Alan Fern Patrik Haslum Jörg Hoffmann Michael Katz (Eds.)

HDIP 2011

3rd Workshop on Heuristics for Domain-independent Planning

Co-located with ICAPS 2011

Freiburg, Germany, June 12, 2011

Proceedings

Editors' addresses:

afern@eecs.oregonstate.edu, patrik.haslum@anu.edu.au, joerg.hoffmann@loria.fr, mkatz@ie.technion.ac.il

Preface

The automatic derivation of heuristic estimators to guide the search has become one of the main approaches in domain-independent planning. This workshop aims to provide a forum for discussing current advances in automatic derivation of heuristic estimators, as well as other related fields.

The workshop on heuristics for domain-independent planning so far took place in conjunction with ICAPS 2007 and ICAPS 2009, and was very successful both times. Many ideas presented at these workshops have led to contributions at major conferences and pushed the frontier of research on heuristic planning in several directions, both theoretically and practically. The workshops, as well as work on heuristic search that has been published since then, have also shown that there are many exciting open research opportunities in this area. Given the considerable success of the past workshops, we intend to continue holding it biennially.

We look for contributions that would help us understand better the ideas underlying current heuristics, their limitations, and the ways for overcoming them. Contributions do not have to show that a new heuristic or new variation of a known heuristic 'beats the competition'. Above all we seek crisp and meaningful ideas and understanding. Also, rather than merely being interested in the 'largest' problems that current heuristic search planners can solve, we are equally interested in the simplest problems that they can't actually solve well.

The workshop series, while having originated mainly in classical planning, is very much open to new ideas on heuristic schemes for more general settings, such as temporal planning, planning under uncertainty and adversarial planning. We are happy to present a varied program including papers addressing temporal planning, hierarchical planning, problem symmetries, SAT, and Bayesian Networks. Apart from these, the program features several new ideas for computing improved heuristics in classical planning as well as improving the existing ones.

We hope that the workshop will constitute one more step towards a better understanding of the ideas underlying current heuristics, of their limitations, and of ways for overcoming those.

We thank the authors for their submissions and the program committee for their hard work.

June 2011 Alan Fern, Patrik Haslum, Jörg Hoffmann, and Michael Katz.

Organizing Committee

Program Committee

Contents

Computing Perfect Heuristics in Polynomial Time: On Bisimulation and Merge-and-Shrink Abstraction in Optimal Planning

Raz Nissim
Ben-Gurion University
Beer-Sheva, Israel
raznis@cs.bgu.ac.il

Jörg Hoffmann
INRIA
Nancy, France
joerg.hoffmann@inria.fr

Malte Helmert
University of Freiburg
Freiburg, Germany
helmert@informatik.uni-freiburg.de

Abstract

A* with admissible heuristics is a very successful approach to optimal planning. But how to derive such heuristics automatically? Merge-and-shrink abstraction (M&S) is a general approach to heuristic design whose key advantage is its capability to make very fine-grained choices in defining abstractions. However, little is known about how to actually make these choices. We address this via the well-known notion of *bisimulation*. When aggregating only bisimilar states, M&S yields a perfect heuristic. Alas, bisimulations are exponentially large even in trivial domains. We show how to apply *label reduction* – not distinguishing between certain groups of operators – without incurring any information loss, while potentially reducing bisimulation size exponentially. In several benchmark domains, the resulting algorithm computes perfect heuristics in polynomial time. Empirically, we show that approximating variants of this algorithm improve the state of the art in M&S heuristics. In particular, a hybrid of two such variants is competitive with the leading heuristic LM-cut.

Introduction

Many optimal planning systems are based on state-space search with A* and admissible heuristics. The research question is how to derive heuristics automatically. Merge-and-shrink abstraction (Dräger *et al.* 2006; Helmert *et al.* 2007), in short M&S, uses solution distance in smaller, *abstract* state spaces to yield consistent, admissible heuristics.

The abstract state space is built in an incremental fashion, starting with a set of atomic abstractions corresponding to individual variables, then iteratively *merging* two abstractions – replacing them with their synchronized product – and *shrinking* them – aggregating pairs of states into one. Thus, despite the exponential size of the state space, M&S allows to select individual pairs of states to aggregate. This freedom in abstraction design comes with significant advantages. M&S dominates most other known frameworks for computing admissible planning heuristics: for any given state, it can with polynomial overhead compute a larger lower bound (Helmert and Domshlak 2009). Further, in difference to most other known frameworks, M&S is able to compute, in polynomial time, perfect heuristics for some benchmark domains where optimal planning is easy (Helmert *et al.* 2007; Helmert and Mattmüller 2008).

