpose* occurs. The *wcd* value of the model is the maximal *wcd-g_i* over all solved instances.

Given a solution π to $P^{'}$, we mark the projection of π on each agent i as π_i. π_i includes all actions in $A^{0,1}$ A^i_{ne} and A^i_e that appear in π . Accordingly, the projections of the optimal solution π^* to $P^{'}$ on each agent is marked as π^*_i. In addition, $\pi^*_D(g_i)$ represents an optimal solution of g_i in D. Following the idea presented by Keren, Gal, and Karpas (2014), we guarantee that the projection of the optimal solution to P' yields optimal plans for both agents by bounding ϵ, which represents the discount that may be collected for performing actions before *DoExpose* occurs to be lower than the smallest possible diversion from a legal path to any of the agents.

Lemma 2 *π^*_0 and π^*_1 are optimal plans for each agent in P' if*

$$\epsilon < \frac{1}{|\pi^*_D(g_0)|}$$

Proof: In order to guarantee both agents choose optimal paths we require that the maximal discount that may be collected in $P^{'}$ is smaller than the minimal cost of a diversion from a valid path of any of the agents. We therefore bound the difference between the cost of achieving G' in P' and

achieving g_0 and g_1 in D to be smaller than the cost of the minimal diversion from the optimal paths in D. Therefore, assuming the minimal diversion cost is 1 we require that :

$$C'(\pi^*) - \sum_i (|\pi^*_D(g_i)|) < 1$$

The compilation guarantees that action costs in P and $P^{'}$ differ only in the discount that may be accumulated by $agent_0$ in the unexposed prefix of π^*_0, whose maximal length is equal to *wcd*-$g_0(D)$. We therefore need to ensure that

$$\epsilon \cdot \textit{wcd-}g_0(D) < 1$$

and

$$\epsilon < \frac{1}{\textit{wcd-}g_0(D)}$$

In the worst case $agent_0$ can reach g_0 without being exposed. This guarantees an upper bound on *wcd*-$g_0(D)$ s.t. *wcd*-$g_0(D) \leq |\pi^*(g_0)_D|$. Therefore if

$$\epsilon < \frac{1}{|\pi^*_D(g_0)|}$$

both agents act optimally. ∎

Next, we show that the observable projection of the paths prior to the exposure point is non-distinctive. Given a solution π to the $P^{'}$, for any agent i, $unexposed(\pi_i)$ denotes the prefix of π_i prior to the exposure point.

Lemma 3 *$unexposed(\pi_i)$ is non-distinctive.*

Proof: To show that $unexposed(\pi_i)$ is non-distinctive we need to show that $\exists g, g^{'}$ s.t. $g \neq g^{'}$ and $unexposed(\pi_i)$ satisfies both g and $g^{'}$. The compilation guarantees that for any action $a \in unexposed(\pi_i)$, $a \in A^{0,1}$ or A^i_{no}. According to Definition 1, $op(unexposed(\pi_i)) = \{\langle a_1 ... a_n | a \in A^{0,1} \rangle\}$. This means that $op(unexposed(\pi_0)) = op(unexposed(\pi_1))$ and $op(unexposed(\pi_i))$ contains only observable actions the agents perform together and which therefore appear on the plans to both g_0 and g_1. Therefore, $op(unexposed(\pi_i))$ satisfies more then one goal and it is non-distinctive. ∎

Finally, Theorem 2 shows that the optimal solution to P' yields the *wcd*-g_0, thus concluding our proof of correctness.

Theorem 2 *Given a* pogrd *model D with two goals $\langle g_0, g_1 \rangle$ and a model P', created according to Definition 6,* wcd-$g_0(D) = |unexposed(\pi^*_0)|$.

Proof: Lemma 2 guarantees that, apart from the no-cost operation *DoExpose* and Done_0, the solution to $P^{'}$ consists solely of actions that form a pair of optimal paths to each of the goals. Therefore, among the solutions that comply with this condition, π^* is the one that maximizes the accumulated discount. The compilation guarantees that the only way to accumulate discount is by maximizing the number of actions $agent_0$ performs before the exposure point, therefore π^* is the solution to $P^{'}$ that maximizes $|unexposed(\pi_0)|$. Therefore $|unexposed(\pi^*_0)| =$ *wcd*-$g_0(D)$. ∎

In Example 1 the *wcd* is 7 since when calculating *wcd*-g_0 for an agent aiming at g_0, π^*_0 consists of $agent_0$ loading O_1 at Loc_1, driving together with $agent_1$ to Loc_2, loading O_2, driving together with $agent_1$ to Loc_3, unloading both packages and loading 0_4 before the $DoExpose$ occurs. Note that all these actions belong to either $A^{0,1}$ or $A^0_n e$ as opposed to the unobservable action of unloading O_4 which occurs after the exposure point and therefore belongs to A^0_e and is not part of the *wcd* path.

Reducing *wcd*

Having formulated the *wcd* measure, we turn to our second objective of finding ways to optimize the *wcd* by redesigning the model. Optimization can be achieved using two possible modifications, namely *action removal* and *exposure*. The former reduces *wcd* by disallowing actions from being performed. The latter involves exposing an action by moving it from A^{no} to A^o, *e.g.*, by placing a sensor in an area of the model that was previously non-observed.

wcd reduction is performed within a modification budget that represents the constraints to be respected by the reduction method. Given the two possible modifications of a model, we can either provide an integrated budget, B_{total}, or separate budgets $B_{sep} = \langle B_{rem}, B_{sen} \rangle$, where B_{rem} and B_{sen} are the bounds on the number of actions that can be removed and exposed, respectively.

Our objective is to use the budget constraint to minimize the *wcd* value of the model. We mark the modifications by a pair $\langle A_\neg, A_{no \to o} \rangle$, where $A_\neg$ and $A_{no \to o}$ are the disallowed and exposed actions in the transformed model respectively. In our exploration we assume a uniform cost for the removal and exposure of all actions. In addition, we add the requirement that the cost of achieving any of the goals must not increase. Note that these assumptions are made for simplicity and can be easily relaxed without major modification to the reduction algorithms.

Let $D_{\langle A_\neg, A_{no \to o} \rangle}$ denote the transformed version of D where $A = A \setminus A_\neg$, $A_{no} = A_{no} \setminus A_{no \to o}$ and $A_{obs} = A_{obs} \cup A_{no \to o}$. For $B_{sep} = \langle B_{rem}, B_{sen} \rangle$ the objective can be expressed by the following optimization problem.

$$\begin{aligned}
&\underset{A_\neg \cup A_{no \to o}}{\text{minimize}}\ wcd(D_{\langle A_\neg, A_{no \to o} \rangle}) \\
&\text{subject to} \\
&|A_\neg| \leq B_{rem} \text{ and} \\
&|A_{no \to o}| \leq B_{sen} \text{ and} \\
&\forall g \in \mathcal{G}, C^*_D(g) = C^*_{D_{\langle A_\neg, A_{no \to o} \rangle}}(g)
\end{aligned}$$

where $C^*_D(g)$ and $C^*_{D_{\langle A_\neg, A_{no \to o} \rangle}}(g)$ represent the optimal costs of achieving goal g in D and $D_{\langle A_\neg, A_{no \to o} \rangle}$, respectively. When the budget is integrated the first two constraints are replaced with $|A_\neg| + |A_{no \to o}| \leq B_{total}$.

The reduction is performed using a BFS search that iteratively explores all possible modifications to the model. The

initial state is the original model and each successor node introduces a single modification, either exposure or reduction, that was not included in the parent node. A node in the search tree is therefore represented by a pair $\langle A_{\neg}, A_{no\rightarrow o}\rangle$. A node is pruned from the search if any of the constraints have been violated or if there are no more actions to add.

The key question remaining is what are the modifications that should be considered at each stage. A naïve approach would be to consider all possible modifications, which is unpractical and wasteful. Instead, we focus our attention on modifications that have the potential of reducing the *wcd* by either eliminating the *wcd* path (action removal) or by reducing the length of its non-distinctive prefix (exposure). It was already shown that the only actions that need to be considered for elimination are the ones on the current *wcd* path (Keren, Gal, and Karpas 2014). We show that the only non-observable actions that need to be considered for exposure are the ones that appear on the non-distinctive prefix of the current *wcd* path. We refer to the pair of plans that maximize the *wcd* as the wcd *plans* of a model and mark them by $\Pi_{wcd}(D)$.

Theorem 3 *Let $D_{\langle A_{\neg}, A_{no\rightarrow o}\rangle}$ be a model and $D_{\langle A_{\neg}, A'_{no\rightarrow o}\rangle}$ be a transformed model s.t. $A_{no\rightarrow o} \subseteq A'_{no\rightarrow o}$. If for all $a \in A'_{no\rightarrow o} \setminus A_{no\rightarrow o}$, $a \notin pre_{nd}(\Pi_{wcd}(D_{\langle A_{\neg}, A_{no\rightarrow o}\rangle}))$ then* $\text{wcd}(D_{\langle A_{\neg}, A_{no\rightarrow o}\rangle}) = \text{wcd}(D'_{\langle A_{\neg}, A'_{no\rightarrow o}\rangle})$.

The proof is immediate from Theorem 1.

The reduction algorithm creates, for each node, one successor for disallowing each action that appears in $\Pi_{wcd}(D)$ and one successor for exposing each non-observable action in $pre_{nd}(\Pi_{wcd}(D'))$ of the parent node. To avoid redundant computation, we cache computed actions combination.

In Example 1 disallowing actions is not possible without increasing the optimal costs. However, by exposing $LoadO_1$ (i.e. by placing a sensor on the object), the *wcd* is reduced to 0 equivalently to the fully observable setting.

Empirical Evaluation

Our empirical evaluation has several objectives. Having shown that reduced observability may increase the *wcd* value of a model, we first examine empirically the extent of this effect. In addition, we compare the fully observable setting (Keren, Gal, and Karpas 2014) with the *partially observable* setting. Finally, we evaluate the reduction process as well as the effectiveness of action reduction vs. exposure. We describe the datasets and the experiment setup before presenting and discussing the results.

Datasets We use 4 domains of plan recognition (Ramirez and Geffner 2009), namely GRID-NAVIGATION, IPC-GRID$^+$, BLOCK-WORDS, and LOGISTICS. Each problem description contains a domain description, a template for a problem description without the goal, a set of goals and a set of non-observable actions. For each benchmark we generated a separate *grd* problem for each pair of hypotheses and randomly sampled actions to form the non-observable set creating 3 instances with 5%, 10% and 20% randomly chosen non-observable actions. We tested 216 GRID-NAVIGATION instances, 660 IPC-GRID$^+$ instances, 600 BLOCK-WORDS instances, and 300 LOGISTICS instances. In addition, we created a hand crafted benchmark for the LOGISTICS domain dubbed LOGISTICS-Generated, which corresponds to Example 1 where packages load and unload actions are non-observable. This corresponds to real-world settings where satellite imaging can easily track movement of vehicles between locations, but the actual actions performed are obscured from view.

Setup For each problem instance, we calculated the *wcd* value and run-time for the fully observable and *partially observable* settings. For the *wcd* reduction we examined the *partially observable* setting with 3 bound settings: an integrated bound of $B_{total} = 4$ and 2 separate bounds $B_{sep} = \langle 0, 2\rangle$ and $B = \langle 2, 0\rangle$, where the first element of each pair represents B_{rem} and the second B_{sen}. We used the Fast Downward planning system (Helmert 2006) running A^* with the LM-CUT heuristic (Helmert 2006). The experiments were run on Intel(R) Xeon(R) CPU X5690 machines, with a time limit of 30 minutes and memory limit of 2 GB.

Results Table 1 summarizes the impact the ratio of non-observable actions has on the execution time and the *wcd* . The *partially observable* setting is partitioned into the various ratios examined, including a problem with no non-observable activities, which is compared against the values collected for the fully observable setting solved using *latest-split*. For each setting we compare average run time (in seconds) over solved problems. Whenever some of the problems timed-out, we mark in parenthesis the ratio of solved instances. For all domains, *wcd* increases with the increase in the ratio of non-observable actions. As for running time, the *latest-split* outperforms the equivalent *partially observable* setting for all domains except GRID-NAVIGATION, for which performance is similar. However, the overhead for adding non-observable actions is negligible. For the LOGISTICS-Generated domain the increase in *wcd* was more noticeable, with the average *wcd* increasing from 3.77 in the fully observable setting to 4.87 in the *partially observable* setting.

Table 2 summarizes the results for the *wcd* reduction for the *partially observable* setting for each ratio, showing for each budget allocation the average *wcd* reduction achieved within the allocated time (for the LOGISTICS domain results refer only to the problems that were successfully solved in the *wcd* calculation stage). The evaluation shows that for all domains the *wcd* can be decreased by applying at least one of the modification methods separately, but the most substantial reduction is achieved by combining the methods. This hypothesis is supported by the LOGISTICS-Generated domain where the results for the reduction were from 4.87 in the original partially observable setting to 3.34, 3.9 and 3.8 for the $4, \langle 0, 2\rangle, \langle 2, 0\rangle$ bound allocations respectively.

Related Work

Goal recognition design was first introduced by Keren et al. (2014; 2015), offering tools to analyze and solve the *grd* model in fully observable settings. This work relaxes the full observability assumption.

	latest-split		0%		5%		10%		20%	
	Time	*wcd*	Time	*wcd*	Time	*wcd*	Time	*wcd*	Time	*wcd*
GRID-NAVIGATION	0.324	10.36	0.319	10.36	0.356	10.41	0.351	10.46	0.357	11.1
IPC-GRID$^+$	3.53	3.454	8.754	3.454	9.56	3.55	9.64	3.67	9.96	3.84
BLOCKSWORLD	3.03	2.06	29.2	2.06	33.2	2.12	25.89	2.14	31.01	2.82
LOGISTICS	238.497 (0.9)	3.51	165.2 (0.6)	3.71	153.26 (0.31)	3.71	155.48 (0.29)	3.78	191.56 (0.2)	4.1

Table 1: Average running time for *wcd* calculation over solved problems for varying non-observable actions ratio

	5				10				20			
	0	4	2:0	0:2	0	4	2:0	0:2	0	4	2:0	0:2
GRID-NAVIGATION	10.41	9.64	9.71	10.36	10.46	9.34	9.76	10.36	11.1	10.91	11.1	10.91
IPC-GRID$^+$	3.55	2.01	2.01	3.55	3.67	1.75	1.87	2.93	3.84	2.6	2.92	3.35
BLOCKSWORLD	2.12	1.78	1.83	2.12	2.14	1.58	1.64	2.1	2.82	2.15	2.45	2.67
LOGISTICS	3.71	3.37	3.44	3.56	3.78	3.26	3.42	3.51	4.1	3.47	3.8	3.67

Table 2: Average *wcd* after reduction for each ratio and budget allocation achieved within allocated time

The first to establish the connection between the closely related fields of automated planning and goal recognition were Ramirez and Geffner (2009), presenting a compilation of plan recognition problems into classical planning problems that can be solved by any planner. Several works on plan recognition followed this approach (Agotnes 2010; Pattison and Long 2011; Ramirez and Geffner 2010; 2011) by using various automated planning techniques. We follow this approach and introduce a novel compilation of goal recognition design problems with non observable actions into classical planning.

Partial observability in goal recognition has been modeled in various ways (Ramirez and Geffner 2011; Geib and Goldman 2005; Avrahami-Zilberbrand, Kaminka, and Zarosim 2005). In particular, observability can be modeled using a sensor model that includes an observation token for each action (Geffner and Bonet 2013). Note that the *pogrd* model presented for the *partially observable* setting, can be thought of one in which the set of observation tokens O includes an empty observation sequence $o_\emptyset$ and A includes a no-cost action a_{idle} by which an agent remains at his current position.

Conclusions

We presented a model for goal recognition design that accounts for partial observability by partitioning of the set of actions to observable and non-observable actions. We extend the *wcd* measure and proposed ways to calculate and reduce it. By accounting for non-observable actions, we increase the model's relevancy to a wide range of real-world settings.

Our empirical evaluation shows that non-observable actions typically increases the *wcd* value. In addition, we showed that for all of the domains, *wcd* reduction by combining disallowed and exposed actions is preferred over each of the methods separately.

In future work we intend to investigate alternative ways to account for partial observability by creating more elaborated sensor models.

References

Agotnes, T. 2010. Domain independent goal recognition. In *Stairs 2010: Proceedings of the Fifth Starting AI Researchers Symposium*, volume 222, 238. IOS Press, Incorporated.

Avrahami-Zilberbrand, D.; Kaminka, G.; and Zarosim, H. 2005. Fast and complete symbolic plan recognition: Allowing for duration, interleaved execution, and lossy observations. In *Proc. of the AAAI Workshop on Modeling Others from Observations, MOO*.

Boddy, M. S.; Gohde, J.; Haigh, T.; and Harp, S. A. 2005. Course of action generation for cyber security using classical planning. In *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS 2005)*, 12–21.

Cohen, P. R.; Perrault, C. R.; and Allen, J. F. 1981. Beyond question-answering. Technical report, DTIC Document.

Fikes, R. E., and Nilsson, N. J. 1972. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence* 2(3):189–208.

Geffner, H., and Bonet, B. 2013. A concise introduction to models and methods for automated planning. *Synthesis Lectures on Artificial Intelligence and Machine Learning* 8(1):1–141.

Geib, C. W., and Goldman, R. P. 2005. Partial observability and probabilistic plan/goal recognition. In *Proceedings of the International workshop on modeling other agents from observations (MOO-05)*.

Helmert, M. 2006. The fast downward planning system. *J. Artif. Intell. Res. (JAIR)* 26:191–246.

Hong, J. 2001. Goal recognition through goal graph analysis. *Journal of Artificial Intelligence Research(JAIR 2001)* 15:1–30.

Jarvis, P. A.; Lunt, T. F.; and Myers, K. L. 2004. Identifying terrorist activity with ai plan recognition technology. In *Proceedings of the Sixteenth National Conference on Innovative Applications of Artificial Intelligence (IAAI 2004*, 858–863. AAAI Press.

Kaluza, B.; Kaminka, G. A.; and Tambe, M. 2011. Towards detection of suspicious behavior from multiple observations. In *AAAI Workshop on Plan, Activity, and Intent Recognition (PAIR 2011)*.

Kautz, H., and Allen, J. F. 1986. Generalized plan recognition. In *Proceedings of the Fifth National Conference of the Amercan Association of Artificial Intelligence (AAAI 1986)*, volume 86, 32–37.

Kautz, H.; Etzioni, O.; Fox, D.; Weld, D.; and Shastri, L.

2003. Foundations of assisted cognition systems. *University of Washington, Computer Science Department, Technical Report*.

Keren, S.; Gal, A.; and Karpas, E. 2014. Goal recognition design. In *ICAPS Conference Proceedings*.

Keren, S.; Gal, A.; and Karpas, E. 2015. Goal recognition design for non optimal agents. In *Proceedings of the Conference of the American Association of Artificial Intelligence (AAAI 2015)*.

Lesh, N., and Etzioni, O. 1995. A sound and fast goal recognizer. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI 1995)*, volume 95, 1704–1710.

Pattison, D., and Long, D. 2011. Accurately determining intermediate and terminal plan states using bayesian goal recognition. *Proceedings of the First Workshop on Goal, Activity and Plan Recognition(GAPRec 2011)* 32.

Ramirez, M., and Geffner, H. 2009. Plan recognition as planning. In *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI 2009)*.

Ramirez, M., and Geffner, H. 2010. Probabilistic plan recognition using off-the-shelf classical planners. In *Proceedings of the Conference of the American Association of Artificial Intelligence (AAAI 2010)*.

Ramirez, M., and Geffner, H. 2011. Goal recognition over pomdps: Inferring the intention of a pomdp agent. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence- Volume Three (IJCAI 2011)*, 2009–2014. AAAI Press.

Classical Planning with Simulators: Results on the Atari Video Games

Nir Lipovetzky
The University of Melbourne
Melbourne, Australia
nir.lipovetzky@unimelb.edu.au

Miquel Ramirez
Australian National University
Canberra, Australia
miquel.ramirez@anu.edu.au

Hector Geffner
ICREA & U. Pompeu Fabra
Barcelona, Spain
hector.geffner@upf.edu

Abstract

The Atari 2600 games supported in the Arcade Learning Environment (Bellemare *et al.* 2013) all feature a known initial (RAM) state and actions that have deterministic effects. Classical planners, however, cannot be used off-the-shelf as there is no compact PDDL-model of the games, and action effects and goals are not known *a priori*. Indeed, there are no explicit goals, and the planner must select actions on-line while interacting with a simulator that returns successor states and rewards. None of this precludes the use of blind lookahead algorithms for action selection like breadth-first search or Dijkstra's yet such methods are not effective over large state spaces. We thus turn to a different class of planning methods introduced recently that have been shown to be effective for solving large planning problems but which do not require prior knowledge of state transitions, costs (rewards) or goals. The empirical results over 54 Atari games show that the simplest such algorithm performs at the level of UCT, the state-of-the-art planning method in this domain, and suggest the potential of width-based methods for planning with simulators when factored, compact action models are not available.

A version of this short paper has been accepted at IJCAI (Lipovetzky et al. *2015)*

Introduction

The Arcade Learning Environment (ALE) provides a challenging platform for evaluating general, domain-independent AI planners and learners through a convenient interface to hundreds of Atari 2600 games (Bellemare *et al.* 2013). Results have been reported so far for basic planning algorithms like breadth-first search and UCT, reinforcement learning algorithms, and evolutionary methods (Bellemare *et al.* 2013; Mnih *et al.* 2013; Hausknecht *et al.* 2014). The empirical results are impressive in some cases, yet a lot remains to be done, as no method approaches the performance of human players across a broad range of games.

While all these games feature a known initial (RAM) state and actions that have deterministic effects, the problem of selecting the next action to be done cannot be addressed with off-the-shelf classical planners (Ghallab *et al.* 2004; Geffner and Bonet 2013). This is because there is no compact PDDL-like encoding of the domain and the goal to be achieved in each game is not given, precluding the automatic derivation of heuristic functions and other inferences. Indeed, there are no goals but rewards, and the planner must select actions on-line while interacting with a simulator that just returns successor states and rewards.

The action selection problem in the Atari games can be addressed as a *reinforcement learning* problem (Sutton and Barto 1998) over a deterministic MDP where the state transitions and rewards are not known, or alternatively, as a *net-benefit planning problem* (Coles *et al.* 2012; Keyder and Geffner 2009) with unknown state transitions and rewards. ALE supports the two settings: an *on-line plannning setting* where actions are selected after a lookahead, and a *learning setting* that must produce controllers for mapping states into actions reactively without any lookahead. In this work, we are interested in the on-line planning setting.

The presence of unknown transition and rewards in the Atari games does not preclude the use of blind-search methods like breadth-first search, Dijkstra's algorithm (Dijkstra 1959), or learning methods such as LRTA* (Korf 1990), UCT (Kocsis and Szepesvári 2006), and Q-learning (Sutton and Barto 1998; Bertsekas and Tsitsiklis 1996). Indeed, the net-benefit planning problem with unknown state transitions and rewards over a given planning horizon, can be mapped into a standard *shortest-path problem* which can be solved optimally by Dijkstra's algorithm. For this, we just need to map the unknown rewards $r(a, s)$ into positive (unknown) action costs $c(a, s) = C - r(a, s)$ where C is a large constant that exceeds the maximum possible reward. The fact that the state transition and cost functions $f(a, s)$ and $c(a, s)$ are not known a priori doesn't affect the applicability of Dijkstra's algorithm, which requires the value of these functions precisely when the action a is applied in the state s.

The limitation of the basic blind search methods is that they are not effective over large state spaces, neither for solving problems off-line, nor for guiding a lookahead search for solving problems on-line. In this work, we thus turn to a recent class of planning algorithms that combine the scope of blind search methods with the performance of state-of-the-art classical planners: namely, like "blind" search algorithms they do not require prior knowledge of state transitions, costs, or goals, and yet like heuristic algorithms they

manage to search large state spaces effectively. The basic algorithm in this class is called IW for Iterated Width search (Lipovetzky and Geffner 2012). IW consists of a sequence of calls IW(1), IW(2), .., IW(k), where IW(i) is a standard breadth-first search where states are pruned right away when they fail to make true some new tuple (set) of at most i atoms. Namely, IW(1) is a breadth-first search that keeps a state only if the state is the first one in the search to make some atom true; IW(2) keeps a state only if the state is the first one to make a pair of atoms true, and so on. Like plain breadth-first and iterative deepening searches, IW is complete, while searching the state space in a way that makes use of the *structure of states* given by the values of a finite set of state variables. In the Atari games, the (RAM) state is given by a vector of 128 bytes, which we associate with 128 variables X_i, $i = 1, \ldots, 128$, each of which may take up to 256 values x_j. A state s makes an atom $X_i = x_j$ true when the value of the i-th byte in the state vector s is x_j. The empirical results over 54 Atari games show that IW(1) performs at the level of UCT, the state-of-the-art planning method in this domain, and suggest the potential of width-based methods for planning with simulators when factored, compact action models are not available.

The paper is organized as follows. We review the iterated width algorithm and its properties, look at the variations of the algorithm that we used in the Atari games, and present the experimental results.

Iterated Width

The Iterated Width (IW) algorithm has been introduced as a classical planning algorithm that takes a planning problem as an input, and computes an action sequence that solves the problem as the output (Lipovetzky and Geffner 2012). The algorithm however applies to a broader range of problems. We will characterize such problems by means of a finite and discrete set of states (the state space) that correspond to vectors of size n. Namely, the states are *structured* or *factored* , and we take each of the locations in the vector to represent a variable X_i, and the value at that vector location to represent the value x_j of variable X_i in the state. In addition to the state space, a problem is defined by an initial state s_0, a set of actions applicable in each state, a transition function f such that $s' = f(a, s)$ is the state that results from applying action a to the state s, and rewards $r(a, s)$ represented by real numbers that result from applying action a in state s. The transition and reward functions do not need to be known *a priori*, yet in that case, the state and reward that results from the application of an action in a state need to be *observable*. The task is to compute an action sequence $a_0, \ldots, a_m$ for a large horizon m that generates a state sequence $s_0, \ldots, s_{m+1}$ that maximizes the accumulated reward $\sum_{i=0}^{m} r(a_i, s_i)$, or that provides a good approximation.

The Algorithm

IW consists of a sequence of calls IW(i) for $i = 0, 1, 2, \ldots$ over a problem P until a termination condition is reached. The procedure IW(i) is a plain forward-state *breadth-first search* with just one change: right after a state s is generated, the state is pruned if it doesn't pass a simple *novelty test*. More precisely,

- The *novelty of a newly generate state s in a search algorithm* is 1 if s is the first state generated in the search that makes true some atom $X = x$, else it is 2 if s is the first state that makes a *pair* of atoms $X = x$ and $Y = y$ true, and so on.
- IW(i) is a breadth-first search that prunes newly generated states when their novelty is greater than i.
- IW calls IW(i) sequentially for $i = 1, 2, \ldots$ until a termination condition is reached, returning then the best path found.

For classical planning, the termination condition is the achievement of the goal. In the *on-line setting,* as in the Atari games, the termination condition is given by a time window or a maximum number of generated nodes. The best path found by IW is then the path that has a maximum accumulated reward. The accumulated reward $R(s)$ of a state s reached in an iteration of IW is determined by the unique parent state s' and action a leading to s from s' as $R(s) = R(s') + r(a, s')$. The best state is the state s with maximum reward $R(s)$ generated but not pruned by IW, and the best path is the one that leads to the state s from the current state. The action selected in the on-line setting is the first action along such a path. This action is then executed and the process repeats from the resulting state.

Performance and Width

IW is a systematic and complete blind-search algorithm like breadth-first search (BRFS) and iterative deepening (ID), but unlike these algorithms, it uses the factored representation of the states in terms of variables to structure the search. This structured exploration has proved to be very effective over classical planning benchmark domains when goals are single atoms.[1] For example, 37% of the 37,921 problems considered in (Lipovetzky and Geffner 2012) are solved by IW(1) while 51.3% are solved by IW(2). These are instances obtained from 37 benchmark domains by splitting problems with N atomic goals into N problems with one atomic goal each. Since IW(k) runs in time that is exponential in k, this means that almost 90% of the 37,921 instances are solved in time that is either linear or quadratic in the number of problem variables, which in such encodings are all *boolean*. Furthermore, when the performance of IW is compared with a Greedy Best First Search guided by the additive heuristic h_{add}, it turns out that "blind" IW solves as many problems as the informed search, 34,627 vs. 34,849, far ahead of other blind search algorithms like BRFS and ID that solve 9,010 and 8,762 problems each. Moreover, IW is faster and results in shorter plans than in the heuristic search.

The min k value for which IW(k) solves a problem is indeed bounded and small in most of these instances. This

[1] Any conjunctive goal can be mapped into a single dummy atomic goal by adding an action that achieves the dummy goal and that has the original conjunctive goal as a precondition. Yet, this mapping changes the definition of the domain.

is actually no accident and has a *theoretical explanation.* Lipovetzky and Geffner define a structural parameter called the problem *width* and show that for many of these domains, *any* solvable instance with atomic goals will have a bounded and small width that is independent of the number of variables and states in the problem. The min value k for which the iteration IW(k) solves the problem cannot exceed the problem width, so the algorithm IW *runs in time and space that are exponential in the problem width.*

Formally, the *width* $w(P)$ of a problem P is i iff i is the minimum positive integer for which there is a sequence $t_0, t_1, \ldots, t_n$ of atom sets t_k with at most i atoms each, such that 1) t_0 is true in the initial state of P, 2) any shortest plan π that achieves t_k in P can be extended into a shortest plan that achieves t_{k+1} by extending π with one action, and 3) any shortest plan that achieves t_n is a shortest plan for achieving the goal of P.

While this notion of width and the iterated width algorithms that are based on it have been designed for problems where a goal state needs to be reached, the notions remain relevant in optimization problems as well. Indeed, if a good path is made of states s_i each of which has a low width, IW can be made to find such path in low polynomial time for a small value of the k parameter. Later on we will discuss a slight change required in IW to enforce this property.

The Algorithms for the Atari Games

The number of nodes *generated* by IW(1) is $n \times D \times b$ in the worst case, where n is the number of problem variables, D is the size of their domains, and b is the number of actions per state. This same number in a breadth-first search is not linear in n but exponential. For the Atari games, $n = 128$, $D = 256$, and $b = 18$, so that the product is equal to $589,824$, which is large but feasible. On the other hand, the number of nodes generated by IW(2) in the worst case is $(n \times D)^2 \times b$, which is equal to $19,327,352,832$ which is too large, forcing us to consider only a tiny fraction of such states. For classical planning problems, the growth in the number of nodes from IW(1) to IW(2) is not that large, as the variables are boolean. Indeed, we could have taken the state vector for the Atari games as a vector of 1024 boolean variables, and apply these algorithms to that representation. The number of atoms would indeed be much smaller, and both IW(1) and IW(2) would run faster then. However by ignoring the correlations among bits in each one of the 128 words, the results would be weaker.

IW is a purely exploration algorithm that does not take into account the accumulated reward for selecting the states to consider. As a simple variant that combines exploration and exploitation, we evaluated a *best-first search* algorithm with two queues: one queue ordered first by novelty measure (recall that novelty one means that the state is the first one to make some atom true), and a second queue ordered by accumulated reward. In one iteration, the best first search picks up the best node from one queue, and in the second iteration it picks up the best node from the other queue. This way for combining multiple heuristics is used in the LAMA planner (Richter and Westphal 2010), and was introduced in the planner Fast Downward (Helmert 2006). In addition, we break ties in the first queue favoring states with largest accumulated reward, and in the second queue, favoring states with smallest novelty measure. Last, when a node is expanded, it is removed from the queue, and its children are placed on both queues. The exception are the nodes with no accumulated reward that are placed in the first queue only. We refer to this best-first algorithm as 2BFS.

For the experiments below, we added two simple variations to IW(1) and 2BFS. First, in the breadth-first search underlying IW(1), we generate the children in random order. This makes the executions that result from the IW(1) lookahead less susceptible to be trapped into loops; a potential problem in local search algorithms with no memory or learning. Second, a discount factor $\gamma = 0.995$ is used in both algorithms for discounting future rewards like in UCT. For this, each state s keeps its depth $d(s)$ in the search tree, and if state s' is the child of state s and action a, $R(s')$ is set to $R(s) + \gamma^{d(s)+1} r(a, s)$. The discount factor results in a slight preference for rewards that can be reached earlier, which is a reasonable heuristic in on-line settings based on lookahead searches.

Experimental Results

We tested IW(1) and 2BFS over 54 of the 55 different games considered in (Bellemare *et al.* 2013), from now on abbreviated as BNVB.[2] The two algorithms were used to play the games in the *on-line planning setting* supported by ALE where we will compare them with the planning algorithms considered by BNVB; namely, breadth-first search and UCT. ALE supports also a *learning setting* where the goal is to learn controllers that map states into actions without doing any lookahead. Algorithms across the two settings are thus not directly comparable as they compute different things. Learning controllers appears as a more challenging problem and it is thus not surprising that planning algorithms like UCT tend to achieve a higher score than learning algorithms. In addition, the learning algorithms reported by BNVB tend to use the state of the screen pixels, while the planning algorithms, use the state of the RAM memory. It is not clear however whether the use of one input representation is more challenging than the use of the other. For the learning algorithms, BNVB mention that the results tend to be better for the screen inputs. Experiments were run on a cluster, where each computing node consists of a 6-core Intel Xeon E5-2440, with 2.4 GHz clock speed, with 64 GBytes of RAM installed.

Table 1 shows the performance of IW(1) and 2BFS in comparison with breadth-first search (BRFS) and UCT. Videos of selected games played by IW(1), 2BFS, and UCT can be seen in Youtube.[3] The discount factor used by all the algorithms is $\gamma = 0.995$. The scores reported for BRFS and UCT are taken from BNVB. Our experimental setup follows theirs except that a maximum budget of $150,000$ simulated frames is applied to IW(1), 2BFS,

[2] We left out SKIING as the reported figures apparently use a different reward structure.

[3] `http://www.youtube.com/playlist?list=PLXpQcXUQ_CwenUazUivhXyYvjuS6KQOI0`.

and UCT. UCT uses this budget by running 500 rollouts of depth 300. The bound on the number of simulated frames is like a bound on lookahead time, as most of the time in the lookahead is spent in calls to the emulator for computing the next RAM state. This is why the average time per action is similar to all the algorithms except IW(1), that due to its pruning does not always use the full budget and takes less time per action on average.

Also, as reported by BNVB, all of the algorithms reuse the frames in the sub-tree of the previous lookahead that is rooted in the selected child, deleting its siblings and their descendants. More precisely, no calls to the emulator are done for transitions that are cached in that sub-tree, and such reused frames are not discounted from the budget that is thus a bound on the number of *new* frames per lookahead. In addition, in IW(1), the states that are reused from the previous searches are ignored in the computation of the novelty of new states so that more states can escape pruning. Otherwise, IW(1) often uses a fraction of the budget. This is not needed in 2BFS which does no pruning. IW(1) and 2BFS are limited to search up to a depth of $1,500$ frames and up to $150,000$ frames per root branch. This is to avoid the search from going too deep or being too committed to a single root action.

Last, in the lookahead, IW(1) and 2BFS select an action every 5 frames, while UCT selects an action every frame. This means that in order to explore a branch 300 frames deep, UCT gets to choose 300 actions, while IW(1) and 2BFS get to choose 60 actions, both however using the same 300 frames from the budget. For this, we followed the setup of BRFS in BNVB that also selects actions every 5 frames, matching the behavior of the emulator that requests an action also every 5 frames. Since the lookahead budget is given by a maximum number of (new) frames, and the time is mostly taken by calls to the emulator, this may not be the best choice for IW(1) and 2BFS that may therefore not be exploiting all the options afforded by the budget. Interestingly, when UCT is limited to one action every 5 frames, its performance is reduced by up to a 50% in games where it performs very well (CRAZY CLIMBER), and does not appear to improve in those games where it performs very poorly (FREEWAY).

Table 1 shows that both IW(1) and 2BFS outperform BRFS, which rarely collects reward in many domains as the depth of the BRFS search tree results in a lookahead of 0.3 seconds (20 frames or 4 nodes deep). The notable exception to this is CENTIPEDE where abundant reward can be collected with a shallow lookahead. On the other hand, both IW(1) and 2BFS normally reach states that are up to 350–1500 frames deep (70–260 nodes or 6–22 seconds), even if IW(1) does not always use all the simulation frames allocated due to its agressive pruning. This can be observed in games such as BREAKOUT, CRAZY CLIMBER, KANGAROO, and POOYAN, where the average CPU time for each lookahead is up to 10 times faster than 2BFS. Computation time for UCT and BRFS are similar to 2BFS, as the most expensive part of the computation is the generation of frames through the simulator, and these three algorithms always use the full budget.

More interestingly, IW(1) outscores UCT in 31 of the 54 games, while 2BFS outscores UCT in 26. On the other hand, UCT does better than IW(1) and 2BFS in 19 and 25 games respectively. The relative performance between IW(1) and 2BFS makes IW(1) the best of the two in 34 games, and 2BFS in 16. In terms of the number of games where an algorithm is the best, IW(1) is the best in 26 games, 2BFS in 13 games, and UCT in 19 games. Also, BRFS is best in 2 games (CENTIPEDE, tied up in BOXING), while the other three algorithms are tied in another 2 games (PONG, BOXING).

Likewise, in FREEWAY and BERZERK both IW(1) and 2BFS attain a better score than the baseline semi-random algorithm *Perturb* in (Bellemare *et al.* 2013), that beats UCT on those games. *Perturb* is a simple algorithm that selects a fixed action with probability 0.95, and a random action with probability 0.05. For *Perturb*, BNVB do not report the average score but the best score. *Perturb* manages to do well in domains where rewards are deep but can be reached by repeating the same action. This is the case of FREEWAY, where a chicken has to run to the top of the screen across a ten lane highway filled with traffic. Every time the chicken gets across (starting at the bottom), there is one unit of reward. If the chicken is hit by a car, it goes back some lanes. In FREEWAY, only 12 out of the 18 possible actions have an effect: 6 actions move the chicken up (up-right, up-left, up-fire, up-right-fire, up-left-fire), 6 actions move the chicken down (down-right, down-left, down-fire, down-right-fire, down-left-fire), and 6 actions do nothing. *Perturb* does well in this domain when the selected fixed action moves the chicken up. As noted in Table 1 and seen in the provided video, UCT does not manage to take the chicken across the highway at all. The reason that UCT does not collect any reward is that it needs to move the chicken up at least 240 times[4] something that is very unlikely in a random exploration. IW(1) does not have this limitation and is best in FREEWAY.

IW(1) obtains better scores than the best learning algorithm (Mnih *et al.* 2013) in the 7 games considered there, and 2BFS does so in 6 of the 7 games. Comparing with the scores reported for the reinforcement learning algorithms in BNVB, we note that both IW(1) and 2BFS do much better than the best learning algorithm in those games where the learning algorithms outperform UCT namely, MONTEZUMA REVENGE, VENTURE and BOWLING. We take this as evidence that IW(1) and 2BFS are as at least as good as learning algorithms at finding rewards in games where UCT is not very effective.

For instance, in MONTEZUMA REVENGE rewards are very sparse, deep, and most of the actions lead to losing a life with no immediate penalty or consequence. In our experiments, all algorithms achieve 0 score, except for 2BFS that achieves an average score of 540, and a score of $2,500$ in one of the runs. This means however that even 2BFS is not able to consistently find rewards in this game. This game and several others like BREAKOUT and SPACE INVADERS could be much simpler by adding negative rewards for losing a life. We have indeed observed that our planning algo-

[4]One needs to move the chicken up for at least 4 seconds (240 frames) in order to get it across the highway.

	IW(1)		2BFS		BRFS	UCT
Game	Score	Time	Score	Time	Score	Score
ALIEN	**25634**	81	12252	81	784	7785
AMIDAR	**1377**	28	1090	37	5	180
ASSAULT	953	18	827	25	414	**1512**
ASTERIX	153400	24	77200	27	2136	**290700**
ASTEROIDS	**51338**	66	22168	65	3127	4661
ATLANTIS	159420	13	154180	71	30460	**193858**
BANK HEIST	**717**	39	362	64	22	498
BATTLE ZONE	11600	86	**330800**	87	6313	70333
BEAM RIDER	9108	23	**9298**	29	694	6625
BERZERK	**2096**	58	802	73	195	554
BOWLING	**69**	10	50	60	26	25
BOXING	**100**	15	**100**	22	**100**	**100**
BREAKOUT	384	4	**772**	39	1	364
CARNIVAL	**6372**	16	5516	53	950	5132
CENTIPEDE	99207	39	94236	67	**125123**	110422
CHOPPER COMMAND	10980	76	27220	73	1827	**34019**
CRAZY CLIMBER	36160	4	36940	58	37110	**98172**
DEMON ATTACK	20116	33	16025	41	443	**28159**
DOUBLE DUNK	-14	41	21	41	-19	**24**
ELEVATOR ACTION	13480	26	10820	27	730	**18100**
ENDURO	**500**	66	359	38	1	286
FISHING DERBY	30	39	6	62	-92	**38**
FREEWAY	**31**	32	23	61	0	0
FROSTBITE	902	12	**2672**	38	137	271
GOPHER	18256	19	15808	53	1019	**20560**
GRAVITAR	3920	62	**5980**	62	395	2850
HERO	**12985**	37	11524	69	1324	12860
ICE HOCKEY	**55**	89	49	89	-9	39
JAMES BOND	**23070**	0	10080	30	25	330
JOURNEY ESCAPE	40080	38	**40600**	67	1327	7683
KANGAROO	**8760**	8	5320	31	90	1990
KRULL	**6030**	28	4884	42	3089	5037
KUNG FU MASTER	**63780**	21	42180	43	12127	48855
MONTEZUMA REVENGE	0	14	**540**	39	0	0
MS PACMAN	21695	21	18927	23	1709	**22336**
NAME THIS GAME	9354	14	8304	25	5699	**15410**
PONG	**21**	17	**21**	35	-21	**21**
POOYAN	11225	8	10760	16	910	**17763**
PRIVATE EYE	-99	18	**2544**	44	58	100
Q*BERT	3705	11	11680	35	133	**17343**
RIVERRAID	**5694**	18	5062	37	2179	4449
ROAD RUNNER	**94940**	25	68500	41	245	38725
ROBOT TANK	**68**	34	52	34	2	50
SEAQUEST	**14272**	25	6138	33	288	5132
SPACE INVADERS	2877	21	**3974**	34	112	2718
STAR GUNNER	1540	19	**4660**	18	1345	1207
TENNIS	**24**	21	**24**	36	-24	3
TIME PILOT	35000	9	36180	29	4064	**63855**
TUTANKHAM	172	15	204	34	64	**226**
UP AND DOWN	**110036**	12	54820	14	746	74474
VENTURE	**1200**	22	980	35	0	0
VIDEO PINBALL	**388712**	43	62075	43	55567	254748
WIZARD OF WOR	**121060**	25	81500	27	3309	105500
ZAXXON	**29240**	34	15680	31	0	22610

# Times Best (54 games)	**26**	13	1	19
# Times Better than IW (54 games)	–	16	1	**19**
# Times Better than 2BFS (54 games)	**34**	–	1	25
# Times Better than UCT (54 games)	**31**	26	1	–

Table 1: Performance that results from various lookahead algorithms in 54 Atari 2600 games. The algorithms, BRFS, IW(1), 2BFS, and UCT, are evaluated over 10 runs (episodes) for each game. The maximum episode duration is 18,000 frames and every algorithm is limited to a lookahead budget of 150,000 simulated frames. Figures for BRFS and UCT taken from Bellemare et al. Average CPU times per action in seconds, rounded to nearest integer, shown for IW(1) and 2BFS. Numbers in bold show best performer in terms of average score, while numbers shaded in light grey show scores that are better than UCT's. Bottom part of the table shows pairwise comparisons among the algorithms.

rithms do not care much about losing lives until there is just one life left, when their play noticeably improves. This can be seen in the videos mentioned above, and suggest a simple form of learning that would be useful to both planners and reinforcement learning algorithms.

We are not reporting the performance of IW(k) with parameter $k = 2$ because in our preliminary tests and according to the discussion in the previous section, it doesn't appear to improve much on BRFS, even if it results in a lookahead that is 5 times deeper, but still too shallow to compete with the other planning algorithms.

Exploration and Exploitation

The notion of width underlying the iterated width algorithm was developed in the context of classical planning in order to understand why most of the hundreds of existing benchmarks appear to be relatively simple for current planners, even though classical planning is PSPACE-complete (Bylander 1994). A partial answer is that most of these domains have a low width, and hence, can be solved in low polynomial time (by IW) when goals contain a single atom. Benchmark problems with multiple atomic goals tend to be easy too, as the goals can often be achieved one at a time after a simple goal ordering (Lipovetzky and Geffner 2012).

In the iterated width algorithm, the key notion is the *novelty measure* of a state in the underlying breadth-first search. These novelty measures make use of the factored representation of the states for providing a structure to the search: states that have width 1 are explored first in linear time, then states that have width 2 are explored in quadratic time, and so on. In classical planning problems with atomic goals, this way of organizing the search pays off both theoretically and practically.

The use of "novelty measures" for guiding an optimization search while ignoring the function that is being optimized is common to the novelty-based search methods developed independently in the context of genetic algorithms (Lehman and Stanley 2011). In these methods individuals in the population are not ranked according to the optimization function but in terms of how "novel" they are in relation to the rest of the population, thus encouraging diversity and exploration rather than (greedy) exploitation. The actual definition of novelty in such a case is domain-dependent; for example, in the evolution of a controller for guiding a robot in a maze, an individual controller will not be ranked by how close it takes the robot to the goal (the greedy measure), but by the distance between the locations that are reachable with it, and the locations reachable with the other controllers in the population (a diversity measure). The novelty measure used by IW, on the other hand, is domain-independent and it is determined by the structure of the states as captured by the problem variables.

The balance between exploration and exploitation has received considerable attention in reinforcement learning (Sutton and Barto 1998), where both are required for converging to an optimal behavior. In the Atari games, as in other deterministic problems, however, "exploration" is not needed for optimality purposes, but just for improving the effectiveness of the lookahead search. Indeed, a best-first search algorithm guided only by (discounted) accumulated reward will deliver eventually best moves, but it will not be as effective over small time windows, where like breadth-first search it's likely not to find any rewards at all. The UCT algorithm provides a method for balancing exploration and exploitation, which is effective over small time windows. The 2BFS algorithm above with two queues that alternate, one guided by the novelty measures and the other by the accumulated reward, provides a different scheme. The first converges to the optimal behavior asymptotically; the second in a bounded number of steps, with the caveat below.

Duplicates and Optimality

The notions of width and the IW algorithm guarantee that states with low width will be generated in low polynomial time through *shortest* paths. In the presence of rewards like the Atari games, however, the interest is not in the shortest paths but in the *best* paths; i.e, the paths with maximum reward. IW may actually fail to find such paths even when calling IW(k) with a high k parameter. Optimality could be achieved by replacing the breadth-first search underlying IW(k) by Dijkstra's algorithm yet such a move would make the relation between IW and the notion of width less clear. A better option is to change IW to comply with a different property; namely, that if there is a "rewarding" path made up of states of low width, then IW will find such paths or better ones in time that is exponential in their width. For this, a simple change in IW suffices: when generating a state s that is a *duplicate* of a state s' that has been previously generated and not pruned, s' is replaced by s if $R(s) > R(s')$, with the change of reward propagated to the descendants of s' that are in memory. This is similar to the change required in the A* search algorithm for preserving optimality when moving from consistent to inconsistent heuristics (Pearl 1983). The alternative is to "reopen" such nodes. The same change is actually needed in 2BFS to ensure that, if given enough time, 2BFS will actually find optimal paths. The code used for IW and 2BFS in the experiments above does not implement this change as the overhead involved in checking for duplicates in some test cases did not appear to pay off. More experiments however are needed to find out if this is actually the most effective option.

Summary

We have shown experimentally that width-based algorithms like IW(1) and 2BFS that originate in work in classical planning, can be used to play the Atari video games where they achieve state-of-the-art performance. The results also suggest more generally the potential of width-based methods for planning with simulators when factored, compact action models are not available. In this sense, the scope of these planning methods is broader than those of heuristic-search planning methods that require propositional encodings of actions and goals, and with suitable extensions, may potentially approach the scope of MCTS methods like UCT that work on simulators as well.

Acknowledgments

We thank the people who created the ALE. Nir Lipovetzky is partially funded by the ARC Linkage grant LP11010015.

References

Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47(47):253–279, 2013.

D. Bertsekas and J. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

T. Bylander. The computational complexity of STRIPS planning. *Artificial Intelligence*, 69:165–204, 1994.

Amanda Coles, Andrew Coles, Angel García Olaya, Sergio Jiménez, Carlos Linares López, Scott Sanner, and Sungwook Yoon. A survey of the seventh international planning competition. *AI Magazine*, 33(1):83–88, 2012.

Edsger. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

H. Geffner and B. Bonet. *A Concise Introduction to Models and Methods for Automated Planning*. Morgan & Claypool Publishers, 2013.

M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: theory and practice*. Morgan Kaufmann, 2004.

M. Hausknecht, J. Lehman, Risto Miikkulainen, and Peter Stone. A neuroevolution approach to general atari game playing. *IEEE Transaction on Computational Intelligence and AI in Games*, (99), 2014.

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

E. Keyder and H. Geffner. Soft goals can be compiled away. *Journal of Artificial Intelligence Research*, 36:547–556, 2009.

L. Kocsis and C. Szepesvári. Bandit based Monte-Carlo planning. In *Proc. ECML-2006*, pages 282–293. Springer, 2006.

R. Korf. Real-time heuristic search. *Artificial Intelligence*, 42:189–211, 1990.

Joel Lehman and Kenneth O. Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary computation*, 19(2):189–223, 2011.

Nir Lipovetzky and Héctor Geffner. Width and serialization of classical planning problems. In *Proc. ECAI*, pages 540–545, 2012.

Nir Lipovetzky, Miquel Ramirez, and Hector Geffner. Classical planning with simulators: Results on the atari video games. In *Proc. International Joint Conference on Artificial Intelligence (IJCAI-15)*, 2015.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. In *Proc. of the NIPS-2013 Workshop on Deep Learning*, 2013.

J. Pearl. *Heuristics*. Addison Wesley, 1983.

S. Richter and M. Westphal. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research*, 39(1):127–177, 2010.

R. Sutton and A. Barto. *Introduction to Reinforcement Learning*. MIT Press, 1998.

Analysis of Bagged Representations in PDDL

Pat Riddle, Mike Barley, Santiago Franco and Jordan Douglas
Department of Computer Science, University of Auckland
Private Bag 92019, Auckland, 1142 NZ

Abstract

In this paper, we demonstrate that many problems used in the IPC are more naturally represented as bags instead of sets. These tend to be problems with lots of objects. In the standard PDDL representations the objects are all denoted individually by unique names and this causes the problem solvers to encounter combinatorial explosions on trying to solve them. Frequently we don't care about individual objects, we want something to happen to a whole set of objects, but currently we are forced to identify them individually anyway. These bags of objects are similar to resources in scheduling. We propose a new formulation for these types of problems in the current PDDL STRIPS representation. We analyze the new and original representations in an attempt to determine when each representation performs better. In this paper's experiments, the best results were always obtained on the new representations. All new representations in this paper were generated manually, but we are currently developing a system that automatically generates these types of representations from the original PDDL representation, so throughout this paper the new representations are called the transformed representations.

Introduction

The PDDL based language is based on sets, so each individual object is uniquely identified which leads to combinatorical explosions. A more natural representation for problems with a large set of objects is a bag based representation (like resources in scheduling). In this paper we explore problems from 6 domains. We analyze results on two representations for each problem. We refer to these representations as the original and transformed representation, although all the transformed representations discussed in this paper were created manually. We are currently developing a transformation system which creates the new representations automatically and translates the resulting solution back into the original representation. (Riddle et al. 2015) The goal of this paper is to determine whether bagged-based representations have an advantage over the normal set-based PDDL representations for domains with a large number of objects of the same type.