M&S currently does not live up to its promises: (A) in the international planning competition (IPC) benchmarks, it gives worse empirical performance than the currently leading heuristics, in particular LM-cut (Helmert and Domshlak 2009); (B) it does not deliver perfect heuristics in the benchmarks where it could. The theoretical power of M&S hinges on the ability to take perfect decisions as to which pairs of states to aggregate. Little is known about how to take such decisions in practice. We herein address this issue, largely solving (B), and making headway towards solving (A).

Our investigation is based on the notion of *bisimulation*, a well-known criterion under which an abstract state space "exhibits the same observable behavior" as the original state space (Milner 1990). Two states s, t are bisimilar if: (1) they agree on whether or not the goal is true; and (2) every transition label, i.e., every planning operator, leads into the same abstract state from both s and t. If we aggregate only bisimilar states during M&S, then the heuristic is guaranteed to be perfect. A coarsest bisimulation can be efficiently constructed, and thus this offers a practical strategy for selecting the states to aggregate. Indeed, this was observed already by Dräger et al. (2006) (in a model checking context). However, bisimulations are exponentially big even in trivial examples, including the aforementioned benchmarks (B). Our key observation is that, for the purpose of computing a heuristic, we can relax bisimulation significantly without losing any information. Namely, we do not need to distinguish the transition labels. Such a *fully label-reduced* bisimulation still preserves solution distance, while often being exponentially smaller.

Unfortunately, while full label reduction does not affect solution distances per se, its application within the M&S framework is problematic. The merging step, in order to synchronize transitions, needs to know which ones share the same label, i.e., correspond to the same operator. We tackle this by using *partial* label reductions, ignoring the difference between two labels only if they are equivalent for "the rest" of the M&S construction. We thus obtain, again, a strategy that guarantees to deliver a perfect heuristic. This method largely solves challenge (B), in that its runtime is polynomially bounded in most of the relevant domains.

Even label-reduced bisimulations are often prohibitively big, thus for practicality one needs a strategy to approximate further if required. We experiment with a variety of such strategies, and examine their performance empirically. Each single strategy still is inferior to LM-cut. However, the different strategies exhibit complementary strengths, i.e., they work well in different domains. We thus experiment with simple hybrid planners running two of the stragies sequentially. We find that, addressing challenge (A), one of these hybrids is competitive with LM-cut in the IPC benchmarks, and even outperforms it when ignoring the Miconic domain.

For space reasons, we omit many details and only outline proofs. Full details are available from the authors on request.

Background

We consider optimal sequential planning with finite-domain variables. A **planning task** is a 4-tuple $(\mathcal{V}, \mathcal{O}, s_0, s_\star)$. $\mathcal{V}$ is a finite set of **variables**, where each $v \in \mathcal{V}$ is associated with a finite domain $\mathcal{D}_v$. A **partial state** over $\mathcal{V}$ is a function s on a subset $\mathcal{V}_s$ of $\mathcal{V}$, so that $s(v) \in \mathcal{D}_v$ for all $v \in \mathcal{V}_s$; s is a **state** if $\mathcal{V}_s = \mathcal{V}$. The **initial state** s_0 is a state. The **goal** $s_\star$ is a partial state. $\mathcal{O}$ is a finite set of **operators**, each being a pair (pre, eff) of partial states, called **precondition** and **effect**.

The semantics of planning tasks are, as usual, defined via their **state spaces**, which are (labeled) **transition systems**. Such a system is a 5-tuple $\Theta = (S, L, T, s_0, S_\star)$ where S is a finite set of **states**, L is a finite set of **transition labels**, $T \subseteq S \times L \times S$ is a set of **labeled transitions**, $s_0 \in S$ is the **start state**, and $S_\star \subseteq S$ is the set of **solution states**. In the state space of a planning task, S is the set of all states; s_0 is identical with the initial state of the task; $s \in S_\star$ if $s_\star \subseteq s$; the transition labels L are the operators $\mathcal{O}$; and $(s, (pre, eff), s') \in T$ if s complies with pre, $s'(v) = eff(v)$ for $v \in \mathcal{V}_{eff}$, and $s'(v) = s(v)$ for $v \in \mathcal{V} \setminus \mathcal{V}_{eff}$.

A **plan** is a path from s_0 to any $s_\star \in S_\star$. The plan is **optimal** iff its length is equal to $sd(s_0)$, where $sd : S \rightarrow \mathbb{N}_0$ assigns to s the length of a shortest path from s to any $s_\star \in S_\star$, or $sd(s) = \infty$ if there is no such path.