PDDL (McDermott et al. 1998) is fundamentally based on sets. In a domain like sokoban-strips-opt-08, each of the stones are individually identified. For instance, in problem p20 there are 5 individually identified stones. The goal state specifies that each stone is at some goal location $(at\text{-}goal\ stone\text{-}05)$, but it doesn't matter which goal location a stone is at. This allows us to call each stone "stoneX". In the PDDL problem file we can rename all the stones in all the predicates to stoneX. We must also alter the goal state, otherwise PDDL's set semantics mean once one stone is put in any goal position the problem is solved. In this instance the goal state becomes $(and\ (at\ stoneX\ pos\text{-}03\text{-}03)\ (at\ stoneX\ pos\text{-}03\text{-}04)\ (at\ stoneX\ pos\text{-}03\text{-}05)\ (at\ stoneX\ pos\text{-}03\text{-}06)\ (at\ stoneX\ pos\text{-}03\text{-}07))$, which is equivalent to the original goal description. For sokoban-strips-opt-08, the domain file does not have to be altered to use this new representation.

We solved p20 in both representations with Fast Downward (Helmert 2006). Using the blind heuristic in the new representation it found a 31 cost solution (with 87 steps) in a search time of 3.07 seconds which expanded 752,651 states until last jump. Fast Downward's "until last jump" values return the number of states for the last fully expanded f-level. We use these counts throughout this paper to avoid any stochastic effects caused by search tree ordering. The problem solver on the original representation ran out of memory during f-level 31. These representations will have the same cost optimal solution, so we can look at the last f-level they both completed. In the original representation, at the end of f-level 30, Fast Downward expanded 30,006,650 states in 105.7 seconds. At the same f-level in the transformed representation Fast Downward expanded 752,651 states in 3.07 seconds. We use blind search to compare the actual size of the state space. This simple change, ignoring the names of identical objects, reduces the search tree size by a factor of 40, as can be seen in Table 3. The reduction factor is calculated by dividing the nodes expanded in the original representation by the nodes expanded in the transformed representation. If both problems are not solved we use the expanded nodes at the last common f-level in this calculation. This works for the sokoban-strips-opt-08 domain because no two stones can occupy the same location, so it never tries to enter $(at\ stoneX\ pos\text{-}4\text{-}3)$ twice within the same state.

We would like to use the same type of representations in other domains. For example, in the gripper domain we would like to call all the balls "ballX". The problem is two different balls can be in the same location within a state. If

Table 1: Transformed gripper problem representation

```
(define (problem strips-gripper-x-1)
(:domain gripper-strips)
(:objects n4 n3 n2 n1 n0 rooma roomb ballX left right)
(:init(room rooma)(room roomb)(ball ballX)(free left)
 (free right)(at-robby rooma)(more n0 n1)(more n1 n2)
 (more n2 n3)(more n3 n4)(gripper left)(gripper right)
 (count ballX rooma n4)
 (count ballX roomb n0))
(:goal(and(count ballX roomb n4))))
```

we enter $(at\ ballX\ rooma)$ more than once in a state, then when one of them is removed all of them will be removed because of PDDL's basic set representation. To solve this problem we create a bag representation that can be used in PDDL instead.

A Bagged Representation

In order to avoid having multiple copies of the same predicate in a state (e.g., $(at\ ballX\ rooma)(at\ ballX\ rooma)$) every predicate that refers to the "objects to be merged" will need to be replaced by a count predicate. So for instance in the gripper domain $(at\ ball1\ rooma)$ $(at\ ball2\ rooma)$ $(at\ ball3\ rooma)$ $(at\ ball4\ rooma)$ will be replaced by $(count\ ballX\ rooma\ 4)$. This is the equivalent of multiple identical predicates within the same state. To implement the counts, we need to do simple arithmetic. Unfortunately only a handful of optimal planners currently handle PDDL numerical fluents. We want to create PDDL that can be used by any PDDL planner, so we add simple counting predicates $(more\ n1\ n2)$ which allow us to alter the domain actions to increment or decrement the count predicates. The transformed representation for gripper is shown in Tables 1 and 2.

We created a gripper problem instance p250 with 250 balls. We solve this problem with both representations with Fast Downward using the blind heuristic. Note that the representation used in these experiments merged the grippers together as well as the balls, as opposed to the PDDL shown in the tables above. This more complex transformed representation is given at (Riddle 2015). With the transformed representation it found a 749 step solution in a search time of 1.22 seconds which expanded 1,495 states until last jump. Whereas with the original representation, Fast Downward runs out of memory during f-level 6. These representations will have the same length optimal solution, so we can look at the last f-level they both completed. In the original representation, at the end of f-level 5 Fast Downward expanded 250,502 states in 10.98 seconds. At the end of f-level 5 in the transformed representation Fast Downward expanded 9 nodes in 1.02 seconds. This change reduces the search tree size by a factor of 27,834, as can be seen in Table 3.

More Complex Domains

So far the transformations have been fairly easy because the objects that we wanted to rename only occured in a single predicate. Things become more complex when we have multiple predicates. For instance in the Barman-opt11-strips domain, let us assume we do not care which shots contain which drinks, but only care that there "exists" a shot with

Table 2: Transformed gripper domain representation

```
( define ( domain gripper-strips )
(:predicates (room ?r)(ball ?b)(gripper ?g)
  (at-robby ?r)(count ?b ?r ?n)(free ?g)(carry ?o ?g)
  (more ?n1 ?n2))

(:action move :parameters ( ?from ?to )
 :precondition (and (room ?from)(room ?to)
  (at-robby ?from))
 :effect (and (at-robby ?to)(not (at-robby ?from))))

(:action pick :parameters (?n1 ?n0 ?obj ?room ?gripper)
 :precondition (and (ball ?obj)(room ?room)
   (gripper ?gripper)(at-robby ?room)(free ?gripper)
   (more ?n1 ?n0)(count ?obj ?room ?n0))
 :effect (and (carry ?obj ?gripper)
   (not (count ?obj ?room ?n0))
   (count ?obj ?room ?n1)(not (free ?gripper))))

(:action drop :parameters (?n1 ?n0 ?obj ?room ?gripper)
 :precondition (and (ball ?obj)(room ?room)
   (gripper ?gripper)(carry ?obj ?gripper)(more ?n0 ?n1)
   (at-robby ?room)(count ?obj ?room ?n0))
 :effect (and (not (count ?obj ?room ?n0))
   (count ?obj ?room ?n1)(free ?gripper)
   (not (carry ?obj ?gripper)))))
```

each specified drink. Similarly, we have two hands "right" and "left" and do not care which hand picks up and holds something, only that we don't pick up 3 things! We want to create the same type of abstracted representation for the barman domain. Unfortunately now we have multiple predicates that refer to shots. To overcome this problem, we create macro-predicates.

The transformed barman-opt11-strips problem for pfile01-001.pddl is given at (Riddle 2015). In the original representation, there are a number of predicates that take a shot as an argument. These are: $(ontable\ ?c$ - $container)$, $(holding\ ?h$ - $hand\ ?c$ - $container)$, $(clean\ ?c$ - $container)$, $(empty\ ?c$ - $container)$, $(contains\ ?c$ - $container\ ?b$ - $beverage)$, $(used\ ?c$ - $container\ ?b$ - $beverage)$. This kind of distributed representation is very common in PDDL. It can be used because the unique identifiers ("shot1" "shot2" etc.) allow it to associate the distributed predicates with each other. Unfortunately, the unique identifiers are the cause of the combinatorial explosion! We combine the predicates into a single macro-predicate. The initial state in the original representation had $\{(ontable\ shot1)$, $(ontable\ shot2)$, $(ontable\ shot3)$, $(ontable\ shot4)$, $(clean\ shot1)$, $(clean\ shot2)$, $(clean\ shot3)$, $(clean\ shot4)$, $(empty\ shot1)$, $(empty\ shot2)$, $(empty\ shot3)$, $(empty\ shot4)\}$. This is represented in the new representation as $(count1\ shotX\ empty\ clean\ ontable\ 4)$, which states that in the initial state there is: 1) a $(clean\ shotX)$, 2) an $(empty\ shotX)$, and 3) an $(ontable\ shotX)$ for the same 4 unique shots. This solves the problem of distributed representations, but it causes problems with representing the goal state.

The original representation's partial goal description was $(and\ (contains\ shot1\ cocktail3)$, $(contains\ shot2$

$cocktail1$), ($contains\ shot3\ cocktail2$)). It contains some information from our macro-predicate but not all the information. We could use 1) variables or 2) existential quantifiers, or 3) an OR construct in our final goal description. Unfortunately many of the current IPC planners do not allow any of these options in a goal description. Alternatively we create a goal-predicate that only contains the information required in the goal description. For this problem that is: (and($count\text{-}goal\ shotX\ cocktail3\ 1$) ($count\text{-}goal\ shotX\ cocktail1\ 1$) ($count\text{-}goal\ shotX\ cocktail2\ 1$)). Of course the domain actions must be designed to deal with the count macro-predicates and the newly created goal-predicates, these can be seen in the Barman domain file at (Riddle 2015).

We solve problem pfile01-001.pddl in these two representations with Fast Downward using the blind heuristic. In the new representation it returned a 90 cost solution with 36 steps in a search time of 4.41 seconds which expanded 289,946 states until last jump. In the original representation, it returned a 90 cost solution with 36 steps in a search time of 38.91 seconds which expanded 5,967,050 states until last jump. This reduces the search tree size by a factor of 21, as is shown in Table 3.

Domains with Lots of Objects

The advantage of the "bagged representation" is that it scales much better than the standard representation, for instance you can solve the 250 ball gripper problem shown above. We present three additional domains to emphasize this point. The first is the Spanner domain from the IPC 2011 Learning track, created by Amanda Coles, Andrew Coles, Maria Fox and Derek Long. The second is the ChildSnack domain from the sequential track in IPC-2014, created by Raquel Fuentetaja and Tomás de la Rosa Turbides. The third is the Pizza domain also created by Raquel Fuentetaja and Tomás de la Rosa Turbides.

In the Spanner domain, we solve a smaller version of problem pfile01-001.pddl (which has 30 nuts to tighten and 30 spanners). In the transformed representation we merge all the nuts and spanners. We keep track of the counts of spanners at each location and the number of loose and tightened nuts.

In the ChildSnack domain, we solve problem childsnack_pfile01.pddl (which has 6 children, 6 breads, 6 contents, and 8 sandwiches). In the transformed representation we merge the bread, contents, and sandwiches. We keep track of the counts of each item.

In the Pizza domain, we solve problem rndgoal_pizza_base_p02.pddl (which has 5 guests and 16 slices of pizza). In the transformed representation we merge the slices. We keep track of the counts for each item.

We present a number of experiments on these domains (as well as the 3 domains presented earlier), but first we discuss the related research.

Related Research

There has been considerable work on problem reformulation, starting with George Polya's *How to Solve It* (1957). Due to lack of space we will focus on planning-specific reformulation research. The Fast Downward system (Helmert 2006) transforms the PDDL representation into a multi-valued planning task, similar in spirit to SAS$^+$. Using this representation, the system generates four internal data structures, which it uses to search for a plan. Helmert (Helmert 2009), extending this work, focused on turning PDDL into a concise grounded representation of SAS$^+$. Additional work in this area transforms between PDDL and binary decision diagrams (Haslum 2007), transforms between PDDL and causal graphs (Helmert 2006), and identifies and removes irrelevant parts from a problem representation (Haslum, Helmert, and Jonsson 2013).

To the best of our knowledge, little research has focused on transforming a PDDL representation into another PDDL representation. Two notable exceptions are (Areces et al. 2014; de la Rosa and Fuentetaja 2015), the latter of which we describe at the end of this section. Instead, it has almost exclusively focused on either reformulating PDDL to a planner's internal representation or transforming one internal representation to another. Although these approaches lead to more knowledge about the connectivity of the search space, and the power to alter the representation in these internal forms, there are advantages to altering PDDL. First, the new representations can be used by any PDDL planner. Second, in some domains, creating SAS$^+$ or the domain transition graphs take a lot of time; this might be avoided by first transforming to a different PDDL representation and transforming that into SAS+ or an internal representation.

Our system has much in common with symmetry reduction systems. Fox and Long (1999) group symmetric objects together in TIM. They require objects to be indistinguishable in both the initial state and the goal description. They keep track of the symmetry groups during planning but only with respect to the goal description, so they cannot remove all the symmetries in gripper.

Pochter et. al. (2011) generalize the work by Fox and Long, by using generators to create automorphic groups. These groups are based on SAS$^+$ and so are more general than objects. They still require the symmetric groups to be indistinguishable in both the initial and goal description.

Domshlak et. al. (2012) extended this work to only require symmetric groups to be indistinguishable in the goal description in the DKS system. They compared their work to Pochter's system, where they solved 8 more problems over 30 domains.

Metis (Alkhazraji et al. 2014), uses orbit search to do symmetry breaking. It is an improvement on DKS, since it does not store extra information with each state. Metis also includes an incremental LM-cut heuristic and partial order reduction with strong stubborn sets. In the experiments in this paper, Metis is always run with symmetry breaking but without these last two components.

The closest work to our automated system for creating these transformations is the system by de la Rosa et. al. (2015). They reformulate PDDL into PDDL and they merge objects in a similar way. The main differences are 1) we merge objects if they are the same in the initial state or the same in the goal description whereas their system merges

Table 3: Analysis of both representations across 6 domains using blind search. X signifies the planner was killed because it ran out of memory under the 2014 IPC constraints. Solution state expansions are "until last jump" to normalize for different tree orderings on the last level. Reduction Faction of expanded states shows by what factor it has been reduced; we use the last common f-bound in this calculation if either representation is not solved.

domain	problem	heuristic	representation	# SAS+ vars	# SAS+ actions	search time	states expanded	states exp. last common f-bound	common f-bound	solution cost	solution length	Reduction Factor of Expanded States
sokoban	p20	blind	transformed	25	120	3.07s	752,651	752,651	30	31	87	40
			original	35	312	X	X	30,006,650	30	X	X	
gripper	p250	blind	transformed	5	500,002	1.22s	1,495	9	5	749	749	27,834
			original	253	2,002	X	X	250,502	5	X	X	
barman-opt11	pfile01-001	blind	transformed	147	4,501	4.41s	289,946			90	36	21
			original	62	358	38.91s	5,967,050			90	36	
spanner	pfile01-001-small	blind	transformed	16	1,851	18.38s	8,699,505	126,611	53	111	111	71
			original	91	981	X	X	9,038,188	53	X	X	
ChildSnack	child-snack_pfile01	blind	transformed	74	1,656	1.8s	90,162	272	5	20	20	5,553
			original	36	456	X	X	1,510,534	5	X	X	
pizza	base_p02	blind	transformed	28	2,304	1.05s	10,837	205	6	15	15	3,400
			original	37	14,668	X	X	705,106	6	X	X	

objects if they do not appear in the goal description 2) they explicitly use numeric fluents in their modeling, restricting them to planners that support them such as metric-FF (Hoffmann 2003) 3) both our systems translate the solution back to the original representation but our system can generate plans which have different explicit values specified in the goal state.

Experimental Results

In this section we explore 6 domains using a number of different problem solvers. First we analyze the representations using blind search. Next we explore two different heuristics, LM-cut and iPDB. Then we explore the difference between transforming the representation versus using a symmetry reduction technique such as Metis. We also run Metis on the transformed representation to see if additional symmetries are found. Lastly we explore the bidirectional search approach used in SymBA* on both representations.

Blind Search

We discussed the sokoban, gripper and barman domains earlier. Table 3 shows the reduction factor of expanded states for each problem in these 3 domains is 40, 27,834 and 21 respectively.

In the Spanner domain, we solve a smaller version of problem pfile01-001.pddl (which has 30 nuts to tighten and 30 spanners) in both representations with Fast Downward using the blind heuristic. In the transformed representation it returns a 111 step solution in a search time of 18.38 seconds which expanded 8,699,505 states until last jump. Whereas in the original representation it runs out of memory during f-level 54. At the end of f-level 53 it expanded 9,038,188 states in 35.08 seconds. At the end of f-level 53 in the transformed representation it expanded 126,611 nodes in 1.24 seconds. The new representation for spanner can be seen at (Riddle 2015). The reduction factor for this problem is 71.

In the ChildSnack domain, we solve problem child-snack_pfile01.pddl in both representations with Fast Downward using the blind heuristic. In the transformed representation it found a 20 step solution in a search time of 1.8 seconds which expanded 90,162 states until last jump. Whereas in the original representation it cannot run out of memory during f-level 6. At the end of f-level 5 it expanded 1,510,534 states in 82.06 seconds. At the end of f-level 5 in the transformed representation it expanded 272 nodes in 1.01 seconds. The new representation for ChildSnack can be seen at (Riddle 2015). The reduction factor for this problem is 5,553.

In the Pizza domain, we solve problem rnd-goal_pizza_base_p02.pddl in both representations with Fast Downward using the blind heuristic. In the transformed representation it found a 15 step solution in a search time of 1.05 seconds which expanded 10,837 states until last jump. Whereas in the original representation it runs out of memory during f-level 7. At the end of f-level 6 it expanded 705,106 states in 34.2 seconds. At the end of f-level 6 in the transformed representation it evaluated 205 nodes in 1.02 second. The new representation for pizza can be seen at (Riddle 2015). The reduction factor for this problem is 3,400.

It is obvious that the transformed representation always has a smaller state space. But what happens when a heuristic is used? In the next two sections we explore the affect our new representation has on two separate heuristics, LM-cut and iPDB.

LM-cut

We now look at both representations with the LM-cut heuristic. LM-cut is a state-of-the-art heuristic which can be viewed as an approximation to the optimal relaxation heuristic h^+(Helmert and Domshlak 2009). Table 4 shows the results.

In the Sokoban domain both representations solve the problem. LM-cut is much less accurate on the transformed representation with an initial h-value of 6 compared to 15 in the original representation. This causes more nodes to be expanded and the reduction factor is 0.3 for this problem.

In the Gripper domain only the transformed representation is solved although it does go over the IPC time constraints. LM-cut is less accurate in the transformed representation with an initial h-value of 251 compared to 501 in the original representation. Despite this, the reduction factor is 332 for this problem.

In the Barman-opt11 domain both representations are solved. LM-cut is still less accurate in the transformed domain with an initial h-value of 16 compared to 28 in the

Table 4: Analysis of both representations across 6 domains using the LM-cut heuristic. X signifies the planner was killed because it ran out of memory under the 2014 IPC constraints. Solution state expansions are "until last jump" to normalize for different tree orderings on the last level. Reduction Faction of expanded states shows by what factor it has been reduced; we use the last common f-bound if if either representation is not solved.

domain	problem	heur	reps	# SAS+ vars	# SAS+ actions	search time	states expanded	states exp. last common f-bound	common f-bound	solution cost	solution length	Initial h-value	Reduction Factor of Expanded States
sokoban	p20	lmct	trans	25	120	19.31s	658,420			31	126	6	0.3
			orig	35	312	31.13s	219,203			31	126	15	
gripper	p250	lmct	trans	5	500,002	9879.52s	1492	752	501	749	749	251	332
			orig	253	2002	X	X	250,001	501	X	X	501	
barman-opt11	pfile01-001	lmct	trans	147	4,501	355.91s	117,832			90	36	16	11
			orig	62	358	268.7s	1,345,145			90	36	28	
spanner	pfile01-001-small	lmct	trans	16	1,851	1959.87s	635,477	15	83	111	111	83	2,179
			orig	91	981	X	X	32,687	83	X	X	82	
ChildSnack	child-snack_pfile01	lmct	trans	74	1,656	7.52s	26,607	24	12	20	20	12	33,335
			orig	36	456	X	X	800,051	12	X	X	10	
pizza	base_p02	lmct	trans	28	2,304	1.52s	1,765	227	10	15	15	5	4,064
			orig	37	14,668	X	X	922,581	10	X	X	5	

original representation. Despite this, the reduction factor is 11 for this problem. The search space is reduced to such an extent it is faster to solve the problem in the transformed domain even though the heuristic is less accurate.

In the Spanner domain only the transformed representation is solved. The initial h-values are nearly identical at 83 and 82 for the transformed and original representations respectively. A comparison of the last common f-boundary shows a reduction factor of 2,179.

In the ChildSnack domain only the new representation is solved. LM-cut is actually more accurate on the transformed representation with an initial h-value of 12 compared to 10 for the original. A comparison of the last common f-boundary shows a reduction factor of 33,335.

In the Pizza domain only the new representation is solved. The initial h-value is 5 for both representations. A comparison of the last common f-boundary shows a reduction factor of 4,064.

LM-cut displays a lower initial h-value in 3 domains, a marginally higher value in 2 domains and the same value in 1. When the initial h-value was higher or the same in the transformed representation, the reduction factor between representations was higher using LM-cut than using the blind heuristic. The question remains whether other heuristics perform differently on the two representations. We attempt to answer this question by looking at the iPDB heuristic.

iPDB

We explore the two representations with the iPDB heuristic (Haslum et al. 2007). The iPDB systems creates PDBs by a greedy search through a space of abstractions. Table 5 shows the results.

In the Sokoban domain both representations are solved, but the transformed domain has a very low initial h-value and therefore expands more states than the original domain resulting in a reduction factor of 0.58.

In the Gripper domain only the transformed problem is solved and finds the perfect initial h-value resulting in 0 nodes until last jump in the final state; therefore the reduction factor is undefined. There is also no common f-boundary explored.

In the Barman domain the initial h-values for the two representations are very close at 18 and 19 for the transformed and original respectively. The transformed representation still expands fewer states resulting in a reduction factor of 17.

In the Spanner domain the transformed representation finds the perfect initial h-value of 111 resulting in 0 nodes until last jump in the final state; therefore the reduction factor is undefined. The original representation finds an initial h-value of 81 but runs out of memory at f-level 84.

In the ChildSnack domain, both representations get an initial h-value of 6, but only the transformed representation is solved. Comparing the number of nodes at the last common f-boundary, 10, the reduction factor is 4,821.

Table 5: Analysis of both representations across 6 domains using the iPDB heuristic. X signifies the planner was killed because it ran out of memory under the 2014 IPC constraints. Solution state expansions are "until last jump" to normalize for different tree orderings on the last level. Reduction Faction of expanded states shows by what factor it has been reduced; we use the last common f-bound if either representation is not solved.

domain	problem	heur	reps	# SAS+ vars	# SAS+ actions	search time	states expanded	states exp. last common f-bound	common f-bound	solution cost	solution length	Initial h-value	Reduction Factor of Expanded States
sokoban	p20	iPDB	trans	25	120	2.92s	737,817			31	90	4	0.58
			orig	35	312	2.47s	429,290			31	115	15	
gripper	p250	iPDB	trans	5	500,002	1.05s	0	0	749	749	749	749	undefined
			orig	253	2,002	X	X	281,628	254	X	X	251	
barman-opt11	pfile01-001	iPDB	trans	147	4,501	4.32s	230,634			90	36	18	17
			orig	62	358	31.45s	3,989,300			90	36	19	
spanner	pfile01-001-small	iPDB	trans	16	1,851	3.88s	0			111	111	111	undefined
			orig	91	981	X	X	222,815	83	X	X	81	
ChildSnack	child-snack_pfile01	iPDB	trans	74	1,656	1.59s	65,256	356	10	20	20	6	4821
			orig	36	456	X	X	1,716,166	10	X	X	6	
pizza	base_p02	iPDB	trans	28	2,304	1.01s	2,728	341	10	15	15	5	4,695
			orig	37	14,668	X	X	1,601,001	10	X	X	5	

In the Pizza domain, both representations get an initial h-value of 5, but only the transformed representation solves the problem. Comparing the number of nodes at the last common f-boundary, 10, the reduction factor is 4,695.

It is clear that even with heuristics many of these domains are solved more efficiently in the transformed representation. The question remains how do they compare to systems which use symmetry breaking techniques? In the next section we do several experiments using the Metis system (Alkhazraji et al. 2014).

Metis

The question remains, does transforming the representation provide any additional benefits over symmetry reduction techniques? The Metis system (Alkhazraji et al. 2014) is a state of the art system that uses symmetry reduction techniques and partial order reduction. Throughout this paper we run Metis without the partial order reduction component, because we wish to compare the symmetry reduction abilities of Metis to the symmetry reduction of the transformed representation, without mudding the waters with other components. We will conduct 3 experiments to explore this question. The first experiment compares Metis on the original representation compared to the transformed representation. We use the blind heuristic on both these representations.

In the Sokoban and Gripper domains the reduction factor is 1.0 showing that in these representations the symmetry reduction is the same. In the Pizza domain, the reduction factor is 0.1 (less than 1.0) which shows that Metis finds more symmetries in the original representation than are removed in the transformed representation. In Barman, Spanner, and ChildSnack the reduction factor is greater than 1.0. We infer from this that our transformed representation has managed to remove symmetries that Metis has missed. In ChildSnack this difference is very small (reduction factor of 1.05) but Barman and Spanner are larger at 1.8 and 2.3 respectively. We will discuss why this happens in more depth at the end of this section.

The previous experiments used the blind heuristic. It shows that the two systems find different symmetries. The question remains what will happen if heuristics are introduced. The next experiment runs Metis with symmetry breaking and LM-cut on the original representation and LM-cut on the transformed representation.

In the Sokoban domain, LM-cut is a big help in the original representation while much less effective in the transformed representation reducing the Reduction factor from 1.0 to 0.09.

In the Gripper domain, LM-cut has little affect. The reduction factor is reduced from 1.00 to 0.99 and both representations expand almost as many nodes as they do in blind search.

In the Barman domain LM-cut is effective in both representations and lowers the reduction factor from 1.8 to 1.4. In the Spanner domain LM-cut is more effective in the transformed domain than in the original and increases the reduction factor from 2.3 to 90420. In the ChildSnack domain, LM-cut is more effective in the transformed domain and increases the reduction factor slightly from 1.05 to 3.15. In the Pizza domain, LM-cut is equally effective on both domains with the reduction factor remaining at 0.1.

The last experiment compares Metis with symmetry breaking and blind search on both representations. We examine the change in the reduction factor from Table 6 to Table 8 to quantify this effect. This shows whether Metis finds additional symmetries in the transformed domain. In the Sokoban and Gripper domains the reduction factor remains steady at 1.0, showing that both techniques find the same symmetries. In the other four domains, the reduction factors are greater than 1.0 showing that the transformed representations removed more symmetries than Metis. In the Spanner domain the reduction factor remains at 2.3 which shows that the transformed space already removed all the symmetries Metis could find.

In the Barman domain the reduction factor goes from 1.8 to 2.9, while in the ChildSnack domain the reduction factor increases from 1.05 to 3.15 and in the Pizza domain the reduction factor increases from 0.1 to 1.12. In these 3 domains, Metis can remove additional symmetries from the transformed representation. Our hypothesis about this behaviour is that these increases show that there were implicit symmetries in the original representation that were made explicit in the transformed representation allowing Metis to remove them. This is a very interesting and unexpected result.

Let us now look at the bigger picture. In some domains (like Sokoban and Gripper) Metis and the transformed rep-

Table 6: Analysis of the old representation using Metis with symmetry reduction and blind search, compared to the new representation using blind search. X signifies the planner was killed because it ran out of memory under the 2014 IPC constraints. Solution state expansions are "until last jump" to normalize for different tree orderings on the last level. Reduction Faction of expanded states shows by what factor it has been reduced; we use the last common f-bound if either representation is not solved.

domain	problem	heuristic	reps	# SAS+ vars	# SAS+ actions	search time	states expanded	states exp. last common f-bound	common f-bound	solution cost	solution length	Reduction Factor of Expanded States
sokoban	p20	blind	trans	25	120	3.07s	752,651			31	87	1.0
		blind-Metis	orig	35	312	2.9s	752,983			31	87	
gripper	p250	blind	trans	5	500,002	1.22s	1,495			749	749	1.0
		blind-Metis	orig	253	2,002	8.27s	1,495			749	749	
barman-opt11	pfile01-001	blind	trans	147	4,501	4.41s	289,946			90	36	1.8
		blind-Metis	orig	62	358	4.81s	533,211			90	36	
spanner	pfile01-001-small	blind	trans	16	1,851	18.38s	8,699,505	3,275,046	64	111	111	2.3
		blind-Metis	orig	91	981	X	X	7,594,192	64	X	X	
ChildSnack	child-snack_pfile01	blind	trans	74	1,656	1.8s	90,162			20	20	1.05
		blind-Metis	orig	36	456	3.98s	94,498			20	20	
pizza	base_p02	blind	trans	28	2,304	1.05s	10,837			15	15	0.1
		blind-Metis	orig	37	14,668	0.11s	1,078			15	15	

Table 7: Analysis of the old representation using Metis symmetry reduction and LM-cut search, compared to the new representation using LM-cut search. X signifies the planner was killed because it ran out of memory under the 2014 IPC constraints. Solution state expansions are "until last jump" to normalize for different tree orderings on the last level. Reduction Faction of expanded states shows by what factor it has been reduced; we use the last common f-bound if either representation is not solved.

domain	problem	heuristic	reps	# SAS+ vars	# SAS+ actions	search time	states expanded	states exp. last common f-bound	common f-bound	solution cost	solution length	Initial h-value	Reduction Factor of Expanded States
sokoban	p20	LM-cut	trans	25	120	19.31s	658,410			31	94	6	0.09
		Metis	orig	35	312	8.38s	58,152			31	124	15	
gripper	p250	LM-cut	trans	5	500,002	9879.52s	1,492			749	749	251	0.99
		Metis	orig	253	2,002	24.08s	1,486			749	749	501	
barman-opt11	pfile01-001	LM-cut	trans	147	4,501	223.49s	80,182			90	36	16	1.4
		Metis	orig	62	358	24.35s	114,120			90	36	28	
spanner	pfile01-001-small	LM-cut	trans	16	1,851	1959.83s	635,477	61	89	111	111	83	90420
		Metis	orig	91	981	X	X	5,515,622	89	X	X	82	
ChildSnack	child-snack_pfile01	LM-cut	trans	74	1,656	2.03s	8,687			20	20	12	3.15
		Metis	orig	36	456	3.24s	27,401			20	20	20	
pizza	base_p02	LM-cut	trans	28	2,304	1.52s	1,765			15	15	5	0.1
		Metis	orig	37	14,668	0.36s	174			15	15	5	

resentation find the same symmetries. This is what we would expect. In other domains (such as Barman) we know that we are finding more symmetries, based on the fact that the objects are the same in the initial state (e.g., we merge all the shots together). This is a type of symmetry not currently handled by the Metis symmetry breaking. So it makes sense that the reduction factor for Barman is greater than 1.0 in Table 8. In the Pizza domain we did not merge the types $tray$ or $people$, therefore it makes sense that Pizza's reduction value in Table 6 is less than 1.0. Both Spanner and ChildSnack are greater than 1.0 and we do not know why we find more symmetries. Certainly in ChildSnack we should find less symmetries because we did not merge the "child" type.

A reduction value in Table 8 being greater than 1.0 means that the transformed domain is removing symmetries that Metis is not removing. This is true of 4 of the domains, including the 3.9 reduction factor in ChildSnack, 2.9 in Barman, 2.3 in Spanner and 1.12 in Pizza. This shows that the transformed representation in combination with Metis is removing further symmetries than Metis did in the original representation.

We do not have a definitive explanation for these results. Our hypothesis is that there were implicit symmetries in the original representation that were made explicit in the transformed representation allowing Metis to remove them. It is related to the very different representations given by the SAS+ variables. Looking at the differences in the number of SAS+ variables and operators, it is not that surprising that these differences would cause different symmetries to be exposed. To further explore the differences caused by the SAS+ variables and operators we will explore one more problem solver, SymBA*.

SymBA*

The SymBA* system (Torralba et al. 2014) is a bidirectional symbolic problem solver built on Fast Downward. It tries to perform blind bidirectional search. When it determines it will run out of space it performs bidirectional heuristic search. This system performs very well when it can stay at the blind search level. Because our transformed representations have a much smaller state space, they are a natural fit for pairing with SymBA*. We run SymBA* on both representations. It runs a bidirectional symbolic search so it is difficult to compare expanded nodes with our previous results, so only times are given. Additionally it uses a more extensive process to determine SAS+ variables and operators (Alcázar and Torralba 2015). It turns out that this is very advantageous to our transformed representations.

Comparing the number of SAS+ variables and operators, the SymBA* translator returns fewer SAS+ variables in both gripper representations and in the transformed representation on ChildSnack. The number of operators is reduced in all the transformed representations as well as two of the original representations. This is a huge help to the problem

Table 8: Analysis of both representations across 6 domains using the Metis symmetric reduction and blind search. X signifies the planner was killed because it ran out of memory under the 2014 IPC constraints. Solution state expansions are "until last jump" to normalize for different tree orderings on the last level. Reduction Faction of expanded states shows by what factor it has been reduced; we use the last common f-bound if either representation is not solved.

domain	problem	heuristic	representation	# SAS+ vars	# SAS+ actions	search time	states expanded	states exp. last common f-bound	common f-bound	solution cost	solution length	Reduction Factor of Expanded States
sokoban	p20	blind-Metis	transformed	25	120	2.37s	752,651			31	87	1.0
			original	35	312	2.9s	752,983			31	87	
gripper	p250	blind-Metis	transformed	5	500,002	0.05s	1,495			749	749	1.0
			original	253	2,002	8.27s	1,495			749	749	
barman-opt11	pfile01-001	blind-Metis	transformed	147	4,501	2.64s	185,224			90	36	2.9
			original	62	358	4.81s	533,211			90	36	
spanner	pfile01-001-small	blind-Metis	transformed	16	1,851	18.38s	8,699,505	3,275,046	63	111	111	2.3
			original	91	981	X	X	7,594,192	63	X	X	
ChildSnack	child-snack_pfile01	blind-Metis	transformed	74	1,656	0.34s	24,336			20	20	3.9
			original	36	456	3.98s	94,498			20	20	
pizza	base_p02	blind-Metis	transformed	28	2,304	0.01s	966			15	15	1.12
			original	37	14,668	0.11s	1078			15	15	

Table 9: Analysis of both representations across 6 domains using the SymBA* problem solver. X signifies the planner was killed because it ran out of memory under the 2014 IPC constraints.

domain	problem	representation	# SAS+ vars	# SAS+ actions	search time
sokoban	p20	transformed	24	120	1.48s
		original	29	267	8.28s
gripper	p250	transformed	5	2,002	5.63s
		original	253	2,002	XXX
barman-opt11	pfile01-001	transformed	147	3,861	9.58s
		original	62	310	6.91s
spanner	pfile01-001-small	transformed	16	981	0.59s
		original	91	981	1.63s
ChildSnack	child-snack_pfile01	transformed	64	298	0.31s
		original	36	456	231.01s
pizza	base_p02	transformed	28	1,332	0.25s
		original	37	14,668	X

solver in terms of both time and memory. Some of these reductions are massive, such as gripper on the transformed domain, where 500,002 operators becomes 2,002 operators. Of course the SymBA* translator could be used with the regular Fast Downward problem solvers, but these experiments are not in the scope of this paper.

Best Overall Results

In Table 10 the search times for all the experiments conducted are summarized. If we look at which combinations were best for each problem, we see that the reformulated representation was always part of the best combination and that Metis with blind search and SymBA* each were best for 3 out of the 6 problems. While these results must be taken with a grain of salt, since they are only single runs on single problems in each domain, they are still interesting. It is possible that transforming the space and spending additional time to get good SAS+ variables and operators and then running blind search with symmetry reduction or blind bidirectional search on these much smaller search spaces is a reasonable approach to take and should be explored in greater detail.

Problems Caused by Bagged Representations

Unfortunately the transformed representation is not always better than the old representation. For instance even the transformed Sokoban representation has some drawbacks with the LM-cut and iPDB heuristics. This further supports earlier results by (Riddle, Holte, and Barley 2011) that there is no "best representation" for all problems within a domain. The main drawbacks concern: how the SAS+ variables and actions are made, how the heuristics are affected. We will discuss each of these drawbacks in turn.

SAS+ Variables This paper showed that the transformed representation sometimes gave a larger number of SAS+ variables and actions and sometimes made them smaller. In some domains such as sokoban, gripper, spanner, and pizza, the new representation actually has fewer variables and actions generated. This is a boon to the planners because they use less memory to represent each state. In other domains such as barman-opt11 and ChildSnack, more variables and actions are created. This means the planner will be using more memory, not less! One solution is to use a better SAS+ preprocessor, such as (Alcázar and Torralba 2015), this is an area we are currently exploring.

Table 10: Analysis of which representation/problem solver combination was best for each problem. Only search time was used since SymBA* cannot return nodes expanded.

domain	reps	blind time	LMct time	iPDB time	Metis Blind time	Metis LMct time	Symba time
sokbn	trans	3.07	19.31	2.92	2.37s		***1.48***
	orig	X	31.3	2.47	2.9	8.38	8.28
gripper	transf	1.22	X	1.05	***0.05***		5.63
	orig	X	X	X	8.27	24.08	XXX
barman	transf	4.41	355.9	4.32	***2.64***		9.58
	orig	38.91	268.7	31.45	4.81s	24.35	6.91
spannr	transf	18.38	1,959	3.88	18.38		***0.59***
	orig	X	X	X	X	X	1.63
chdsnk	transf	1.8	7.52	1.59	0.34		***0.31***
	orig	X	X	X	3.98	3.24	231.01
pizza	transf	1.05	1.52	1.01	***0.01***		0.25
	orig	X	X	X	0.11	0.36	X

Additionally we have found that the number of states expanded in the transformed representation is reduced to such an extent, that the extra costs of the additional SAS+ variables are not a problem. This is not always the case. In problems where you only bagged 2 or 3 objects together and got an increase in SAS+ variables, the new representation would likely take longer to solve.

Heuristics Another issue is how the new representation affects the heuristics, this relates to the section above because any change to the SAS+ variables will change the heuristics' predictive power. As we can see in our experiments sometimes the new representation improves the accuracy of the initial heuristic estimate and sometimes it decreases it.[1] LM-cut's initial heuristic values are better in the ChildSnack and Spanner domains, while iPDB's initial heuristic values are better in the Gripper and Spanner domains. iPDB's estimates in both these domains are perfect and significantly larger than in the original representation. In addition we have made the state space so much smaller, the fact that the heuristics are not as good frequently has little effect.

Conclusion and Further Research

To scale up PDDL representations to large numbers of objects, it is better to use a Bagged representation. This allows solving much larger problems with less combinatorial explosion. We are creating a system for automatically generating a bagged representation from the original PDDL representation (Riddle et al. 2015), although the PDDL representations shown here were created manually. This allows the use of a bagged representation even when you care which object is used, because it translates the solution back into the original representation. It does require that the objects to be merged are the same in the initial state or in the goal description.

The results using Metis show that symmetry breaking techniques and transforming the representation have some overlaps. But each seems to find some symmetries that the other misses. The best approach seems to be to use both approaches with symmetry breaking on the transformed space.

[1] We are assuming that the initial heuristic estimate reflects the overall accuracy of the heuristics.

The SymBA* results were also very interesting. It seems that using the transformed representation to reduce the state space and then using an uninformed search (especially a bidirectional one), could be an interesting new direction of research.

We found that altering the representation before the planning begins is a much better use of time. In Tables 3 & 5, only 2 problems can be solved in the original representation within the 2014 IPC space constraints, while all problems in the transformed representations can be solved. Using our automated system significantly reduces overall solving time, even after accounting for the extra preprocessing time.

The transformed representation is certainly not always better, especially when there are only a few symmetrical objects. Currently we use RIDA*'s (Barley, Franco, and Riddle 2014) runtime formula to choose between representations on a problem by problem basis. We are exploring more extensive methods (as in (Alcázar and Torralba 2015)) for creating SAS+ variables, which will help control any explosion in variables and actions caused by the bagged representation. We plan to explore which heuristics perform better on a bagged representation. Some heuristics seem to perform equally across most domains and some do particularly badly on the bagged representation.

We should include bagged representations in future IPC competitions. They are an easy way to explore larger spaces, whether automatically generated or created by hand. It would be interesting to see which heuristics and problems solvers work well on these representations and which do not.

References

Alcázar, V., and Torralba, Á. 2015. A reminder about the importance of computing and exploiting invariants in planning. In *ICAPS*.

Alkhazraji, Y.; Katz, M.; Mattmuller, R.; Pommerening, F.; Shleyfman, A.; and Wehrle, M. 2014. Metis: Arming fast downward with pruning and incremental computation. In *In the Eighth International Planning Competition Description of Participant Planners of the Deterministic Track*.

Areces, C.; Bustos, F.; Dominguez, M.; and Hoffmann, J. 2014. Optimizing planning domains by automatic action schema splitting. In *Twenty-Fourth International Conference on Automated Planning and Scheduling*.

Barley, M.; Franco, S.; and Riddle, P. 2014. Overcoming the utility problem in heuristic generation: Why time matters. In *ICAPS*.

de la Rosa, T., and Fuentetaja, R. 2015. Automatic compilation of objects to counters in automatic planning. case of study: Creation planning. http://e-archivo.uc3m.es/bitstream/handle/10016/19707/TR-objects-to-counters-10-2014.pdf. Accessed: 2015-02-20.

Domshlak, C.; Katz, M.; and Shleyfman, A. 2012. Enhanced symmetry breaking in cost-optimal planning as forward search. In *ICAPS*.

Fox, M., and Long, D. 1999. The detection and exploitation of symmetry in planning problems. In *IJCAI*, 956–961. Morgan Kaufmann.

Haslum, P.; Botea, A.; Helmert, M.; Bonet, B.; and Koenig, S. 2007. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *AAAI*, volume 22-2, 1007. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999.

Haslum, P.; Helmert, M.; and Jonsson, A. 2013. Safe, strong and tractable relevance analysis for planning. In *ICAPS*.

Haslum, P. 2007. Reducing accidental complexity in planning problems. In *IJCAI*, 1898–1903.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In *ICAPS*.

Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research* 26(1):191–246.

Helmert, M. 2009. Concise finite-domain representations for PDDL planning tasks. *Artificial Intelligence* 173(5).

Hoffmann, J. 2003. The metric-ff planning system: Translating"ignoring delete lists"to numeric state variables. *Journal of Artificial Intelligence Research* 291–341.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL-the planning domain definition language.

Pochter, N.; Zohar, A.; and Rosenschein, J. S. 2011. Exploiting problem symmetries in state-based planners. In *AAAI*.

Pólya, G. 1957. *How to solve it: A new aspect of mathematical method*. Princeton University Press, second edition.

Riddle, P.; Barley, M.; Franco, S.; and Douglas, J. 2015. Automated transformation of PDDL representations. In *International Symposium on Combinatorial Search*.

Riddle, P.; Holte, R.; and Barley, M. 2011. Does representation matter in the planning competition? In *SARA*.

Riddle, P. 2015. PDDL files for the representations. http://www.cs.auckland.ac.nz/∼pat/socs-2015/. Created: 2015-02-20.

Torralba, Á.; Alcázar, V.; Borrajo, D.; Kissmann, P.; and Edelkamp, S. 2014. SymBA*: A symbolic bidirectional A* planner. In *The 2014 International Planning Competition - Description of Planners*.

Finding and Exploiting LTL Trajectory Constraints in Heuristic Search

Salomé Simon and **Gabriele Röger**
University of Basel, Switzerland
{salome.simon,gabriele.roeger}@unibas.ch

An archival version (Simon and Röger 2015) of this paper will appear at SoCS 2015.

Abstract

We suggest the use of linear temporal logic (LTL) for expressing declarative information about optimal solutions of search problems. We describe a general framework that associates LTL_f formulas with search nodes in a heuristic search algorithm. Compared to previous approaches that integrate specific kinds of path information like landmarks into heuristic search, the approach is general, easy to prove correct and easy to integrate with other kinds of path information.

Introduction

Temporal logics allow to formulate and reason about the development of logic-based systems, for example about paths in factored state spaces. These are for instance common in planning, where temporal logics have always been present. As one extreme, the entire planning task can be specified in a temporal logic language and plans are generated by theorem proving (Koehler and Treinen 1995) or model construction (Cerrito and Mayer 1998).

In a different approach, the planning system can exploit domain-specific search control knowledge, given as part of the input. Such control knowledge can be a temporal logical formula that every "meaningful" plan must satisfy, therefore reducing the size of the search space and speeding up the plan generation (Bacchus and Kabanza 2000; Doherty and Kvarnström 2001).

This is related to planning for *temporally extended goals* (e. g. Bacchus and Kabanza 1998; Kabanza and Thiébaux 2005), where a plan does not need to reach a *state* with a given goal property but the action *sequence* must satisfy a given temporal formula.

PDDL 3 *state trajectory constraints* (Gerevini and Long 2005) integrate both worlds by extending a fragment of linear temporal logic with an additional operator that allows to specify a classical goal property.

For all these approaches, the temporal formulas are part of the input. In contrast, Wang et al. (2009) generate temporal task-specific trajectory constraints in a fully automated fashion from landmark orderings (Hoffmann, Porteous, and Sebastia 2004; Richter and Westphal 2010). This information is used to derive better estimates with the FF heuristic (Hoffmann and Nebel 2001) by evaluating it on a modified task that makes the constraints visible to the heuristic.

We argue that trajectory constraints in propositional linear temporal logic on finite traces (LTL_f) are suitable for a much more extensive application in heuristic search: a unified way of describing path-dependent information inferred during the search. In planning, there are many techniques that exploit a specific type of information and maintain it with specialized data structures and implementations. For example, landmark heuristics (Richter and Westphal 2010; Karpas and Domshlak 2009; Pommerening and Helmert 2013) track the landmark status for each operator sequence. A unified formalism for these techniques would offer two main advantages: *decoupling the derivation and exploitation of information* and *easily combining different sources of information*.

Currently the derivation and exploitation of information are integrated in most cases: someone proposes a new source of information and shows how it can correctly be exploited (requiring new correctness proofs every time). In our framework, LTL_f formulas meeting a feasibility criterion provide an interface between derivation and exploitation. Thus, new sources of information only need to be proven to be feasible, while new ways of exploiting information only need to be proven to derive (path) admissible heuristics for any information meeting this criterion. This separation will also make it easier for practitioners in real-world applications to specify domain-specific knowledge and correctly exploit it without needing to know the details of heuristic search.

Due to the unified LTL_f representation in our framework, it is trivial to combine information from different sources. If heuristics are able to use any kind of feasible information, combining information will strengthen the heuristic without needing to adapt the heuristic itself.

In the rest of the paper, we first introduce the necessary background on LTL_f and the considered planning formalism. We define a feasibility criterion for LTL_f trajectory constraints in optimal planning that allows to combine constraints and to develop them over the course of actions. Afterwards, we demonstrate how feasible trajectory constraints can be derived from several established sources of information. We present a heuristic based on LTL_f constraints and show how the entire framework integrates with the A^* algorithm. We finish with an experimental evaluation.

Background

In linear temporal logic (LTL, Pnueli 1977) a *world* w over a set $\mathcal{P}$ of propositional symbols is represented as set $w \subseteq \mathcal{P}$ of the symbols that are *true* in w. LTL extends propositional logic (with operators $\neg$ and $\wedge$) with a unary operator $\circ$ and a binary operator $\mathcal{U}$, which allow to formulate statements over infinite sequences of such worlds. Intuitively, $\circ\varphi$ means that φ is true in the *next* world in the sequence, and $\varphi\mathcal{U}\psi$ means that in some future world ψ is true and *until* then φ is true.

In addition, we use the following common abbreviations:

- Standard abbreviations of propositional logic such as $\vee$ (*or*), $\rightarrow$ (*implies*), $\top$ (*true*), $\bot$ (*false*)
- $\Diamond\varphi = \top\mathcal{U}\varphi$ expresses that φ will *eventually* be true.
- $\Box\varphi = \neg\Diamond\neg\varphi$ expresses that from the current world on, φ will *always* be true.
- $\varphi\mathcal{R}\psi = \neg(\neg\varphi\mathcal{U}\neg\psi)$ expresses that ψ holds until φ holds as well (*releasing* ψ) or forever.[1]

LTL on finite traces (LTL$_f$, De Giacomo and Vardi 2013; De Giacomo, De Masellis, and Montali 2014) defines semantics for *finite* world sequences, using the additional abbreviation *last* $= \neg\circ\top$, which is true exactly in the last world of the sequence.

Definition 1 (Semantics of LTL$_f$)**.** *Let φ be an LTL$_f$ formula over a set of propositional symbols $\mathcal{P}$ and let $\boldsymbol{w} = \langle w_0, \ldots, w_n\rangle$ be a sequence of worlds over $\mathcal{P}$.*

For $0 \leq i \leq n$, we inductively define when φ is true at instant *i (written $\boldsymbol{w}, i \models \varphi$) as:*

- *For $p \in \mathcal{P}$, $\boldsymbol{w}, i \models p$ iff $p \in w_0$*
- *$\boldsymbol{w}, i \models \neg\psi$ iff $\boldsymbol{w}, i \not\models \psi$*
- *$\boldsymbol{w}, i \models \psi_1 \wedge \psi_2$ iff $\boldsymbol{w}, i \models \psi_1$ and $\boldsymbol{w}, i \models \psi_2$*
- *$\boldsymbol{w}, i \models \circ\psi$ iff $i < n$ and $\boldsymbol{w}, i+1 \models \psi$*
- *$\boldsymbol{w}, i \models \psi_1\mathcal{U}\psi_2$ iff there exists a j with $i \leq j \leq n$ s.t. $\boldsymbol{w}, j \models \psi_2$ and for all $i \leq k < j$, $\boldsymbol{w}, k \models \psi_1$*

If $\boldsymbol{w}, 0 \models \varphi$, we say that φ is true in $\boldsymbol{w}$ or that $\boldsymbol{w}$ *satisfies* φ and also write this as $\boldsymbol{w} \models \varphi$.

In our application, we do not only want to reason about *finalized* world sequences but also about possible continuations of given prefixes. Bacchus and Kabanza (2000) showed for standard LTL with infinite world sequences how we can evaluate formulas *progressively* over the beginning $\langle w_0, \ldots, w_i\rangle$ of a world sequence. The idea is to create a formula which is satisfied by arbitrary continuations $\langle w_{i+1}, w_{i+2}, \ldots\rangle$ iff the original formula is satisfied by the entire sequence $\langle w_0, \ldots, w_i, w_{i+1}, \ldots\rangle$. As long as we do not progress over the last world in the sequence, the progression rules (shown in Figure 1) also work for LTL$_f$:

Proposition 1. *For an LTL$_f$ formula φ and a world sequence $\langle w_0, \ldots, w_n\rangle$ with $n > 0$ it holds that $\langle w_1, \ldots, w_n\rangle \models$ progress(φ, w_0) iff $\langle w_0, \ldots, w_n\rangle \models \varphi$.*

[1] While $\mathcal{R}$ can be expressed in terms of the essential operators, it is necessary for the positive normal form, which we will introduce and use later.

φ	**progress**(φ, w)
$p \in \mathcal{P}$:	$\top$ if $p \in w$, $\bot$ otherwise
$\neg\psi$:	$\neg$progress(ψ, w)
$\psi \wedge \psi'$:	progress$(\psi, w) \wedge$ progress(ψ', w)
$\psi \vee \psi'$:	progress$(\psi, w) \vee$ progress(ψ', w)
$\circ\psi$:	ψ
$\Box\psi$:	progress$(\psi, w) \wedge \Box\psi$
$\Diamond\psi$:	progress$(\psi, w) \vee \Diamond\psi$
$\psi\mathcal{U}\psi'$:	progress$(\psi', w) \vee$ (progress$(\psi, w) \wedge (\psi\mathcal{U}\psi')$)
$\psi\mathcal{R}\psi'$:	progress$(\psi', w) \wedge$ (progress$(\psi, w) \vee (\psi\mathcal{R}\psi')$)
last:	$\bot$
$\top$:	$\top$
$\bot$:	$\bot$

Figure 1: Progression Rules

For an example, consider formula $\varphi = \circ x \wedge \Diamond z$ over $\mathcal{P} = \{x, y, z\}$ stating that in the following world proposition x should be true and at some point z should be true. The progression of φ with a world $w = \{x, z\}$ is

$$\begin{aligned}\text{progress}(\circ x \wedge \Diamond z, w) &= \text{progress}(\circ x, w) \wedge \\ &\quad\ \text{progress}(\Diamond z, w) \\ &= x \wedge (\text{progress}(z, w) \vee \Diamond z) \\ &= x \wedge (\top \vee \Diamond z) \equiv x\end{aligned}$$

The progression eliminates the $\Diamond z$ from the formula because it is already satisfied with w. Subformula $\circ x$ gets replaced with x, which fits the requirement that x should now be true if φ is satisfied by the overall world sequence.