A **heuristic** is a function $h : S \rightarrow \mathbb{N}_0 \cup \{\infty\}$. The heuristic is **admissible** iff, for every $s \in S$, $h(s) \leq sd(s)$; it is **consistent** iff, for every $(s, l, s') \in T$, $h(s) \leq h(s') + 1$. As is well known, A* with an admissible heuristic returns an optimal solution, and does not need to re-open any nodes if the heuristic is consistent. We will also consider **perfect** heuristics, that coincide with sd. If we know that h is perfect, then we can extract an optimal plan without any search.

How to automatically compute a heuristic, given a planning task as input? Our approach is based on designing an **abstraction**. This is a function α mapping S to a set of **abstract states** S^α. The **abstract state space** Θ^α is defined as $(S^\alpha, L, T^\alpha, s_0^\alpha, S_\star^\alpha)$, where $T^\alpha := \{(\alpha(s), l, \alpha(s')) \mid (s, l, s') \in T\}$, $s_0^\alpha := \alpha(s_0)$, and $S_\star^\alpha := \{\alpha(s_\star) \mid s_\star \in S_\star\}$. The **abstraction heuristic** h^α maps each $s \in S$ to the solution distance of $\alpha(s)$ in Θ^α; h^α is admissible and consistent. We will sometimes consider the **induced equivalence relation** $\sim^\alpha$, defined by setting $s \sim^\alpha t$ iff $\alpha(s) = \alpha(t)$.

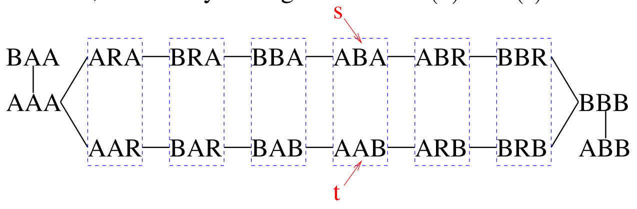


Figure 1: An abstraction of (simplified) Gripper.

To illustrate abstractions, Figure 1 gives an example. In the Gripper benchmark, one needs to transport n objects from a room A into a room B, with a robot R that can carry two objects at a time. We show here the state space for $n = 2$, in the simplified situation where the robot can carry only a single object. The planning task has 4 variables, R with $\mathcal{D}_R = \{A, B\}$, F with $\mathcal{D}_F = \{0, 1\}$ (hand free?), O_i with $\mathcal{D}_{O_i} = \{A, B, R\}$ for $i = 1, 2$. States in Figure 1 are shown as triples giving the value of R, O_1, and O_2 in this order (omitting F, whose value is implied). For example, in the state marked "s", the robot is at A, O_1 is at B, O_2 is at A (and $F = 1$). The abstraction α is indicated by the (blue) dashed boxes. This abstraction aggregates states – assigns them to the same abstract state – iff they agree on the status of the robot and on the number of objects in each room. Thus the abstraction does not distinguish the upper solution path (transporting O_1 first) from the lower one (transporting O_2 first). This does not affect solution length, so h^α is perfect. The same applies for arbitrary n, yielding perfect heuristics and polynomial-sized (measured in $|S^\alpha|$) abstractions.

How to choose a good α in general? Pattern databases (PDBs) (Edelkamp 2001) simplify this question by limiting α to be a **projection**. Given $V \subseteq \mathcal{V}$, the projection π_V onto V is defined by setting $\pi_V(s)$ to be the restriction of s onto V. π_V can be computed very efficiently, and the solution distances in Θ^{π_V} can be pre-computed and stored in a table (the PDB) prior to search.

The downside of PDBs is their lack of flexibility in abstraction design. To be computationally efficient, any PDB can consider only a small subset of variables. Thus, even in trivial domains, PDBs cannot compactly represent the perfect heuristic (Helmert *et al.* 2007). For example, in Gripper, any polynomial-sized PDB considers only a logarithmic number of objects. The position of the other objects will be abstracted away, thus under-estimating sd to an arbitrarily large extent. For example, in Figure 1, if $V = \{R, O_1\}$, then s and its right neighbor are aggregated, shortening the upper solution path. Summing over "additive PDBs" doesn't help, since at most one PDB considers the robot position.

Inspired by work in the context of model checking automata networks (Dräger *et al.* 2006), Helmert et al. (2007) propose M&S abstraction as an alternative allowing more fine-grained abstraction design, selecting individual pairs of states to aggregate. To make such selection feasible in exponentially large state spaces, the approach builds the abstraction incrementally, iterating between *merging* and *shrinking* steps. In detail, an abstraction α is a **M&S abstraction over** $V \subseteq \mathcal{V}$ if it can be constructed using these rules:

(i) For $v \in \mathcal{V}$, $\pi_{\{v\}}$ is an M&S abstraction over $\{v\}$.

(ii) If β is an M&S abstraction over V and γ is a function on S^β, then $\gamma \circ \beta$ is an M&S abstraction over V.