In the special case where a progression results to $\bot$ (or $\top$), it is clear that no (or any) continuation will result in an overall world sequence that satisfies the original formula.

We consider search problems given as *STRIPS planning tasks* with action costs. Such a planning task is a tuple $\Pi = \langle V, A, I, G\rangle$, where V is a set of propositional *state variables*, A is a set of *actions*, $I \subseteq V$ is the *initial state* and $G \subseteq V$ is the *goal description*. An *action* $a = \langle pre(a), add(a), del(a), c(a)\rangle \in A$ consists of the *precondition* $pre(a) \subseteq V$, the *add effects* $add(a) \subseteq V$, the *delete effects* $del(a) \subseteq V$ and the *action cost* $c(a) \in \mathbb{R}_0^+$. A *state* $s \subseteq V$ is defined by a subset of the variables. Action a is *applicable in state* s if $pre(a) \subseteq s$. Applying a in s leads to the *successor state* $s[a] = (s \setminus del(a)) \cup add(a)$. A *path* is a sequence of actions $\pi = \langle a_1, \ldots, a_n\rangle$ and is applicable in state s if the actions are applicable consecutively. We denote the resulting state with $s[\langle a_1, \ldots, a_n\rangle]$. For the empty path, $s[\langle\rangle] = s$. The *cost* of the path is $c(\pi) = \sum_{i=1}^{n} c(a_i)$. A *plan* is a path π such that $G \subseteq I[\pi]$. It is *optimal* if it has minimal cost.

We will formulate LTL$_f$ trajectory constraints that do not only cover state variables but also action applications. We follow the approach by Calvanese, De Giacomo, and Vardi (2002) and introduce an additional variable a for each action of the planning task. Therefore, the set $\mathcal{P}$ comprises these action variables plus the state variables of the planning task.

For the concatenation of two finite sequences σ and σ', we write $\sigma\sigma'$. For example, $\langle w_1, \ldots, w_n\rangle\langle w'_1, \ldots, w'_m\rangle = \langle w_1, \ldots, w_n, w'_1, \ldots, w'_m\rangle$. We use this notation for action and world sequences.

Feasible LTL$_f$ Formulas for Optimal Planning

Graph search algorithms like A* operate on search nodes n that associate a state $s(n)$ with a path of cost $g(n)$ from the initial state I to this state. We want to associate search nodes with LTL$_f$ trajectory constraints that characterize how the path to the node should be continued. This information can then be exploited for deriving heuristic estimates.

Such per node LTL$_f$ constraints are suitable for optimal planning if they are satisfied by any continuation of the path to the node into an optimal plan. To capture this notion, we define the world sequence induced by path $\rho = \langle a_1, \ldots, a_n\rangle$ in state s as $\boldsymbol{w}^{\boldsymbol{s}}_{\boldsymbol{\rho}} = \langle\{a_1\} \cup s[a_1], \{a_2\} \cup s[\langle a_1, a_2\rangle], \ldots, \{a_n\} \cup s[\rho], s[\rho]\rangle$ and introduce the following feasibility criterion:

Definition 2 (Feasibility for nodes)**.** *Let Π be a planning task with goal G and optimal plan cost h^* and let n be a node in the search space that is associated with state s.*

An LTL$_f$ formula φ is feasible for n *if for all paths ρ such that*

- *ρ is applicable in s,*
- *the application of ρ leads to a goal state ($G \subseteq s[\rho]$), and*
- *$g(n) + c(\rho) = h^*$*

it holds that $\boldsymbol{w}^{\boldsymbol{s}}_{\boldsymbol{\rho}} \models \varphi$.

If the path to node n is not a prefix of an optimal plan, then any formula is feasible for n. Otherwise, φ must be true for any continuation of the path into an optimal plan but not necessarily for continuations into suboptimal plans.

If a formula is feasible for a node, its progression is feasible for the corresponding successor node.

Theorem 1. *Let φ be a feasible formula for a node n, and let n' be the successor node reached from n with action a. Then progression$(\varphi, \{a\} \cup s(n'))$ is feasible for n'.*

Proof. Let R_a be the set of paths that continue the history of n to an optimal plan (and are therefore relevant for the feasibility of φ) and in addition begin with action a. If R_a is empty, the path leading to n' cannot be continued to an optimal plan and therefore any formula is feasible for n'. Otherwise let ρ' be a path that continues the path to n' to an optimal plan. Then $\rho = \langle a\rangle\rho'$ is in R_a and $\boldsymbol{w}^{\boldsymbol{s(n)}}_{\boldsymbol{\rho}} \models \varphi$. Since $\boldsymbol{w}^{\boldsymbol{s(n)}}_{\boldsymbol{\rho}} = \langle\{a\} \cup s(n')\rangle\boldsymbol{w}^{\boldsymbol{s(n')}}_{\boldsymbol{\rho'}}$, Proposition 1 implies that $\boldsymbol{w}^{\boldsymbol{s(n')}}_{\boldsymbol{\rho'}} \models \text{progression}(\varphi, \{a\} \cup s(n'))$. □

We also have the opportunity to incorporate additional feasible trajectory constraints: if we can derive a new feasible formula φ_{new} for a node, we can safely combine it with the progressed formula $\varphi_{\text{progress}}$ to a feasible formula $\varphi_{\text{progress}} \wedge \varphi_{\text{new}}$.

Since graph search algorithms eliminate nodes with duplicate states, a strategy for combining feasible formulas from nodes with identical state is desirable. Instead of only keeping the formula of the preserved node, we can exploit the information of two paths of equal cost by combining the two feasible formulas:

Theorem 2. *Let n and n' be two search nodes such that $g(n) = g(n')$ and $s(n) = s(n')$. Let further φ_n and $\varphi_{n'}$ be feasible for the respective node. Then $\varphi_n \wedge \varphi_{n'}$ is feasible for both n and n'.*

Proof sketch. Whether a formula is feasible for a node only depends on the set of relevant paths characterized in Definition 2. The node only influences this characterization with its g-value and its associated state s. Therefore, a formula is either feasible for *all* nodes that agree on these components or for *none* of them. Feasibility of the conjunction follows directly from the LTL$_f$ semantics. □

Feasible formulas for a node only talk about the future but do not cover the current state. This can lead to an needlessly complicated specification of information that is derived before starting the search. For example, the notion of landmarks is defined for the entire state sequence traversed by a plan – including the initial state. We therefore also introduce *feasibility for tasks* that allows to naturally formulate information about this entire state sequence.

Definition 3 (Feasibility for tasks)**.** *Let Π be a planning task with initial state I. An LTL$_f$ formula φ is* feasible *for Π if $\langle I\rangle\boldsymbol{w}^{\boldsymbol{I}}_{\boldsymbol{\pi^*}} \models \varphi$ for all optimal plans π^*.*

Progressing a feasible formula for a task with its initial state yields a feasible formula for the initial search node.

Finding Feasible LTL$_f$ Trajectory Constraints

In this section we will demonstrate how existing examples from the literature can be used to derive feasible LTL$_f$ trajectory constraints. We will present details for landmarks and unjustified action applications. To give an intuition for the scope of the framework, we will also briefly comment on further possibilities.

Fact Landmarks and Landmark Orderings

A (fact) landmark (Hoffmann, Porteous, and Sebastia 2004) is a state variable that must be true at some point in every plan. A landmark ordering specifies that before a landmark becomes true some other landmark must have been true.

There are several methods in the literature to extract such landmark information for a given planning task (Zhu and Givan 2003; Hoffmann, Porteous, and Sebastia 2004; Richter and Westphal 2010; Keyder, Richter, and Helmert 2010).

Wang, Baier, and McIlraith (2009) already encoded landmark orderings in LTL. Since in our scenario, progression notices when a landmark has been reached, we can use a slightly more compact formalization.

We base our definitions of landmarks and landmark orderings on the notions by Richter and Westphal (2010).

A state variable p is a *(fact) landmark* of a planning task Π if the application of every plan π of Π visits some state

that contains p. Therefore, $\Diamond p$ is a feasible formula for the task if p is a landmark.

There are three types of sound landmark orderings.

A *natural* ordering $l \rightarrow_{\text{nat}} l'$ states that if l' becomes true for the first time at time step i, then l must be true at some time step $j < i$. This can be expressed as $\neg l' \mathcal{U}(l \wedge \neg l')$ because l' can only become true after l was true.

A *greedy-necessary* ordering $l \rightarrow_{\text{gn}} l'$ states that if l' becomes true for the first time at step i, then l must be true at time step $i-1$. This can be formulated as $\neg l' \mathcal{U}(l \wedge \neg l' \wedge \bigcirc l')$ because l must be true directly before l' becomes true for the first time.

A *necessary* ordering $l \rightarrow_{\text{nec}} l'$ states that for each time step i where l' becomes true, l must be true at time step $i-1$. Put differently, whenever l' becomes true in the next step, then currently l must be true: $\Box(\neg l' \wedge \bigcirc l' \rightarrow l)$.

There are important other landmark-related notions for which we are not aware of any previous LTL encodings.

One such relevant source of information are the first achievers of a landmark. A *first achiever set* FA_l for a landmark l is a set of actions such that in any plan one of these actions makes the landmark true for the first time. For a first achiever set FA_l the formula $l \vee \bigvee_{a \in FA_l} \Diamond a$ describes that if the landmark is not initially true, then one of the actions in the set needs to be applied.

The landmark count heuristic (Richter and Westphal 2010) counts how many landmarks have not yet been reached (i. e., they have not been true on the path to the node) and how many reached landmarks are required again. A reached landmark l is required again if l is false in the current state and there is a greedy-necessary (or necessary) ordering $l \rightarrow l'$ where l' has not yet been reached. Additionally, any reached landmark l that is a goal proposition and false in the current state is required again.

We can formulate these conditions for required again landmarks as $(\Diamond l)\mathcal{U}l'$ for greedy-necessary orderings $l \rightarrow_{\text{gn}} l'$ and as $(\Diamond g)\mathcal{U} \bigwedge_{g' \in G} g'$ for goal landmarks g and goal specification G.

All these formulas can be combined into a single LTL_f formula that is feasible for the task:

Proposition 2. *For planning task* Π *with goal* G*, let* L *be a set of landmarks and let* O_{nat}, O_{gn}*, and* O_{nec} *be sets of natural, greedy-necessary and necessary landmark orderings, respectively. Let FA be a set of first achiever sets for landmarks. The following formula* φ *is feasible for* Π*.*

$$\varphi = \varphi_{lm} \wedge \varphi_{fa} \wedge \varphi_{nat} \wedge \varphi_{gn} \wedge \varphi_{nec} \wedge \varphi_{ra} \wedge \varphi_{goal}$$

where

- $\varphi_{lm} = \bigwedge_{l \in L} \Diamond l$
- $\varphi_{fa} = \bigwedge_{FA_l \in FA} \left(l \vee \bigvee_{a \in FA_l} \Diamond a\right)$
- $\varphi_{nat} = \bigwedge_{l \rightarrow_{nat} l' \in O_{nat}} \left(\neg l' \mathcal{U}(l \wedge \neg l')\right)$
- $\varphi_{gn} = \bigwedge_{l \rightarrow_{gn} l' \in O_{gn}} \left(\neg l' \mathcal{U}(l \wedge \neg l' \wedge \bigcirc l')\right)$
- $\varphi_{nec} = \bigwedge_{l \rightarrow_{nec} l' \in O_{nec}} \Box(\neg l' \wedge \bigcirc l' \rightarrow l)$
- $\varphi_{ra} = \bigwedge_{l \rightarrow l' \in O_{gn} \cup O_{nec}} \left((\Diamond l)\mathcal{U}l'\right)$
- $\varphi_{goal} = \bigwedge_{g \in G} \left((\Diamond g)\mathcal{U} \bigwedge_{g' \in G} g'\right)$

The subformula $\varphi_{\text{lm}} \wedge \varphi_{\text{ra}} \wedge \varphi_{\text{goal}}$ also provides an alternative way of computing the inadmissible landmark count heuristic, which can only be used for planning without optimality guarantee: We determine this task-feasible formula from the same precomputation as performed by the landmark count heuristic and progress it in our framework. The heuristic estimate for a node can then be determined from the associated feasible formula as the cardinality of the set of all state variables that occur within a $\Diamond$ formula but are false in the state of the node.

Action Landmarks

A *(disjunctive) action landmark* is a set of actions of which at least one must occur in any plan.

If $\mathcal{L}$ is a set of action landmarks for state s' then $\varphi_{\mathcal{L}} = \bigwedge_{L \in \mathcal{L}} \Diamond(\bigvee_{a \in L} a)$ is feasible for all nodes n with $s(n) = s'$.

One example for an action landmark is the set of first achievers for a landmark that is currently not true. Also the LM-Cut heuristic (Helmert and Domshlak 2009) derives a set of action landmarks as a byproduct of the heuristic computation. The *incremental* LM-Cut heuristic (Pommerening and Helmert 2013) stores and progresses this set to speed up the LM-Cut computation for the successor states. At the application of an action a, incremental LM-Cut creates a new landmark set for the successor state by removing all action landmarks that contain a. Let $\mathcal{L}$ and $\mathcal{L}'$ be the respective landmark sets before and after the action application. The progression of $\varphi_{\mathcal{L}}$ over a is logically equivalent to $\varphi_{\mathcal{L}'}$, so LTL_f progression reflects the landmark-specific progression by incremental LM-Cut.

Unjustified Action Applications

The key idea behind *unjustified action applications* (Karpas and Domshlak 2011; 2012) is that every action that occurs in a plan should contribute to the outcome of the plan – by enabling another action in the plan or by making a goal proposition finally true.

The definition is based on the notion of *causal links*: a path $\rho = \langle a_1, \ldots, a_n \rangle$ has a *causal link* between the i-th and the j-th action application if $i < j$, a_i adds a proposition p which stays true and is not added again until step $j-1$, and p is a precondition of a_j. If there is a link between the i-th and a later action application in a plan π or the i-th action adds a goal proposition that is not added again later, then the i-th action application is *justified*, otherwise it is *unjustified*.

A plan π with an unjustified application of a positive-cost action a cannot be optimal because removing this action application from π results in a cheaper plan π'.

The notion of unjustified action applications is defined for *entire plans* but during search the question is whether the current path can be extended to a plan without unjustified action applications and how we can characterize such an extension. The relevant information can easily be encoded within the LTL_f framework: if the last action application is justified, at least one of the add effects must stay true and cannot be added again until it is used as a precondition or for satisfying the goal. Since we want to preserve *all* optimal plans, we do not exploit this information for zero-cost actions.

Theorem 3. *Let $\Pi = \langle V, A, I, G\rangle$ be a planning task and let n be a search node that was reached with a positive-cost action a. Then the following formula φ_a is feasible for n:*

$$\varphi_a = \bigvee_{e \in add(a) \setminus G} \Big((e \wedge \bigwedge_{\substack{a' \in A \text{ with} \\ e \in add(a')}} \neg a') \mathcal{U} \bigvee_{\substack{a' \in A \text{ with} \\ e \in pre(a')}} a' \Big) \vee \bigvee_{e \in add(a) \cap G} \Big((e \wedge \bigwedge_{\substack{a' \in A \text{ with} \\ e \in add(a')}} \neg a') \mathcal{U} \big(\mathit{last} \vee \bigvee_{\substack{a' \in A \text{ with} \\ e \in pre(a')}} a' \big) \Big)$$

Proof sketch. Let ρ denote the path that lead to n. Assume that φ_a is not feasible for n, so there is a path ρ' that satisfies the conditions from Definition 2 but $\boldsymbol{w}_{\rho'}^{\boldsymbol{s(n)}} \not\models \varphi_a$. Then ρ' does not use any add effect of a before it gets deleted or added again by another action application. Also each goal atom added by a is deleted or added again by ρ'. Therefore the application of a in the plan $\rho\rho'$ is unjustified. As $c(a) > 0$ there exists a cheaper plan and $g(n)+c(\rho')$ cannot be equal to the optimal plan cost of Π. This is a contradiction to ρ' satisfying the conditions from Definition 2. □

The original implementation of unjustified action applications requires an analysis of the causal links and resulting causal chains of the action sequence. All information required for this reasoning is encoded in the LTL$_f$ formulas and standard LTL$_f$ progression replaces this analysis of the causal interactions.

We could even go further: Instead of adding a feasible formula φ_a after each application of action a, we could also extend the feasible formula for the initial node with a conjunction $\bigwedge_{a \in A} a \rightarrow \bigcirc \varphi_a$, ranging over the set of all actions A, and let the progression do the rest. However, since planning tasks can have thousands of actions, this would lead to significant overhead in the progression.

Other Sources of Information

The scope of our approach is by far not limited to the previously introduced sources of information. Since space is limited, we only briefly mention some other ideas.

One obvious possibility is the integration of previous LTL methods like hand-written (Bacchus and Kabanza 2000; Doherty and Kvarnström 2001) or learned (de la Rosa and McIlraith 2011) LTL search control knowledge.

Also invariants such as mutex information can be added to the LTL$_f$ formula to make them explicit to the heuristic computation. The same holds for the fact that at the end the goal G must be true ($\Diamond(\mathit{last} \wedge \bigwedge_{g \in G} g)$).

The recent flow-based heuristics (van den Briel et al. 2007; Bonet 2013; Pommerening et al. 2014) build on the observation that a variable cannot be (truly) deleted more often than it is made true (with some special cases for the initial state and the goal) but they ignore the order of the corresponding action applications. With LTL$_f$ formulas we can express statements of the same flavor but preserving parts of the ordering information. For an intuition, consider a formula that states that whenever we apply an action deleting p and later apply an action requiring p, we in between have to apply an action adding p.

In principle, we could go as far as encoding the entire planning task in the LTL formula (Cerrito and Mayer 1998). The challenge with the framework will be to find a suitable balance of the incurred overhead and the gain in heuristic information.

Deriving Heuristic Estimates from Feasible LTL$_f$ Trajectory Constraints

Deriving heuristic estimates from feasible LTL$_f$ trajectory constraints is an interesting research question, which cannot finally be answered in this paper. For the moment, we only present a proof-of-concept heuristic that extracts landmarks, essentially ignoring the temporal information carried by the LTL$_f$ formula. However, the temporal aspects of the LTL$_f$ formulas are still important for the heuristic estimate because they preserve information over the course of progression. In the future we also want to investigate methods that are based on LTL reasoning and which are therefore able to derive stronger estimates from the temporal information.

Although we extract landmarks from the input LTL$_f$ formula, this does not mean that this LTL$_f$ formula must stem from landmark information. The heuristic is correct for any kind of feasible LTL$_f$ formulas.

The heuristic computation first derives so-called *node-admissible disjunctive action landmarks* from the LTL$_f$ formula. The heuristic estimate is then determined from these landmarks with the landmark heuristic by Karpas and Domshlak (2009).

As introduced earlier, a disjunctive action landmark for a state s is a set of actions such that *every* path from s to a goal state contains at least one action from the set. We use a weaker path-dependent notion that covers all *optimal* plans:

Definition 4. *Let $\Pi = \langle V, A, I, G\rangle$ be a planning task and n be a search node. A set $L \subseteq A$ is a* node-admissible disjunctive action landmark *for n if every continuation of the path to n into an optimal plan contains an action from L.*

Using such node-admissible disjunctive action landmarks in the landmark heuristic gives admissible estimates for all nodes that correspond to a prefix of an optimal plan. Karpas and Domshlak (2012) call this property *path admissible*.

Our method of extracting node-admissible landmarks from LTL$_f$ formulas requires the formula to be in positive normal form (also called negation normal form), where $\neg$ only appears in literals or before *last*. This is uncritical because any LTL$_f$ formula can efficiently be transformed into positive normal form with De Morgan's law and the following equivalences:

$$\neg\bigcirc\varphi \equiv \mathit{last} \vee \bigcirc\neg\varphi \qquad \neg\Box\varphi \equiv \Diamond\neg\varphi$$
$$\neg(\varphi_1 \mathcal{U} \varphi_2) \equiv (\neg\varphi_1)\mathcal{R}(\neg\varphi_2) \qquad \neg\Diamond\varphi \equiv \Box\neg\varphi$$
$$\neg(\varphi_1 \mathcal{R} \varphi_2) \equiv (\neg\varphi_1)\mathcal{U}(\neg\varphi_2)$$

Moreover, progression preserves the normal form.

For the landmark extraction from the feasible formula, we first derive an implied LTL$_f$ formula (Proposition 3) in conjunctive normal form. We then extract node-admissible disjunctive action landmarks from its clauses (Theorem 4).

Proposition 3. *Let φ be an LTL$_f$ trajectory constraint in negation normal form. The following function lm defines an LTL$_f$ formula such that $\varphi \models lm(\varphi)$:*

$$\begin{aligned} lm(x) &= \Diamond x \text{ for literals } x \\ lm(last) &= \Diamond last \\ lm(\neg last) &= \Diamond \neg last \\ lm(\varphi \wedge \psi) &= lm(\varphi) \wedge lm(\psi) \\ lm(\varphi \vee \psi) &= lm(\varphi) \vee lm(\psi) \\ lm(\bigcirc \varphi) &= lm(\Box \varphi) = lm(\Diamond \varphi) = lm(\varphi) \\ lm(\varphi \mathcal{U} \psi) &= lm(\varphi \mathcal{R} \psi) = lm(\psi) \end{aligned}$$

The proposition can easily be checked from the semantics of LTL$_f$. By distributing $\vee$ over $\wedge$, we can transform the formula into *conjunctive normal form* (CNF) $\bigwedge_{i=1}^{n} \bigvee_{j=1}^{m_i} \Diamond \varphi_{i,j}$, where each $\varphi_{i,j}$ is a literal, *last*, or $\neg$*last*. Clauses containing $\Diamond$*last* are tautologies and therefore not valuable for the heuristic. Clauses containing $\Diamond\neg$*last* are trivially true for each world sequence of length at least two, which is the case for the induced world sequences for any path leading from a non-goal state to a goal state. Therefore, we derive the action landmarks only from the remaining clauses.

Theorem 4. *Let $\Pi = \langle V, A, I, G \rangle$ be a planning task and let φ be a feasible LTL$_f$ trajectory constraint for node n. Let further $\psi = \bigvee_{j=1}^{m} \Diamond x_j$ (with x_j being literals) be a formula such that $\varphi \models \psi$.*

If progress$(\psi, s(n)) \not\equiv \top$, then $L = \bigcup_{j=1}^{m}$ support(x_j) with

$$support(x) = \begin{cases} \{a \in A \mid x \in add(a)\} & \text{if } x \in V \\ \{a \in A \mid x \in del(a)\} & \text{if } \bar{x} \in V \\ \{x\} & \text{if } x \in A \end{cases}$$

is a node-admissible disjunctive action landmark for n.

Proof. Since $\varphi \models \psi$, ψ also is feasible for n. By the semantics of $\Diamond$, at least one x_j must be true in some world in any continuation to an optimal plan. If the progression is not valid, then no state-variable literal x_j is already true in the current state.[2] Thus, one of the x_j needs to become true in any optimal plan. For a proposition p, this means that p must be added by an action. Similarly, for a negated proposition $\neg p$, the proposition p must be deleted. An action variable a requires the action to be applied. Therefore, any continuation of the path to n into an optimal plan contains an action from L. □

Our proposed heuristic is the landmark heuristic computed from all node-admissible disjunctive action landmarks that can be derived from the trajectory constraints as described in Proposition 3 and Theorem 4. In the special case where the LTL$_f$ formula is detected unsatisfiable (simplifies to $\bot$), the heuristic returns ∞ because the path to the node cannot be extended into an optimal plan.

[2]Moreover, none of the x_j is a negated action variable. Although these would not prevent the clause from generating an action landmark, the landmark would be very weak.

A* with LTL$_f$ Trajectory Constraints

Since LTL$_f$ trajectory constraints are path-dependent and the heuristic is not admissible but only path admissible, we need to adapt the A* algorithm so that it still guarantees optimal plans. This is only a small modification: whenever a cheaper path to a state has been found, we need to recompute the heuristic estimate with the new feasible information instead of using a cached estimate (Karpas and Domshlak 2012). This leads to a different treatment of infinite heuristic estimates. We also use the opportunity to show the integration of feasible LTL$_f$ formulas.

Algorithm 1 shows the adapted A* algorithm. We formulated the algorithm with "eager" duplicate elimination so that there is always at most one node for each state. Besides the planning task, the algorithm takes a path admissible heuristic function as input that computes the estimate from a state and an LTL$_f$ formula.

We use a method *taskFeasibleFormula*(Π) that returns a feasible formula for Π, and a method *feasibleFormula*(Π, π) that generates a (path-dependent) feasible formula for the node reached by path π.

The feasible formula for the task (line 3) is progressed with the initial state to receive a feasible formula φ for the initial search node, which can be further strenghened with any other feasible formula for this node (line 4). If the heuristic for the initial state and this formula is ∞ then the task is unsolvable. Otherwise, we create the initial node and add it to the open list (lines 6–8).

When generating the successor n' of node n with action a, we first progress the LTL$_f$ formula of n to get a new feasible formula for n', based on Theorem 1 (line 17). Based on Theorem 2, this formula can be strengthened with a newly derived LTL$_f$ formula that is feasible for this node (line 18). If we do not want to incorporate additional information, the method can simply return $\top$.

If we encounter the successor state for the first time, we create a new node n' with the respective properties (line 21). Otherwise, we distinguish three cases for the previously best node for this state. If we already found a better path to this state, we skip this successor (line 24). If we previously found an equally good path to the state, we strengthen the LTL$_f$ formula of the previous node based on Theorem 2 and recompute the heuristic estimate (lines 26–28). If the previously best path to the state was worse, we update the node with the data from the newly found path (lines 30–33).

In contrast to an admissible heuristic, with a path admissible heuristic it is not safe to prune a *state* from the search space if the heuristic estimate via one path (of cost g') is ∞. However, we can conclude that no optimal plan traverses the state with a cost of at least g'. To exploit this pruning power for nodes encountered later, we do not discard a node with infinite heuristic estimate but store it in the closed list (lines 35–36). If the node has a finite estimate, it gets enqueued in the open list (line 38).

Experimental Evaluation

For the experimental evaluation, we implemented the LTL$_f$ framework on top of Fast Downward (Helmert 2006). We

Algorithm 1: A* with LTL$_f$ trajectory constraints

input : Planning task $\Pi = \langle V, A, I, G\rangle$ and path-admissible heuristic function h
output: Optimal plan π or *unsolvable* if Π is unsolvable

1 *open* $\leftarrow$ empty priority queue of nodes
2 *closed* $\leftarrow \emptyset$
3 $\varphi_\Pi \leftarrow$ *taskFeasibleFormula*(Π)
4 $\varphi \leftarrow$ *progress*$(\varphi_\Pi, I) \wedge$ *feasibleFormula*$(\Pi, \langle\rangle)$
5 *hval* $\leftarrow h(I, \varphi)$
6 **if** *hval* $\neq \infty$ **then**
7 $\quad n \leftarrow$ new node with $n.state = I$, $n.g = 0$, $n.h =$ *hval*, $n.\varphi = \varphi$ and $n.parent = \perp$
8 $\quad$ add n to *open* with priority *hval*
9 **while** *open is not empty* **do**
10 $\quad n \leftarrow$ remove min from open
11 $\quad s \leftarrow n.state$
12 $\quad$ **if** s *is goal state* **then**
13 $\quad\quad$ **return** *extractPlan*(n)
14 $\quad$ add n to *closed*
15 $\quad$ **for** *all actions* a *applicable in* s **do**
16 $\quad\quad s' \leftarrow s[a]$
17 $\quad\quad \varphi' \leftarrow$ *progress*$(n.\varphi, \{a\} \cup s')$
18 $\quad\quad \varphi' \leftarrow \varphi' \wedge$ *feasibleFormula*$(\Pi,$ path to s' via $n)$
19 $\quad\quad g' \leftarrow n.g + c(a)$
20 $\quad\quad$ **if** *there exists no node* n' *with* $n'.state = s'$ **then**
21 $\quad\quad\quad n' \leftarrow$ new node with $n'.state = s'$, $n'.g = g'$, $n'.h = h(s', \varphi')$, $n'.\varphi = \varphi'$ and $n'.parent = n$
22 $\quad\quad$ **else**
23 $\quad\quad\quad n' \leftarrow$ unique node in *open* or *closed* with $n'.state = s'$
24 $\quad\quad\quad$ **if** $g' > n'.g$ **then continue**
25 $\quad\quad\quad$ remove n' from *open*/*closed*
26 $\quad\quad\quad$ **if** $g' = n'.g$ **then**
27 $\quad\quad\quad\quad n'.\varphi \leftarrow n'.\varphi \wedge \varphi'$
28 $\quad\quad\quad\quad n'.h \leftarrow h(s', n'.\varphi)$
29 $\quad\quad\quad$ **else**
30 $\quad\quad\quad\quad n'.\varphi \leftarrow \varphi'$
31 $\quad\quad\quad\quad n'.g \leftarrow g'$
32 $\quad\quad\quad\quad n'.h \leftarrow h(s', \varphi')$
33 $\quad\quad\quad\quad n'.parent \leftarrow n$
34 $\quad\quad$ **end**
35 $\quad\quad$ **if** $n'.h = \infty$ **then**
36 $\quad\quad\quad$ add n' to *closed*
37 $\quad\quad$ **else**
38 $\quad\quad\quad$ add n' to *open* with priority $n'.g + n'.h$
39 $\quad\quad$ **end**
40 $\quad$ **end**
41 **end**
42 **return** unsolvable

conducted all experiments with a memory limit of 4 GB and a time limit of 30 minutes (excluding Fast Downward's translation and preprocessing phase).

Our heuristic derives disjunctive action landmarks from LTL$_f$ constraints as input for the admissible landmark heuristic (Karpas and Domshlak 2009). To evaluate the overhead of our approach, we compare it to the standard implementation of the landmark heuristic with specialized data structures exploiting the same (initial) landmark information. In both cases, the landmark heuristic applies optimal cost-partitioning for computing the estimate.

For the landmark generation, we use the same approach as the BJOLP planner (Domshlak et al. 2011), combining landmark information from two generation methods (Richter and Westphal 2010; Keyder, Richter, and Helmert 2010).

For the LTL$_f$ approach, we generate an initial feasible LTL$_f$ formula from the first achiever and the required again formulas (corresponding to $\varphi_{fa}, \varphi_{ra}$, and φ_{goal} in Proposition 2) for these landmarks. We do not use the landmark and ordering formulas because they would not additionally contribute to the heuristic estimates. We use this LTL$_f$ formula as shown in Algorithm 1, not incorporating additional information in line 18. In the following, we refer to this setup as LTL-A* with $h^{\text{LM}}_{\text{AL}}$.

A meaningful application of the standard implementation of the landmark heuristic requires the search algorithm LM-A* (Karpas and Domshlak 2009) that extends A* with multi-path support for landmarks. We refer to this setup as LM-A* with h_{LA}.

The comparison of these two approaches is not entirely fair because LM-A* combines information from *all* paths to a state while LTL-A* only combines formulas from equally expensive paths. Thus, with a comparable search history, LM-A* can sometimes derive better heuristic estimates.

Table 1 shows results for the STRIPS benchmarks of the International Planning Competitions 1998–2011.

Overall, LM-A* with h_{LA} solves 723 tasks and LTL-A* with $h^{\text{LM}}_{\text{AL}}$ finds solutions for 711 tasks. All unsolved instances of the former are due to the time limit. With the LTL$_f$ implementation 11 instances fail due to the memory limit with 9 of them being airport instances.

To get a clearer idea of the memory overhead of the approach, we summed up the memory consumption of all commonly solved instances of each domain. The percentage in parentheses shows the fraction of these two values where numbers above 100% indicate that the LTL$_f$ approach required more memory. A positive surprise is that more often than not our approach requires less memory. However, there are also cases, where the increase in memory consumption is significant, for example in the logistics-00 domain where the LTL$_f$ implementation needs more than three times the amount of the specialized implementation. This result is not caused by the unfavorable comparison of the approaches because the expansion numbers in both cases are identical. Nevertheless, the memory consumption only is responsible for a single unsolved task in this domain because 7 of the 8 affected instances fail due to a timeout.

	LM-A*	LTL-A*		
	h_{LA}	$h_{\text{AL}}^{\text{LM}}$		$h_{\text{AL}}^{\text{LM+UAA}}$
airport (50)	31	28	(335%)	26
barman (20)	0	0	(−%)	0
blocks (35)	26	26	(107%)	26
depot (22)	7	7	(86%)	7
driverlog (20)	14	14	(88%)	14
elevators-08 (30)	14	14	(78%)	13
elevators-11 (20)	11	11	(77%)	11
floortile (20)	2	2	(95%)	4
freecell (80)	52	51	(123%)	50
grid (5)	2	2	(108%)	2
gripper (20)	6	6	(187%)	6
logistics-00 (28)	20	20	(327%)	20
logistics-98 (35)	5	5	(99%)	5
miconic (150)	141	141	(116%)	141
mprime (35)	19	19	(90%)	20
mystery (30)	15	15	(83%)	15
nomystery (20)	18	17	(147%)	16
openstacks-08 (30)	14	12	(200%)	12
openstacks-11 (20)	9	7	(229%)	7
openstacks (30)	7	7	(107%)	7
parcprinter-08 (30)	15	14	(149%)	14
parcprinter-11 (20)	11	10	(152%)	10
parking (20)	1	1	(121%)	1
pathways (30)	4	4	(98%)	4
pegsol-08 (30)	26	26	(155%)	26
pegsol-11 (20)	16	16	(174%)	16
pipesworld-notan (50)	17	17	(91%)	17
pipesworld-tan (50)	9	10	(98%)	10
psr-small (50)	49	49	(87%)	49
rovers (40)	7	7	(91%)	7
satellite (36)	7	7	(86%)	7
scanalyzer-08 (30)	10	9	(111%)	9
scanalyzer-11 (20)	6	6	(114%)	6
sokoban-08 (30)	22	21	(76%)	22
sokoban-11 (20)	18	18	(76%)	18
tidybot (20)	14	14	(103%)	13
tpp (30)	6	6	(95%)	6
transport-08 (30)	11	11	(90%)	11
transport-11 (20)	6	6	(86%)	6
trucks (30)	7	7	(82%)	7
visitall (20)	16	16	(136%)	16
woodworking-08 (30)	14	14	(80%)	14
woodworking-11 (20)	9	9	(78%)	9
zenotravel (20)	9	9	(93%)	9
Sum (1396)	723	711		709

Table 1: Results for LM-A* with the standard landmark heuristic and for LTL-A* using a feasible landmark-based constraint ($h_{\text{AL}}^{\text{LM}}$) and using additional feasible LTL$_f$ constraints from unjustified action applications ($h_{\text{AL}}^{\text{LM+UAA}}$). The percentage in parentheses shows the memory consumption on the commonly solved instances compared to the first configuration. All other numbers show coverage results.

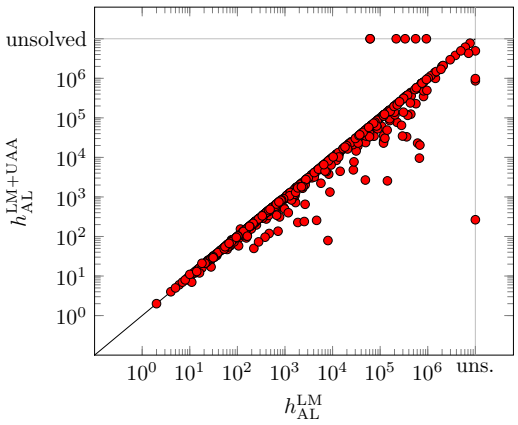


Figure 2: Comparison of expansions.

In a second experiment, we include in addition feasible LTL$_f$ trajectory constraints as described in the section on unjustified action applications. We denote the resulting heuristic $h_{\text{AL}}^{\text{LM+UAA}}$. Coverage results are shown in the last column of Table 1. Overall, the inclusion of the additional feasible information leads to two fewer solved instances. However, there are also domains where the coverage increases. One main reason for failure is a higher memory consumption leading to 83 instances that failed due to the memory limit. Another reason is a time overhead that leads to 13.4% fewer evaluations per second on the commonly solved instances. On the positive side, the heuristic is indeed better informed which translates to a reduction in the number of expanded nodes (cf. Figure 2).

Conclusion

We propose a clean and general LTL$_f$ framework for optimal planning that is easy to prove correct. It is based on a feasibility criterion for LTL$_f$ trajectory constraints and there are plenty of possibilities to derive such feasible constraints from established planning methods.

We presented a baseline heuristic from such constraints, based on the extraction of disjunctive action landmarks. This heuristic does not yet fully exploit the potential of the approach because it does not consider the temporal information of the constraints. We plan to change this in future work where we will to a greater extend exploit LTL$_f$ reasoning methods in the heuristic computation. We also will investigate the potential for strengthening other heuristic computations with the information from LTL$_f$ trajectory constraints, similar to what Wang, Baier, and McIlraith (2009) have done with landmark orderings and the FF heuristic. Another research direction will be the examination of further sources of information and of possible positive interactions of their combination.

Acknowledgments

This work was supported by the European Research Council as part of the project "State Space Exploration: Principles, Algorithms and Applications".

References

Bacchus, F., and Kabanza, F. 1998. Planning for temporally extended goals. *Annals of Math. and AI* 22(1,1):5–27.

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

Bonet, B. 2013. An admissible heuristic for SAS$^+$ planning obtained from the state equation. In *Proc. IJCAI 2013*, 2268–2274.

Calvanese, D.; De Giacomo, G.; and Vardi, M. Y. 2002. Reasoning about actions and planning in LTL action theories. In *Proc. KR 2002*, 593–602.

Cerrito, S., and Mayer, M. C. 1998. Using linear temporal logic to model and solve planning problems. In Giunchiglia, F., ed., *Artificial Intelligence: Methodology, Systems, and Applications (AIMSA 98)*, volume 1480 of *LNCS*, 141–152. Springer-Verlag.

De Giacomo, G., and Vardi, M. Y. 2013. Linear temporal logic and linear dynamic logic on finite traces. In *Proc. IJCAI 2013*, 854–860.

De Giacomo, G.; De Masellis, R.; and Montali, M. 2014. Reasoning on LTL on finite traces: Insensitivity to infiniteness. In *Proc. AAAI 2014*, 1027–1033.

de la Rosa, T., and McIlraith, S. 2011. Learning domain control knowledge for TLPlan and beyond. In *ICAPS 2011 Workshop on Planning and Learning*, 36–43.

Doherty, P., and Kvarnström, J. 2001. TALplanner: A temporal logic based planner. *AI Magazine* 22(3):95–102.

Domshlak, C.; Helmert, M.; Karpas, E.; Keyder, E.; Richter, S.; Röger, G.; Seipp, J.; and Westphal, M. 2011. BJOLP: The big joint optimal landmarks planner. In *IPC 2011 planner abstracts*, 91–95.

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

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In *Proc. ICAPS 2009*, 162–169.

Helmert, M. 2006. The Fast Downward planning system. *JAIR* 26:191–246.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *JAIR* 14:253–302.

Hoffmann, J.; Porteous, J.; and Sebastia, L. 2004. Ordered landmarks in planning. *JAIR* 22:215–278.

Kabanza, F., and Thiébaux, S. 2005. Search control in planning for temporally extended goals. In *Proc. ICAPS 2005*, 130–139.

Karpas, E., and Domshlak, C. 2009. Cost-optimal planning with landmarks. In *Proc. IJCAI 2009*, 1728–1733.

Karpas, E., and Domshlak, C. 2011. Living on the edge: Safe search with unsafe heuristics. In *ICAPS 2011 Workshop on Heuristics for Domain-Independent Planning*, 53–58.

Karpas, E., and Domshlak, C. 2012. Optimal search with inadmissible heuristics. In *Proc. ICAPS 2012*, 92–100.

Keyder, E.; Richter, S.; and Helmert, M. 2010. Sound and complete landmarks for and/or graphs. In *Proc. ECAI 2010*, 335–340.

Koehler, J., and Treinen, R. 1995. Constraint deduction in an interval-based temporal logic. In Fisher, M., and Owens, R., eds., *Executable Modal and Temporal Logics*, volume 897 of *LNCS*, 103–117. Springer-Verlag.

Pnueli, A. 1977. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FOCS 1977)*, 46–57.

Pommerening, F., and Helmert, M. 2013. Incremental LM-cut. In *Proc. ICAPS 2013*, 162–170.

Pommerening, F.; Röger, G.; Helmert, M.; and Bonet, B. 2014. LP-based heuristics for cost-optimal planning. In *Proc. ICAPS 2014*, 226–234.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *JAIR* 39:127–177.

Simon, S., and Röger, G. 2015. Finding and exploiting LTL trajectory constraints in heuristic search. In *Proc. SoCS 2015*. To appear.

van den Briel, M.; Benton, J.; Kambhampati, S.; and Vossen, T. 2007. An LP-based heuristic for optimal planning. In *Proc. CP 2007*, 651–665.

Wang, L.; Baier, J.; and McIlraith, S. 2009. Viewing landmarks as temporally extended goals. In *ICAPS 2009 Workshop on Heuristics for Domain-Independent Planning*, 49–56.

Zhu, L., and Givan, R. 2003. Landmark extraction via planning graph propagation. In *ICAPS 2003 Doctoral Consortium*, 156–160.

Simulation-Based Admissible Dominance Pruning

Álvaro Torralba and **Jörg Hoffmann**
Saarland University
Saarbrücken, Germany
torralba@cs.uni-saarland.de, hoffmann@cs.uni-saarland.de

Abstract

In optimal planning as heuristic search, admissible pruning techniques are paramount. One idea is *dominance pruning*, identifying states "better than" other states. Prior approaches are limited to simple dominance notions, like "more STRIPS facts true" or "higher resource supply". We apply *simulation*, well-known in model checking, to compute much more general dominance relations based on comparing transition behavior across states. We do so effectively by expressing state-space simulations through the *composition* of simulations on orthogonal projections. We show how simulation can be made more powerful by intertwining it with a notion of *label dominance*. Our experiments show substantial improvements across several IPC benchmark domains.

Introduction

Heuristic search is the predominant approach to cost-optimal planning. But the number of states that must be explored to prove optimality often grows exponentially even when using extremely well-informed heuristics (Helmert and Röger 2008). Therefore, recent years have seen substantial effort devoted to identifying and exploiting structure allowing to prune redundant parts of the state space. Known techniques of this kind pertain to *symmetries* (e. g. (Fox and Long 1999; 2002; Domshlak, Katz, and Shleyfman 2012)), *partial-order reduction* based methods like expansion-core (Chen and Yao 2009) or strong stubborn sets (Valmari 1989; Wehrle and Helmert 2012; Wehrle et al. 2013; Wehrle and Helmert 2014), and *dominance pruning* (Hall et al. 2013). We follow up on the latter here.

Dominance pruning is based on identifying states "better than" other states. For example, consider a Logistics task where one truck must carry several packages to location G. Consider the position of any one package p. All other state variables having equal values, the best is to have p at G, and it is better for p to be in the truck than at any location other than G. We refer to this kind of relation between states as a *dominance relation*. (Hall et al. use the term "partial-order", which we change here to avoid ambiguity with, e. g., partial-order reduction.)

Two main questions need to be answered: *(1) How to discover the dominance relation? (2) How to simplify the search (and/or planning task) given a dominance relation?* Hall et al. answered (2) in terms of an admissible pruning method, pruning state s if a dominating state t, with an at-most-as-costly path, has already been seen. We follow that idea here, contributing a BDD implementation. Our main contribution regards (1). Hall et al. use dominance relations characterized by consumed resources: state t dominates state s if s and t are identical except that $t(r) \geq s(r)$ for all resources r.[1] Herein, we instead find the dominance relation through *simulation*, used in model checking mainly to compare different system models (Milner 1971; Gentilini, Piazza, and Policriti 2003).

A simulation is a relation $\preceq$ on states where, whenever $s \preceq t$, for every transition $s \to s'$ there exists a transition $t \to t'$ using the same action, such that $s' \preceq t'$. In words, t simulates s if anything we can do in s, we can do also in t, leading to a simulating state. (For the reader familiar with the use of bisimulation in merge-and-shrink (Helmert et al. 2014): simulation is "one half of" bisimulation.) A simulation clearly qualifies as a dominance relation. But how to find a simulation on the state space?

We employ a *compositional* approach, obtaining our simulation relation on the state space from simulation relations on *orthogonal projections*, i. e., projections whose variable subsets do not overlap. We enhance simulation with a concept of *label* (action) *dominance*, in addition to states. In our Logistics example above, e. g., for each package this detects the described relation (G is better than being in the truck is better than being at any location other than G). This yields a very strong dominance relation that allows to ignore any state in which a package is unnecessarily unloaded at an irrelevant location. Empirically, we find that indeed our pruning method often substantially reduces the number of expanded nodes.

For space reasons, we omit some proofs. Full proofs, and more examples, will be made available in a TR.

Background

A **planning task** is a 4-tuple $\Pi = (V, A, I, G)$. V is a finite set of **variables** v, each $v \in V$ being associated with a fi-

[1] Precisely, Hall et al. consider numeric state variables r and analyze whether higher r is always good, or is always bad, or neither. In Metric-FF's (Hoffmann 2003) linear normal form, this is equivalent to the formulation above. Hall et al. also handle STRIPS facts, as variables with domain $\{0, 1\}$. But, there, their notions trivialize to "t dominates s if $t \supseteq s$".

nite domain D_v. A **partial state** over V is a function s on a subset $V(s)$ of V, so that $s(v) \in D_v$ for all $v \in V(s)$; s is a **state** if $V(s) = V$. The **initial state** I is a state. The **goal** G is a partial state. A is a finite set of **actions**, each $a \in A$ being a pair (pre_a, eff_a) of partial states, called its **precondition** and **effect**. Each $a \in A$ is also associated with its non-negative **cost** $c(a) \in \mathbb{R}_0^+$.

A **labeled transition system (LTS)** is a tuple $\Theta = (S, L, T, s_0, S_G)$ where S is a finite set of **states**, L is a finite set of **labels** each associated with a **label cost** $c(l) \in \mathbb{R}_0^+$, $T \subseteq S \times L \times S$ is a set of **transitions**, $s_0 \in S$ is the **start state**, and $S_G \subseteq S$ is the set of **goal states**.

The **state space** of a planning task Π is the LTS Θ_Π where: S is the set of all states; s_0 is the initial state I of Π; $s \in S_G$ iff $G \subseteq s$; the labels L are the actions A, and $s \xrightarrow{a} s'$ is a transition in T if s complies with pre_a, and $s'(v) = \mathit{eff}_a(v)$ for $v \in V(\mathit{eff}_a)$ while $s'(v) = s(v)$ for $v \in V \setminus V(\mathit{eff}_a)$. A **plan** for a state s is a path from s to any $s_G \in S_G$. The cost of a cheapest plan for s is denoted $h^*(s)$. A plan for s_0 is a plan for Π, and is **optimal** iff its cost equals $h^*(s_0)$. As defined by Wehrle and Helmert (2014), a plan for s is **strongly optimal** if its number of 0-cost actions is minimal among all optimal plans for s. We denote by $h^{0*}(s)$ the number of 0-cost actions in a strongly optimal plan of s.

Abstractions and abstract state spaces are quite common in planning (e. g. (Helmert, Haslum, and Hoffmann 2007)). We build on this work, but only indirectly. We use merge-and-shrink abstractions as the basis from which our simulation process starts. That process itself will be described in a generic form not relying on these specific constructs. Hence, in what follows, we provide only a summary view that suffices to present our contribution.

Say we have a task $\Pi = (V, A, I, G)$ with state space $\Theta_\Pi = (S, A, T, I, S_G)$, and a variable subset $W \subseteq V$. The **projection** onto W is the function $\pi^W : S \mapsto S^W$ from the states S over V into the states S^W over W, where $\pi^W(s)$ is the restriction of s to W. The **projected state space** Θ_Π^W is the LTS $(S^W, A, T^W, \pi^W(I), S_G^W)$, where $T^W := \{(\pi^W(s), l, \pi^W(s')) \mid (s, l, s') \in T\}$ and $S_G^W := \{\pi^W(s_G) \mid s_G \in S_G\}$. Given two variable subsets W and U, (the projections onto) W and U are **orthogonal** if $W \cap U = \emptyset$. Orthogonality ensures the following **reconstruction** property: $\Theta_\Pi^W \otimes \Theta_\Pi^U = \Theta_\Pi^{W \cup U}$ for orthogonal W and U, where $\otimes$ is the **synchronized product** operation. Namely, given any two labeled transition systems $\Theta^1 = (S^1, L, T^1, s_0^1, S_G^1)$ and $\Theta^2 = (S^2, L, T^2, s_0^2, S_G^2)$ that share the same set L of labels, $\Theta^1 \otimes \Theta^2$ is the labeled transition system with states $S^1 \times S^2$, labels L, transition $(s_1, s_2) \xrightarrow{l} (s_1', s_2')$ iff $s_1 \xrightarrow{l} s_1' \in T^1$ and $s_2 \xrightarrow{l} s_2' \in T^2$, start state (s_0^1, s_0^2), and goal states $\{(s_G^1, s_G^2) \mid s_G^1 \in S_G^1, s_G^2 \in S_G^2\}$.

Merge-and-shrink abstractions (Helmert, Haslum, and Hoffmann 2007; Helmert et al. 2014) construct more general abstraction functions, and the corresponding abstract state spaces, by starting from *atomic abstractions* (projections onto single state variables), and interleaving *merging steps* (replacing two abstractions with their synchronized product) with *shrinking steps* (replacing an abstraction with an abstraction of itself). It has been shown that, if every shrinking step replaces the selected abstraction with a bisimulation of itself, then the final abstraction is a bisimulation of the overall state space Θ_Π. It has also been shown that such shrinking can be combined with *exact label reduction* (Sievers, Wehrle, and Helmert 2014). A **label reduction** is a function τ from the labels L into a set L^τ of reduced labels preserving label cost i. e. $c(l) = c(\tau(l))$. Given an LTS Θ, denote by $\tau(\Theta)$ the LTS identical to Θ except that all labels l have been replaced by $\tau(l)$. Given a set $\{\Theta^1, \ldots, \Theta^k\}$ of LTSs sharing labels L, a label reduction τ is **exact** if $\tau(\Theta^1) \otimes \cdots \otimes \tau(\Theta^k) = \tau(\Theta^1 \otimes \cdots \otimes \Theta^k)$. Reducing labels in this way, and using bisimulation shrinking, merge-and-shrink delivers a bisimulation of $\tau(\Theta_\Pi)$.

Simulation Relations

Given a planning task with states S, a **dominance relation** is a binary relation $\preceq \subseteq S \times S$ where $s \preceq t$ implies $h^*(t) \leq h^*(s)$ and, if $h^*(t) = h^*(s)$ then $h^{0*}(t) \leq h^{0*}(s)$. This is exactly what is needed for admissible pruning during search, as discussed in the next section.

To find dominance relations in practice, we focus on the special case of *simulation relations*. These are well known in model-checking (e. g. (Grumberg and Long 1994; Loiseaux et al. 1995)). Here we use a variant adapted to planning (only) in making explicit the distinction between goal states and non-goal states:

Definition 1 (Simulation) *Let $\Theta = (S, L, T, s_0, S_G)$ be an LTS. A binary relation $\preceq \subseteq S \times S$ is a **simulation** for Θ if, whenever $s \preceq t$ (in words: t **simulates** s), for every transition $s \xrightarrow{l} s'$ there exists a transition $t \xrightarrow{l} t'$ s.t. $s' \preceq t'$. We call $\preceq$ **goal-respecting** for Θ if, whenever $s \preceq t$, $s \in S_G$ implies that $t \in S_G$.*

*We call $\preceq$ the **coarsest** goal-respecting simulation if, for every goal-respecting simulation $\preceq'$, we have $\preceq' \subseteq \preceq$.*

A unique coarsest goal-respecting simulation always exists and can be computed in time polynomial in the size of Θ (Henzinger, Henzinger, and Kopke 1995). Note that the coarsest simulation is always *reflexive*, i. e., $s \preceq s$; the same is true of all simulation relations considered here. Intuitively, every state "dominates itself".

Observe that $s \preceq t$ implies $h^*(t) \leq h^*(s)$, because any plan for s can be also applied to t. Hence a goal-respecting simulation over states is a dominance relation. But, for obtaining that property, goal-respecting simulation is unnecessarily strict. It suffices to preserve, not the label l of the transition $s \to s'$, but only its cost:

Definition 2 (Cost-Simulation) *Let $\Theta = (S, L, T, s_0, S_G)$ be an LTS. A binary relation $\preceq \subseteq S \times S$ is a **cost-simulation** for Θ if, whenever $s \preceq t$, $s \in S_G$ implies that $t \in S_G$, and for every transition $s \xrightarrow{l} s'$ there exists a transition $t \xrightarrow{l'} t'$ s.t. $s' \preceq t'$ and $c(l') \leq c(l)$.*[2]

A cost-simulation still is a dominance relation, and any goal-respecting simulation is a cost-simulation but not vice

[2] Hall et al. (2013) include an equivalent definition, calling it "compatibility" and not relating it to simulation.

versa. However, in our compositional approach where individual dominance relations are computed on orthogonal projections, we need to preserve labels for synchronization across these projections. Hence, to ensure we obtain a dominance relation of the state space, we must use goal-respecting simulation (rather than cost-simulation) on each projection.

For our notion of label dominance, it will be important to consider LTSs with **NOOPs added**. For any LTS Θ, we denote by Θ_{noop} the same LTS but with a new additional label $noop$ where $c(noop) = 0$ and, for every state s, a new transition $s \xrightarrow{noop} s$. Obviously, any dominance relation over Θ_{noop} is a dominance relation over Θ. So for our purposes it suffices to find a dominance relation over the state space $(\Theta_\Pi)_{noop}$ with NOOPs added.

Admissible Dominance Pruning

By **A* with dominance pruning**, we refer to the following modification of A*: *Whenever a node N with state $s(N)$ and path cost $g(N)$ is generated, check whether there exists a node N' in the open or closed lists, with state $s(N')$ and path cost $g(N')$, so that $s(N) \preceq s(N')$ and $g(N') \leq g(N)$. If so, **prune** N, i. e. do not insert it into the open list, nor into the closed list.*

As $s \preceq t$ implies $h^*(t) \leq h^*(s)$, any plan through N costs at least as much as an optimal plan through N', so A* with dominance pruning guarantees optimality. In presence of 0-cost actions, one must be careful to not prune s if this eliminates all possible plans for t. However, s cannot belong to the strongly optimal plan of t because $s \preceq t$ and $h^*(t) = h^*(s)$ implies $h^{0*}(t) \leq h^{0*}(s)$.

Dominance pruning can reduce A*'s search space, but comes with a computational overhead. First, **checking cost**, the runtime required for checking whether there exists a node N' in the open or closed lists, with the mentioned properties. Second, **maintenance cost**, the runtime and memory required for maintaining whichever data structure is used to keep checking cost (which is excessive in a naïve implementation) at bay. Depending on which of these two costs tend to be higher, variants of dominance pruning make sense.

Hall et al.'s (2013) dominance relations are characterized by resources r. They maintain, for each r, the set $S(r)$ of seen states with a positive value for r. Given a new state s, their check iterates over the states in the intersection of $S(r)$ for those r where $s(r)$ is positive. This implementation has high checking cost but low maintenance cost. Hence Hall et al. perform the check not at node-generation time, but at node-expansion time, reducing the number of checks that will be made.

To deal with our much more general dominance relations, we developed a BDD-based (Bryant 1986) implementation. This has low checking cost but high maintenance cost. Hence we perform the check at node-generation time, but only against the closed list, reducing the number of maintenance operations needed.

We maintain a BDD $\mathcal{B}_g$ for the set of states simulated by any $s(N')$ where N' is a previously expanded node with $g(N') = g$. This is done for every g-value of the expanded nodes so far. Every time a node N' is expanded, we determine the set of states $S_{\preceq s(N')}$ simulated by $s(N')$, and add $S_{\preceq s(N')}$ into $\mathcal{B}_{g(N')}$. The checking operation then is very fast: when a node N is generated, test membership of $s(N)$ in $\mathcal{B}_g$ for all $g \leq g(N)$. Each such test takes time linear in the size of the state.

The Compositional Approach

As hinted, our approach is *compositional*, constructing the dominance relation over the state space Θ_Π as the composition of simulation relations over orthogonal projections thereof. Stating this in a generic manner (and simplifying to the atomic case of two orthogonal projections), we have an LTS Θ^{12} which equals the synchronized product $\Theta^1 \otimes \Theta^2$ of two smaller LTSs. We obtain a simulation for Θ^{12} from simulations for Θ^1 and Θ^2:

Definition 3 (Relation Composition) *Let $\Theta^1 = (S^1, L, T^1, s^1_0, S^1_G)$ and $\Theta^2 = (S^2, L, T^2, s^2_0, S^2_G)$ be LTSs sharing the same labels. For binary relations $\preceq_1 \subseteq S_1 \times S_1$ and $\preceq_2 \subseteq S_2 \times S_2$, the **composition** of $\preceq_1$ and $\preceq_2$, denoted $\preceq_1 \otimes \preceq_2$, is the binary relation on $S_1 \times S_2$ where $(s_1, s_2)(\preceq_1 \otimes \preceq_2)(t_1, t_2)$ iff $s_1 \preceq_1 t_1$ and $s_2 \preceq_2 t_2$.*

Proposition 1 *Let $\Theta^{12} = \Theta^1 \otimes \Theta^2$, and let $\preceq_1$ and $\preceq_2$ be goal-respecting simulations for Θ^1 and Θ^2 respectively. Then $\preceq_1 \otimes \preceq_2$ is a goal-respecting simulation for Θ^{12}.*

The proof is direct by definition, and is almost identical to that of a similar result concerning bisimulation, stated by Helmert et al. (2014).

Our basic idea can now be described as follows. Say we have a planning task $\Pi = (V, A, I, G)$ with state space Θ_Π, a partition $V_1, \ldots, V_k$ of the task's variables, and a goal-respecting bisimulation abstraction α_i of each $\tau(\Theta_\Pi^{V_i})$ where τ is an exact label reduction. This is precisely the input we will get from merge-and-shrink abstraction. We will henceforth refer to this input as our *initial abstractions*. Say we construct a goal-respecting simulation $\preceq_i$ for each abstract state space Θ^{α_i}. Because bisimulation is a special case of simulation, $\preceq_i$ is a goal-respecting simulation for $\tau(\Theta_\Pi^{V_i})$. Applying Definition 3 and Proposition 1 iteratively, $\bigotimes_i \preceq_i$ is a goal-respecting simulation for $\bigotimes_i \tau(\Theta_\Pi^{V_i})$. Because τ is exact, $\bigotimes_i \tau(\Theta_\Pi^{V_i}) = \tau(\bigotimes_i \Theta_\Pi^{V_i})$, and by the reconstruction property $\tau(\bigotimes_i \Theta_\Pi^{V_i}) = \tau(\Theta_\Pi)$. Because the label reduction is cost-preserving, $\bigotimes_i \preceq_i$ is a cost-simulation for Θ_Π, and hence a dominance relation as desired.

One can use this result and method as-is, obtaining a new dominance pruning method as a corollary of suitably assembling existing results and methods. However, empirically, this method's ability to find interesting dominance relations is quite limited. We now extend the simulation concept to overcome that problem.

Label-Dominance Simulation

Sievers et al. (2014) introduce label "subsumption", where l' subsumes l if it labels all transitions labeled by l. To intertwine dominance between labels with dominance between states, we extend that concept as follows:

Definition 4 (Label Dominance) *Let Θ be an LTS with states S, let $\preceq \subseteq S \times S$ be any binary relation on S, and let l, l' be labels. We say that l' **dominates** l **in** Θ **given** $\preceq$ if $c(l') \leq c(l)$, and for every transition $s \xrightarrow{l} s'$ there exists a transition $s \xrightarrow{l'} t'$ s.t. $s' \preceq t'$.*

The relation $\preceq$ here is arbitrary, but will be a simulation in practice. Hence, intuitively, a label dominates another one if it "applies to the same states and always leads to an at least as good state". To give a simple example, consider the LTS corresponding to a single vehicle's position, and say we have a 0-cost "beam" action which takes us from any position to the vehicle's goal. Provided that every position is, per $\preceq$, simulated by the goal position, "beam" dominates all other labels.

In IPC benchmarks, typically Definition 4 is important not for regular actions, but for NOOPs.

Example 1 *say we have a truck variable v_T, two locations A and B, and a package variable v_P whose goal is to be at B. Our variable partition is the trivial one, $V_1 = \{v_T\}$ and $V_2 = \{v_P\}$. Bisimulation using exact label reduction will return LTSs as shown in Figure 1. The "load" and "unload" actions get reduced in a way allowing to synchronize with the correct truck position; the distinction between the truck "drive" actions is irrelevant so these are reduced to the same label. Clearly, no label dominates any other in either of these two LTSs. However, consider Θ^1 and Θ^2 with NOOPs added. The new label $noop$ dominates the load/unload actions in Θ^1, and dominates the drive actions in Θ^2, provided $\preceq_1$ and $\preceq_2$ are reflexive as will be the case in practice.*

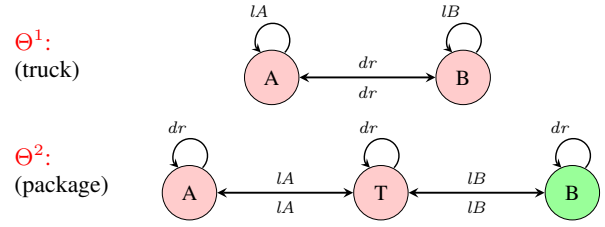


Figure 1: Label-reduced bisimulations, i. e. the input to our simulation process, in the Logistics example.

This behavior allows us, e. g., to conclude that, in Θ^2, B dominates T: While T has an outgoing transition to B, labeled lB, B itself has no such outgoing label. However, B has the outgoing label $noop$ leading to B. The transition $B \xrightarrow{noop} B$ simulates $T \xrightarrow{lB} B$, except that it uses label $l' = noop$ instead of label $l = lB$. This is admissible (only) if l' dominates l in all other LTSs involved. In our case here, the only other LTS is Θ^1, and indeed the label $l' = noop$ dominates $l = lB$ in that LTS.

We exploit this kind of information as follows:

Definition 5 (Label-Dominance Simulation) *Let $\mathcal{T} = \{\Theta^1, \ldots, \Theta^k\}$ be a set of LTSs sharing the same labels. Denote the states of Θ^i by S_i. A set $\mathcal{R} = \{\preceq_1, \ldots, \preceq_k\}$ of binary relations $\preceq_i \subseteq S_i \times S_i$ is a **label-dominance simulation** for $\mathcal{T}$ if, whenever $s \preceq_i t$, $s \in S_i^G$ implies that $t \in S_i^G$, and for every transition $s \xrightarrow{l} s'$ in Θ^i, there exists a transition $t \xrightarrow{l'} t'$ in Θ^i such that $c(l') \leq c(l)$, $s' \preceq_i t'$, and, for all $j \neq i$, l' dominates l in Θ^j given $\preceq_j$.*

*We call $\mathcal{R}$ the **coarsest** label-dominance simulation if, for every label-dominance simulation $\mathcal{R}' = \{\preceq'_1, \ldots, \preceq'_k\}$ for $\mathcal{T}$, we have $\preceq'_i \subseteq \preceq_i$ for all i.*

A unique coarsest label-dominance simulation always exists, and can be computed in time polynomial in the size of $\mathcal{T}$. We will prove this in the next section as a corollary of specifying our algorithm for doing this computation.

In the example, $T \preceq_2 B$ holds because $s \xrightarrow{l} s'$ in Θ^2 is $T \xrightarrow{lB} B$, and the simulating $t \xrightarrow{l'} t'$ in Θ^2 is $B \xrightarrow{noop} B$, which works because $c(noop) = 0 \leq 1 = c(lB)$, $B \preceq_2 B$, and $noop$ dominates lB in Θ^1. In the same fashion, provided that $A \preceq_2 B$, the transition $T \xrightarrow{lA} A$ is simulated by $B \xrightarrow{noop} B$. Note that neither of these two inferences could be made with the standard concept of simulation (even after exact label reduction), because that concept insists on using the same labels, not dominating ones.

We now prove soundness of label-dominance simulation, i. e., that label-dominance simulations $\mathcal{R}$ yield cost-simulations of the original state space. Similarly to before, we iteratively compose $\mathcal{R}$'s element relations, as captured by the following lemma:

Lemma 1 *Let $\mathcal{T} = \{\Theta^1, \ldots, \Theta^k\}$ be a set of LTSs sharing the same labels, and let $\mathcal{R} = \{\preceq_1, \ldots, \preceq_k\}$ be a label-dominance simulation for $\mathcal{T}$. Then $\{\preceq_1 \otimes \preceq_2, \preceq_3, \ldots, \preceq_k\}$ is a label dominance simulation for $\{\Theta^1 \otimes \Theta^2, \Theta^3, \ldots, \Theta^k\}$.*

Proof Sketch: The claim regarding $\preceq_3, \ldots, \preceq_k$ is simple. For $\preceq_1 \otimes \preceq_2$, consider states $(s_1, s_2) \preceq_{12} (t_1, t_2)$ and a transition $(s_1, s_2) \xrightarrow{l} (s'_1, s'_2)$. We identify a dominating transition $(t_1, t_2) \xrightarrow{l'} (t'_1, t'_2)$ as follows: 1. As $s_1 \preceq_1 t_1$, obtain a transition $t_1 \xrightarrow{l^{tmp}} t_1^{tmp}$ dominating $s_1 \xrightarrow{l} s'_1$ in Θ^1. 2. As l^{tmp} dominates l in Θ^2, obtain a transition $s_2 \xrightarrow{l^{tmp}} s_2^{tmp}$ dominating $s_2 \xrightarrow{l} s'_2$ in Θ^2. 3. As $s_2 \preceq_2 t_2$, obtain a transition $t_2 \xrightarrow{l'} t'_2$ dominating $s_2 \xrightarrow{l^{tmp}} s_2^{tmp}$ in Θ^2. 4. As l' dominates l^{tmp} in Θ^1, obtain a transition $t_1 \xrightarrow{l'} t'_1$ dominating $t_1 \xrightarrow{l^{tmp}} t_1^{tmp}$ in Θ^1. □

Theorem 1 *Let $\mathcal{T} = \{\Theta^1, \ldots, \Theta^k\}$ be a set of LTSs sharing the same labels, and let $\mathcal{R} = \{\preceq_1, \ldots, \preceq_k\}$ be a label-dominance simulation for $\mathcal{T}$. Then $\bigotimes_i \preceq_i$ is a cost-simulation for $\bigotimes_i \Theta^i$.*

Proof: Applying Lemma 1, we get that $\bigotimes_i \preceq_i$ is a label-dominance simulation for $\{\bigotimes_i \Theta^i\}$. Now, for such a singleton set of LTSs, the requirements on the transition $t \xrightarrow{l'} t'$ replacing $s \xrightarrow{l} s'$ are that $c(l') \leq c(l)$, and $s' \preceq_i t'$. Hence label-dominance simulation simplifies to cost-simulation, and the claim follows. □

To clarify the overall process, assume now again the initial abstractions provided by merge-and-shrink abstraction, i. e., a partition $V_1, \ldots, V_k$ of the variables, and a goal-respecting bisimulation abstraction α_i, with abstract state space Θ^{α_i}, of each $\tau(\Theta_\Pi^{V_i})$ where τ is an exact label reduction. We compute the coarsest label-dominance simulation $\mathcal{R} = \{\preceq_1, \ldots, \preceq_k\}$ for $\mathcal{T} := \{\Theta^{\alpha_1}_{noop}, \ldots, \Theta^{\alpha_k}_{noop}\}$. As adding NOOPs does not affect bisimulation and is interchangeable with the synchronized product, with Theorem 1 we have that $\bigotimes_i \preceq_i$ is a cost-simulation for $[\bigotimes_i \tau(\Theta_\Pi^{V_i})]_{noop}$, and hence a cost-simulation for Θ_Π as desired.

Computing $\mathcal{R}$

We now show how to operationalize Definition 5: Given $\mathcal{T} = \{\Theta^1, \ldots, \Theta^k\}$, how to compute the coarsest label-dominance simulation $\mathcal{R}$ for $\mathcal{T}$?

It is well known that the coarsest simulation can be computed in time polynomial in the size of the input LTS (Henzinger, Henzinger, and Kopke 1995). The algorithm starts with the most generous relation $\preceq$ possible, then iteratively removes pairs $s \preceq t$ that do not satisfy the simulation condition. When no more changes occur, the unique coarsest simulation has been found. This method extends straightforwardly to label-dominance simulation.

Proposition 2 *Let $\mathcal{T} = \{\Theta^1, \ldots, \Theta^k\}$ be a set of LTSs sharing the same labels. Then a unique coarsest label-dominance simulation for $\mathcal{T}$ exists.*

Proof: The identity relation is a label-dominance simulation. If $\mathcal{R} = \{\preceq_1, \ldots, \preceq_k\}$ and $\mathcal{R}' = \{\preceq'_1, \ldots, \preceq'_k\}$ are label-dominance simulations, then $\{\preceq_1 \cup \preceq'_1, \ldots, \preceq_k \cup \preceq'_k\}$, also is a label-dominance simulation. □

Denote the states of Θ^i by S_i. Define the Boolean function $\mathbf{Ok}(i, s, t)$, where $s \preceq_i t$, to return **true** iff the condition for label-dominance simulation holds, i. e., iff $s \in S_i^G$ implies that $t \in S_i^G$, and for every transition $s \xrightarrow{l} s'$ in Θ^i there exists a transition $t \xrightarrow{l'} t'$ in Θ^i such that $c(l') \leq c(l)$, $s' \preceq_i t'$, and, for all $j \neq i$, l' dominates l in Θ^j given $\preceq_j$. Our algorithm proceeds as follows:

For all i, set $\preceq_i := \{(s,t) \mid s,t \in S_i, s \notin S_G^i \text{ or } t \in S_G^i\}$
while ex. (i,s,t) s.t. not $\mathbf{Ok}(i,s,t)$ **do**
  Select one such triple (i,s,t)
  Set $\preceq_i := \preceq_i \setminus \{(s,t)\}$
endwhile
return $\mathcal{R} := \{\preceq_1, \ldots, \preceq_k\}$

Proposition 3 *Let $\mathcal{T} = \{\Theta^1, \ldots, \Theta^k\}$ be a set of LTSs sharing the same labels. Our algorithm terminates in time polynomial in the size of $\mathcal{T}$, and returns the coarsest label-dominance simulation for $\mathcal{T}$.*

Proof: Each iteration reduces one $\preceq_i$ by one element. This gives a polynomial bound on the number of iterations, and every iteration takes polynomial time.

The returned $\mathcal{R}$ is a label-dominance simulation as that is the termination condition. $\mathcal{R}$ is coarsest as every label-dominance simulation must refine the initial relations $\{(s,t) \mid s,t \in S_i, s \notin S_G^i \text{ or } t \in S_G^i\}$, and every time we remove a pair (s,t) we know that $s \not\preceq_i t$ in any label-dominance simulation. □

Example 2 *Consider again our Logistics example. The initial relation $\preceq_1$ for the truck is complete, i. e. $\preceq_1 = \{(A,A), (A,B), (B,A), (B,B)\}$ because the truck has no own goal. In the initial relation $\preceq_2$ for the package, we have all pairs (s,t) except ones where s is in the goal but t is not: $\preceq_2 = \{(A,A), (A,T), (T,A), (T,T), (A,B), (T,B), (B,B)\}$.*

Figure 1 shows the LTSs the fixed point algorithm will work on. Considering the package relation $\preceq_2$, note that $(2,T,A)$ is not Ok: A cannot match the transition $T \xrightarrow{lB} B$ because the only value dominating B is B itself, and A does not have any outgoing transition to B. Now consider the truck variable. Note first that $(1,A,B)$ is not Ok: A has the outgoing transition $A \xrightarrow{lA} A$. B could only match this via $B \xrightarrow{dr} A$, $B \xrightarrow{lB} B$, or $B \xrightarrow{noop} B$ but neither dr, lB, nor $noop$ dominate lA in the package LTS Θ^2 since (T,A) is not in $\preceq_2$ anymore. The same holds similarly for $(1,B,A)$ because of the outgoing transition $B \xrightarrow{lB} B$ of B.

Hence $\preceq_1$ is reduced to the identity relation and $\preceq_2$ is reduced to $\{(A,A), (A,T), (T,T), (A,B), (T,B), (B,B)\}$. Note that this relation corresponds to the statement "T dominates A, and B dominates T", which is exactly what we wanted to obtain. Indeed, the algorithm stops here, i. e., all elements of $\preceq_2$ are now Ok. This is trivial for the identity pairs (A,A), (T,T), (B,B). Regarding (A,T), transition $A \xrightarrow{lA} T$ is simulated by $T \xrightarrow{noop} T$ because $noop$ dominates lA in Θ^1. Regarding (T,B), we already discussed above that both $T \xrightarrow{lB} B$ and $T \xrightarrow{lA} A$ are simulated by $B \xrightarrow{noop} B$. The same is true, regarding (A,B), for $A \xrightarrow{lA} T$.

Note that standard simulation relation does not derive any dominance relation other than the identity relation in our example. The desired relation is only obtained thanks to using label-dominance and the $noop$ operation.

Experiments

Our techniques are implemented in Fast Downward (FD) (Helmert 2006). We ran all optimal-track STRIPS planning instances from the international planning competitions (IPC'98 – IPC'14). All experiments were conducted on a cluster of Intel E5-2660 machines running at 2.20 GHz, with time (memory) cut-offs of 30 minutes (4 GB). We run A* with FD's blind heuristic, and with LM-cut (Helmert and Domshlak 2009). We perform an ablation study of label-dominance simulation, vs. standard simulation (neither NOOPs nor label-dominance), vs. bisimulation as computed by merge-and-shrink (not doing any work on top of merge-and-shrink, just using its output for the pruning). To represent the state of the art in alternative pruning methods, we include the best-performing partial-order reduction based on strong stubborn sets, which dominates

		Blind												LM-cut											
		Coverage						Evaluations				Gen/sec.		Coverage						Evaluations				Gen/sec.	
Domain	#	A	L_0	L	S	B	P	L	S	B	P	L	P	A	L_0	L	S	B	P	L	S	B	P	L	P
Airport	50	**22**	−7	−7	−7	**0**	−1	1.2	1.2	1	4.4	341	11.3	28	−3	−1	−1	−1	**+1**	1	1	1	4.7	1.1	2
Driverlog	20	7	**+2**	**+2**	**0**	**0**	**0**	15.8	2	2	1	4.8	2.8	13	**0**	**0**	**0**	**0**	**0**	1.9	1.2	1.2	1	1.1	1.2
Elevators08	30	**14**	−1	**0**	**0**	**0**	**0**	1	1	1	1.1	0.9	4.1	22	**0**	**0**	**0**	**0**	**0**	1	1	1	1.3	1	1.2
Elevators11	20	**12**	−1	**0**	**0**	**0**	**0**	1	1	1	1.1	1	4.2	18	**0**	**0**	**0**	**0**	**0**	1	1	1	1.2	1	1.2
Floortile11	20	2	**+4**	**+4**	**+4**	**0**	**0**	177	177	1.8	1.3	5.7	3.7	7	**+1**	**+1**	**+1**	**0**	**0**	6.4	6.4	1	1	1.1	1.1
Floortile14	20	0	**+5**	**+5**	**+5**	**0**	**0**	–	–	–	–	–	–	6	**+2**	**+2**	**+2**	**0**	**0**	6.3	6.3	1	1	1.3	1.1
FreeCell	80	**20**	−7	**0**	**0**	**0**	−6	1	1	1	1	1	31.2	**15**	−1	**0**	**0**	**0**	**0**	1	1	1	1	0.9	1.4
Gripper	20	8	**+6**	**+6**	**+6**	**+6**	**0**	53968	53968	28353	1	292	3.1	7	**+7**	**+7**	**+7**	**+7**	**0**	14662	14662	10049	1	31.9	1.3
Hiking14	20	**11**	**0**	**0**	**0**	**0**	−3	2.4	1.9	1.8	1	3.1	30.5	9	**0**	**0**	**0**	**0**	**0**	1.7	1.5	1.5	1	1.9	1.8
Logistics00	28	10	**+7**	+6	**0**	**0**	**0**	32.7	3.1	1.2	1	9.3	3	20	**0**	**0**	**0**	**0**	**0**	1.9	1.1	1.1	2.9	0.8	1.4
Logistics98	35	2	**+1**	**+1**	**0**	**0**	**0**	6.7	1.2	1.2	1.5	4	4.4	6	**0**	**0**	**0**	**0**	**0**	1.3	1	1	4.3	0.9	1.3
Miconic	150	55	**+6**	**+6**	−1	**0**	−5	58.3	8.7	3.4	1	15.6	5.5	141	**0**	**0**	**0**	**0**	**0**	2.1	1.5	1.1	1	0.6	1.1
Mprime	35	**20**	−1	−1	**0**	**0**	−1	1.1	1	1	1	18	18.5	22	**0**	**0**	**0**	**0**	**0**	1.1	1	1	1	1	1.1
Mystery	30	**15**	−3	−3	−3	**0**	**0**	1.9	1.9	1	1.1	29.2	14.2	17	**0**	**0**	**0**	**0**	**0**	3.5	3.5	1	1.4	3.4	1.8
NoMystery	20	8	**+10**	**+10**	+1	+1	**0**	2497	128	29.1	1.1	46.4	23	14	**+6**	**+6**	+3	**0**	**0**	6.5	3.1	1	1	0.6	1.2
OpenStack08	30	22	**+2**	**+2**	**+2**	+1	**0**	2.1	2	1.8	2	8.1	9.3	21	**0**	**0**	**0**	**0**	**0**	2.5	2.4	2.1	1.8	2.3	2
OpenStack11	20	17	**+2**	**+2**	**+2**	+1	**0**	2.1	2	1.8	2	7.8	9.3	16	**0**	**0**	**0**	**0**	**0**	2.5	2.4	2.1	1.8	2.3	2.1
OpenStack14	20	3	**0**	**0**	**0**	**0**	**+1**	2.8	2.8	2.5	1.8	7	7.8	3	**0**	**0**	**0**	**0**	**0**	2.9	2.8	2.5	1.8	2.4	2.1
ParcPrint08	30	10	+6	+5	+3	+1	**+20**	862	10	1.5	18349	13.6	532	18	**0**	**0**	**0**	**0**	**+12**	5	1.2	1.1	1028	2.2	20.3
ParcPrint11	20	6	+6	+5	+3	+1	**+14**	869	10	1.5	21826	11.2	371	13	**0**	**0**	**0**	**0**	**+7**	5	1.2	1.1	1246	2.4	17.8
PegSol08	30	27	**0**	**0**	**0**	**0**	**0**	1	1	1	1	1	3.8	**28**	−1	**0**	**0**	**0**	−1	1	1	1	1	1	1.4
PegSol11	20	17	**0**	**0**	**0**	**0**	**0**	1	1	1	1	1	3.8	**18**	−1	**0**	**0**	**0**	−1	1	1	1	1	1	1.4
PipesNoTank	50	**17**	−8	−1	−1	**0**	−3	1	1	1	1	1.1	10.3	**17**	−3	**0**	**0**	**0**	**0**	1	1	1	1	1	1.1
PipesTank	50	**12**	−1	**0**	**0**	**0**	−3	1.1	1.1	1.1	1	16.8	25.2	**12**	**0**	**0**	**0**	**0**	−1	1.8	1.8	1.8	1	1.1	1.2
Rovers	40	6	**+2**	**+2**	+1	**0**	+1	33.4	9.6	1.7	2	20.6	3	7	**+2**	**+2**	+1	+1	**+2**	6.1	3.8	1.2	4.4	1.8	1.8
Satellite	36	6	**0**	**0**	**0**	**0**	**0**	72.9	35.3	9.9	10.7	8.4	3.8	7	+3	+3	+3	+3	**+4**	4.8	1.8	1.7	21.5	0.9	2.3
Scanalyzer08	30	**12**	**0**	**0**	**0**	**0**	−4	1	1	1	1	1	8.7	**15**	−1	−1	−1	**0**	**0**	1	1	1	1	1	1.2
Scanalyzer11	20	**9**	**0**	**0**	**0**	**0**	−4	1	1	1	1	1	8.7	**12**	−1	−1	−1	**0**	**0**	1	1	1	1	1	1.2
Sokoban08	30	**22**	−9	**0**	**0**	**0**	−1	1	1	1	1	1.7	8.2	**29**	−7	−1	**0**	**0**	**0**	1	1	1	1	1.1	1.2
Sokoban11	20	**19**	−9	**0**	**0**	**0**	−1	1	1	1	1	1.6	8.1	**20**	−2	**0**	**0**	**0**	**0**	1	1	1	1	1.1	1.2
Tetris	17	9	−6	**−1**	**−1**	**−1**	−4	1	1	1	1	5.2	52.2	6	−3	−2	−2	−2	**−1**	1	1	1	1	1	1.3
Tidybot11	20	9	−8	−7	−7	**−1**	−2	5.5	5.5	1	1.8	59.4	8.5	**14**	−2	−2	−2	**0**	**0**	6.8	6.8	1	1.5	2.6	1.3
Tidybot14	20	2	−2	−2	−2	**−1**	−2	–	–	–	–	–	–	9	−7	−7	−7	**−1**	**−1**	3.9	3.9	1	1.7	3.1	1.4
TPP	30	6	**0**	**0**	**0**	**0**	**0**	6.5	3.4	1	1	22.7	3.3	6	**+1**	**+1**	**+1**	**+1**	**0**	1.2	1.1	1	1	1.3	1.1
Transport14	20	**7**	**0**	**0**	**0**	**0**	−1	1	1	1	1	1	9.1	6	**0**	**0**	**0**	**0**	**0**	1.4	1.4	1.4	1	1.4	1.2
Trucks	30	6	**+2**	**+2**	**0**	**0**	**0**	24.8	21.9	2.8	1	13.8	6	10	**0**	**0**	**0**	**0**	**0**	2.7	2.3	1	1	1.2	1.2
VisitAll11	20	9	**0**	**0**	**0**	**0**	**0**	30	25.5	1	1	104	3.5	10	**+1**	**+1**	**+1**	**0**	**0**	7	6.8	1	1	1.5	1.1
VisitAll14	20	3	**+1**	**+1**	**+1**	**0**	**0**	27.8	23.4	1	1	92.8	3.5	5	**0**	**0**	**0**	**0**	**0**	5.2	5.1	1	1	1.6	1.1
Woodwork08	30	8	**+10**	**+10**	+5	+4	+7	981	112	87.8	488	7.6	7.5	17	+7	+7	+5	+5	**+10**	91.4	23.7	16.9	762	1.8	3.1
Woodwork11	20	3	**+9**	**+9**	+5	+4	+6	1059	116	92.2	514	6.7	6.3	12	+5	+5	+4	+4	**+7**	91.6	23.8	17	772	1.8	2.9
Zenotravel	20	8	**+1**	**+1**	**0**	**0**	**0**	41.6	1.5	1.1	1	4.3	6.2	13	**0**	**0**	**0**	**0**	**0**	3.6	1.6	1	1	1	1.2
$\sum$	1271	605	+19	**+57**	+16	+16	+8	1.8	1.7	1.5	1.4	4.2	7.4	833	+3	+20	+14	+17	**+38**	0	0	0	0	1.7	1.5

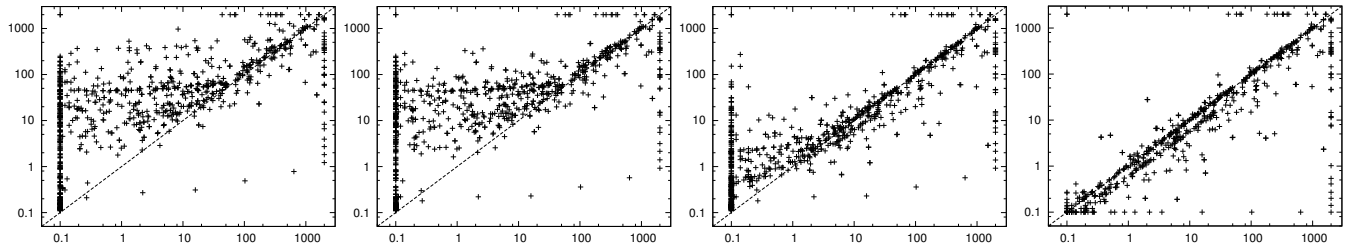
Table 1: Experiments. "A": A^* without pruning. "L_0", "L": label-dominance simulation; "S": simulation; "B": bisimulation; "L_0" is without safety belt (see text), all others with safety belt. "P": partial-order reduction. Domains where no changes in coverage occur anywhere are omitted. "Evaluations" is the factor by which the per-domain summed-up number of evaluated states, relative to "A", decreases. "Gen/sec." is the factor by which the per-node runtime (summed-up number of generated nodes divided by summed-up search time), relative to "A", increases.

other partial-order pruning approaches such as expansion-core (Wehrle et al. 2013).

Our initial abstractions are obtained using merge-and-shrink with exact label reduction, bisimulation shrinking, and the non-linear merge DFP strategy (Dräger, Finkbeiner, and Podelski 2006; 2009; Sievers, Wehrle, and Helmert 2014). We impose two bounds on this process, namely a time limit of 300 seconds, as well as a limit M on the number of abstract transitions. When either of these limits is reached, the last completed abstractions form the starting point for our simulation process, i. e., are taken to be the initial abstractions. With this arrangement of parameters, the trade-off between merge-and-shrink overhead incurred vs. benefits gained is relatively easy to control. The bound on transitions works better than the more usual bound on abstract states, because the same number of abstract states may lead to widely differing numbers of transitions and thus actual effort. A reasonably good "magic" setting for M, in our current context, is $100k$. For $M = 0$, i. e. computing the component simulations on individual state variables only, performance is substantially worse. For $M = 200k$, the overhead becomes prohibitive. In between, overall coverage undergoes relatively small changes only (in the order of 5 instances).

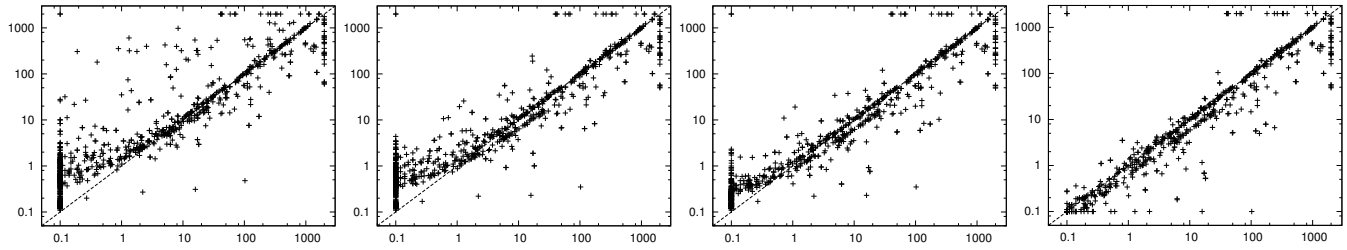
Consider Table 1. With our pruning method, nodes are first generated and then checked for pruning, so the evaluated states are exactly the non-pruned generated ones. Hence the number of evaluated states assesses our pruning power, and the ratio between generated nodes and search time assesses the average time-per-node. The "safety belt" disables pruning if, after 1000 expansions, no node has been pruned. This is a simple yet effective method to avoid runtime overhead in cases where no or not much pruning will be obtained.

Compared to partial-order reduction, simulation-based pruning tends to be "stronger on its own, but less complementary to LM-cut". Consider first the blind heuristic, which assesses the pruning power of each technique "on its own". Simulation-based pruning typically yields much stronger evaluation reductions, the only clear exception being ParcPrinter where partial-order reduction excels. This results in much better coverage in many domains and overall. With LM-cut, on the other hand, while simulation-based pruning still applies more broadly – there are 14 test suites



(a) total time (b) − M&S time (c) − simulation time (d) − BDD setup time

Figure 2: Runtime of A* with LM-cut, without pruning on the x-axis, with simulation-based pruning on the y-axis. We distinguish the successive pre-processing overheads in our current implementation by deducting them iteratively. Version with max number transitions of 100 000.



(a) total time (b) − M&S time (c) − simulation time (d) − BDD setup time

Figure 3: Runtime of A* with LM-cut, without pruning on the x-axis, with simulation-based pruning on the y-axis. We distinguish the successive pre-processing overheads in our current implementation by deducting them iteratively. Version with max number transitions of 10 000.

where it reduces evaluations but partial-order reduction does not – the extent of the reduction is dramatically diminished. Partial-order reduction suffers from this as well, but retains much of its power in ParcPrinter and Woodworking, and consistently causes very little runtime overhead relative to this slow heuristic function. Thus partial-order reduction has better overall coverage. It does not dominate simulation-based pruning though, which yields better coverage in Floortile, Gripper, NoMystery, TPP, and VisitAll.

Label-dominance simulation clearly pays off against standard simulation as well as bisimulation. The latter already is very helpful in some domains, like Gripper and Woodworking. Simulation does add over this, but suffers in some domains, like Tidybot, from the additional runtime overhead. Label-dominance simulation has such issues as well, but makes up for them by more pronounced gains on other domains.

The per-node runtime overhead in simulation-based pruning is almost consistently outweighed by the search space size reduction (compare the respective "Gen/sec." vs. "Evaluations" columns in Table 1). The most substantial runtime overhead stems from computing the simulation relations. Our current implementation of that process is largely naïve. We experimented with ideas from model checking for doing this more effectively, but with limited success due to the different context (especially, label-dominance). It remains an important open topic to improve this part of our machinery.

Figure 2 shows a comparison of the time to solve a problem with and without dominance pruning when considering different parts of the preprocessing or not. We subdivide preprocessing time into three separated components: the time to generate the M&S abstractions, the time to compute the simulation relation and the time to initialize the BDDs. The plots show the comparison of total time and then substract the three parts of the preprocessing, one at a time.

The per-node overhead is almost consistently outweighted by the search space size reduction, as shown by the search time comparison in Figure 2d. Thus, runtime is improved in large instances, where the preprocessing runtime is dominated by the search runtime, but not on small ones due to spending up to 300 seconds in the preprocessing phase (see Figure 2).

However, the time spent in preprocessing can be controlled by lowering the parameter max number of transitions, M. Smaller M avoids much of the preprocessing overhead, at a small price in overall coverage. Figure 3 shows the time comparison when the abstraction size is kept below 10 000 transitions. In this case, the overhead of computing the label-dominance simulation and BDD initialization is heavily reduced. Most of the overhead is due to the creation of the M&S abstractions. Reducing M does not increase coverage since the main cause for failing at problems solved by the baseline is that M&S runs out of time or memory during label reduction. Total coverage is 846, seven problems less than the version with abstractions of 100 000 nodes. The coverage decreases in Woodworking (-3), Scanalyzer (-2), and Gripper(-7), but the reduced overhead makes coverage increase in OpenStack($+4$) and Sokoban($+1$).

Conclusion

The idea of pruning states based on some form of "dominance" is old, but has previously been incarnated in planning with simple special cases ("more facts true", "more resources available") only. Simulation relations are *the* natural framework to move beyond this. Our work constitutes a first step towards leveraging the power of simulation relations in, as well as extending them for, admissible pruning in planning. The method is orthogonal to existing pruning methods, and empirically exhibits complementary strengths relative to partial-order reduction, so there is potential for synergy. A major challenge in our view is how to intelligently control initial-abstraction size, investing a lot of overhead where simulation pruning is promising and, ideally, avoiding any overhead altogether where it is not.

Proofs

Proposition 1 *Let $\Theta^{12} = \Theta^1 \otimes \Theta^2$, and let $\preceq_1$ and $\preceq_2$ be goal-respecting simulations over Θ^1 and Θ^2 respectively. Then $\preceq_1 \otimes \preceq_2$ is a goal-respecting simulation for Θ^{12}.*

Proof: Denote, for simplicity, $\preceq_1 \otimes \preceq_2$ by $\preceq_{12}$. First, we show that $\preceq_{12}$ is a simulation of Θ^{12}. For every $(s_1, s_2) \preceq_{12} (t_1, t_2)$ and transition $(s_1, s_2) \xrightarrow{l} (s_1', s_2')$ in Θ^{12}, we have to show that there exists a transition $(t_1, t_2) \xrightarrow{l} (t_1', t_2')$ in Θ^{12} such that $(s_1', s_2') \preceq_{12} (t_1', t_2')$.

As $(s_1, s_2) \xrightarrow{l} (s_1', s_2')$, by the definition of the synchronized product, $s_1 \xrightarrow{l} s_1'$ is a transition in Θ^1 and $s_2 \xrightarrow{l} s_2'$ is a transition in Θ^2. From $(s_1, s_2) \preceq_{12} (t_1, t_2)$, by construction of $\preceq_{12}$ we know that $s_1 \preceq_1 t_1$ and $s_2 \preceq_2 t_2$. Therefore, because $\preceq_1$ and $\preceq_2$ are simulations over Θ^1 and Θ^2 respectively, there exist transitions $t_1 \xrightarrow{l} t_1'$ in Θ^1 and $t_2 \xrightarrow{l} t_2'$ in Θ^2 such that $s_1' \preceq_1 t_1'$ and $s_2' \preceq_2 t_2'$. We have $(s_1', s_2') \preceq_{12} (t_1', t_2')$ by construction of $\preceq_{12}$. Moreover, by the definition of the synchronized product, $(t_1, t_2) \xrightarrow{l} (t_1', t_2')$ in Θ^{12} as desired.

We now prove that $\preceq_{12}$ is goal-respecting. Suppose for contradiction that $(s_1, s_2) \preceq_{12} (t_1, t_2)$ and (s_1, s_2) is a goal state, but (t_1, t_2) is not. Then both s_1 and s_2 must be goal states, and at least one of t_1 or t_2 must be a non-goal state. By construction of $\preceq_{12}$, $s_1 \preceq_1 t_1$ and $s_2 \preceq_2 t_2$. Therefore, either $\preceq_1$ or $\preceq_2$ is not goal-respecting, in contradiction. □

Lemma 1 *Let $\mathcal{T} = \{\Theta^1, \ldots, \Theta^k\}$ be a set of LTSs sharing the same labels, and let $\mathcal{R} = \{\preceq_1, \ldots, \preceq_k\}$ be a label-dominance simulation for $\mathcal{T}$. Then $\{\preceq_1 \otimes \preceq_2, \preceq_3, \ldots, \preceq_k\}$ is a label dominance simulation for $\{\Theta^1 \otimes \Theta^2, \Theta^3, \ldots, \Theta^k\}$.*

Proof: Denote, for simplicity, $\preceq_1 \otimes \preceq_2$ by $\preceq_{12}$ and $\Theta^1 \otimes \Theta^2$ by Θ^{12}. We first show that $\preceq_{12}$ satisfies its part of the claim: For every $(s_1, s_2) \preceq_{12} (t_1, t_2)$ we show that (i) if (s_1, s_2) is a goal state in Θ^{12} then (t_1, t_2) also is a goal state in Θ^{12}, and (ii) for every transition $(s_1, s_2) \xrightarrow{l} (s_1', s_2')$, there exists a transition $(t_1, t_2) \xrightarrow{l'} (t_1', t_2')$ where $c(l') \leq c(l)$, $(s_1', s_2') \preceq_{12} (t_1', t_2')$ and l' dominates l in Θ^j given $\preceq_j$ for all $j \geq 3$.

Part (i) is easy to see: As $\mathcal{R}$ is a label dominance simulation, and $s_1 \preceq_1 t_1$ as well as $s_2 \preceq_1 t_2$ by construction of $\preceq_{12}$, we know that $s_i \in S_i^G$ implies $t_i \in S_i^G$ (for $i = 1, 2$). The claim then follows directly from the definition of the synchronized product.

Regarding part (ii), observe first that, because $s_1 \xrightarrow{l} s_1'$ is a transition in Θ^1, and $\mathcal{R}$ is a label dominance simulation, there is a transition $t_1 \xrightarrow{l^{tmp}} t_1^{tmp}$ in Θ^1 such that $c(l^{tmp}) \leq c(l)$, $s_1' \preceq_1 t_1^{tmp}$, and l^{tmp} dominates l in Θ^j given $\preceq_j$ for all $j \geq 2$. As l^{tmp} dominates l in Θ^2, and $s_2 \xrightarrow{l} s_2'$ is a transition in Θ^2, there is a transition $s_2 \xrightarrow{l^{tmp}} s_2^{tmp}$ in Θ^2, where $s_2' \preceq_2 s_2^{tmp}$. Now, as $s_2 \preceq_2 t_2$ and $\mathcal{R}$ is a label dominance simulation, there is a transition $t_2 \xrightarrow{l'} t_2'$ in Θ^2 such that $c(l') \leq c(l^{tmp})$, $s_2^{tmp} \preceq_2 t_2'$, and l' dominates l^{tmp} in Θ^j given $\preceq_j$ for all $j \neq 2$. Because l' dominates l^{tmp} in Θ^1 given $\preceq_1$, and $t_1 \xrightarrow{l^{tmp}} t_1^{tmp}$ is a transition in Θ^1, there is a transition $t_1 \xrightarrow{l'} t_1'$ in Θ^1, where $t_1^{tmp} \preceq_1 t_1'$.

We now have (a) transitions $t_1 \xrightarrow{l'} t_1'$ in Θ^1 and $t_2 \xrightarrow{l'} t_2'$ in Θ^2, where $c(l') \leq c(l^{tmp}) \leq c(l)$. We furthermore have $s_1' \preceq_1 t_1^{tmp}$ and $t_1^{tmp} \preceq_1 t_1'$, from which because all relations in a label-dominance simulation must be transitive we have that (b) $s_1' \preceq_1 t_1'$. Similarly, from $s_2' \preceq_2 s_2^{tmp}$ and $s_2^{tmp} \preceq_2 t_2'$ we get (c) $s_2' \preceq_2 t_2'$. Together, (a–c) clearly show what we needed to prove.

Consider now the relations $\preceq_3, \ldots, \preceq_k$. Since these are unchanged from the original label-dominance simulation $\mathcal{R}$, for their part of the claim it suffices to prove that, if l' dominates l in Θ^1 with respect to $\preceq_1$ and in Θ^2 with respect to $\preceq_2$, then l' dominates l in Θ^{12} with respect to $\preceq_{12}$. Hence we have to prove that, for every transition $(s_1, s_2) \xrightarrow{l} (t_1, t_2)$, there exists another transition $(s_1, s_2) \xrightarrow{l'} (t_1', t_2')$ where $(t_1, t_2) \preceq_{12} (t_1', t_2')$.

As l' dominates l in Θ^1 with respect to $\preceq_1$, there exists a transition $s_1 \xrightarrow{l'} t_1'$ in Θ^1 with $t_1 \preceq_1 t_1'$. As l' dominates l in Θ^2 with respect to $\preceq_2$, there exists a transition $s_2 \xrightarrow{l'} t_2'$ in Θ^2 with $t_2 \preceq_2 t_2'$. The claim follows directly from the definitions of the synchronized product and $\preceq_{12}$. □

References

Bryant, R. E. 1986. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers* 35(8):677–691.

Chen, Y., and Yao, G. 2009. Completeness and optimality preserving reduction for planning. In *Proc. 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, 1659–1664.

Domshlak, C.; Katz, M.; and Shleyfman, A. 2012. Enhanced symmetry breaking in cost-optimal planning as forward search. In *Proc. 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*.

Dräger, K.; Finkbeiner, B.; and Podelski, A. 2006. Directed model checking with distance-preserving abstractions. In

Proceedings of the 13th International SPIN Workshop (SPIN 2006), volume 3925 of *Lecture Notes in Computer Science*, 19–34.

Dräger, K.; Finkbeiner, B.; and Podelski, A. 2009. Directed model checking with distance-preserving abstractions. *STTT* 11(1):27–37.

Fox, M., and Long, D. 1999. The detection and exploitation of symmetry in planning problems. In *Proc. 16th International Joint Conference on Artificial Intelligence (IJCAI-99)*, 956–961.

Fox, M., and Long, D. 2002. Extending the exploitation of symmetries in planning. In *Proc. 6th International Conference on Artificial Intelligence Planning and Scheduling (AIPS-02)*, 83–91.

Gentilini, R.; Piazza, C.; and Policriti, A. 2003. From bisimulation to simulation: Coarsest partition problems. *Journal of Automated Reasoning* 31(1):73–103.

Grumberg, O., and Long, D. E. 1994. Model checking and modular verification. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 16(3):843–871.

Hall, D.; Cohen, A.; Burkett, D.; and Klein, D. 2013. Faster optimal planning with partial-order pruning. In *Proc. 23rd International Conference on Automated Planning and Scheduling (ICAPS'13)*.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In *Proc. 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*, 162–169.

Helmert, M., and Röger, G. 2008. How good is almost perfect? In *Proc. 23rd National Conference of the American Association for Artificial Intelligence (AAAI-08)*, 944–949.

Helmert, M.; Haslum, P.; Hoffmann, J.; and Nissim, R. 2014. Merge & shrink abstraction: A method for generating lower bounds in factored state spaces. *Journal of the Association for Computing Machinery* 61(3).

Helmert, M.; Haslum, P.; and Hoffmann, J. 2007. Flexible abstraction heuristics for optimal sequential planning. In *Proc. 17th International Conference on Automated Planning and Scheduling (ICAPS'07)*, 176–183.

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

Henzinger, M. R.; Henzinger, T. A.; and Kopke, P. W. 1995. Computing simulations on finite and infinite graphs. In *36th Annual Symposium on Foundations of Computer Science.*, 453–462.

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

Loiseaux, C.; Graf, S.; Sifakis, J.; Bouajjani, A.; and Bensalem, S. 1995. Property preserving abstractions for the verification of concurrent systems. *Formal Methods in System Design* 6(1):11–44.

Milner, R. 1971. An algebraic definition of simulation between programs. In *Proc. 2nd International Joint Conference on Artificial Intelligence (IJCAI-71)*, 481–489.

Sievers, S.; Wehrle, M.; and Helmert, M. 2014. Generalized label reduction for merge-and-shrink heuristics. In *Proc. 28th AAAI Conference on Artificial Intelligence (AAAI'14)*, 2358–2366.

Valmari, A. 1989. Stubborn sets for reduced state space generation. volume 483 of *Lecture Notes in Computer Science*, 491–515.

Wehrle, M., and Helmert, M. 2012. About partial order reduction in planning and computer aided verification. In *Proc. 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*.

Wehrle, M., and Helmert, M. 2014. Efficient stubborn sets: Generalized algorithms and selection strategies. In *Proc. 24th International Conference on Automated Planning and Scheduling (ICAPS'14)*.

Wehrle, M.; Helmert, M.; Alkhazraji, Y.; and Mattmüller, R. 2013. The relative pruning power of strong stubborn sets and expansion core. In *Proc. 23rd International Conference on Automated Planning and Scheduling (ICAPS'13)*.