(iii) If α_1 and α_2 are M&S abstractions over disjoint sets V_1 and V_2, then $\alpha_1 \otimes \alpha_2$ is an M&S abstraction over $V_1 \cup V_2$.

It is important to keep these rules in mind since we will be referring back to them throughout the paper. **Rule (i)** allows to start from **atomic projections**, i.e., projections $\pi_{\{v\}}$ onto a single variable, also written π_v in the rest of this paper. **Rule (ii)**, the **shrinking step**, allows to iteratively aggregate an arbitrary number of state pairs, in abstraction β. Formally, this simply means to apply an additional abstraction γ to the image of β. In **rule (iii)**, the **merging step**, the merged abstraction $\alpha_1 \otimes \alpha_2$ is defined by $(\alpha_1 \otimes \alpha_2)(s) := (\alpha_1(s), \alpha_2(s))$. It is easy to see that this definition generalizes PDBs:[1]

Proposition 1 (Helmert et al., 2007) *Let Θ be the state space of a planning task with variables $\mathcal{V}$, and let α be an M&S abstraction over $V \subseteq \mathcal{V}$ constructed using only rules (i) and (iii). Then Θ^α is isomorphic to Θ^{π_V}.*

Moreover, as Helmert et al. (2007) show, M&S *strictly* generalizes PDBs in that it can compactly represent the perfect heuristic in domains where PDBs cannot. For example, in Gripper we obtain the perfect heuristic by aggregating states as exemplified in Figure 1.

Proposition 1 holds even if we drop the constraint $V_1 \cap V_2 = \emptyset$ in rule (iii). That constraint plays an important role in the *computation* of h^α. Note that, a priori, this issue is separate from the *design* of α. We follow Helmert et al. (2007) in that, while designing α, we maintain also the abstract state space Θ^α (and are thus able to compute h^α). In a little more detail, we maintain a transition system Θ_α, in a way so that Θ_α is identical with the (mathematically defined) abstract state space Θ^α of α; note the use of α as a subscript respectively superscript to distinguish these two. The correct maintenance of Θ_α is trivial for rules (i) and (ii), but is a bit tricky for rule (iii). We need to compute the abstract state space $\Theta^{\alpha_1 \otimes \alpha_2}$ of $\alpha_1 \otimes \alpha_2$, based on the transition systems Θ_{α_1} and Θ_{α_2} computed for α_1 and α_2 beforehand. As an induction hypothesis, assume that $\Theta_{\alpha_1} = \Theta^{\alpha_1}$ and $\Theta_{\alpha_2} = \Theta^{\alpha_2}$. We compute $\Theta^{\alpha_1 \otimes \alpha_2}$ as the **synchronized product** $\Theta^{\alpha_1} \otimes \Theta^{\alpha_2}$. This is a standard operation, its state space being $S^{\alpha_1} \times S^{\alpha_2}$, with a transition from (s_1, s_2) to (s_1', s_2') via label l iff $(s_1, l, s_1') \in T^{\alpha_1}$ and $(s_2, l, s_2') \in T^{\alpha_2}$. For this to be correct, i.e., to have $\Theta^{\alpha_1} \otimes \Theta^{\alpha_2} = \Theta^{\alpha_1 \otimes \alpha_2}$, the constraint $V_1 \cap V_2 = \emptyset$ is required. Namely, this constraint ensures that α_1 and α_2 are **orthogonal** in the sense of Helmert et al. (2007), meaning basically that there exists no variable on whose value both abstractions depend. Then:

Theorem 2 (Helmert et al., 2007) *Let Θ be the state space of a planning task, and let α_1 and α_2 be orthogonal abstractions of Θ. Then $\Theta^{\alpha_1} \otimes \Theta^{\alpha_2} = \Theta^{\alpha_1 \otimes \alpha_2}$.*

In practice, we need a *merging strategy* deciding which abstractions to merge in (iii), and a *shrinking strategy* deciding which (and how many) states to aggregate in (ii). Helmert et al. do not investigate this in detail. Our main issue herein is with their shrinking strategy. This aggregates states until a size limit N – an input parameter – is reached. The strategy is based exclusively on the initial state and goal distances in the abstract state space at hand; it is *f-preserving* in that (if possible) it aggregates states only if they agree on these distances. This strategy preserves distances *locally* – in the abstraction at hand – but does not take into account at all the *global* impact of aggregating states.

[1]Helmert et al. do not state Proposition 1 (or Theorem 2 below) in this form, but they follow trivially from their discussion.

```
$v_1, \dots, v_n$ := an ordering of $\mathcal{V}$
$\alpha := \pi_{v_1}$, $\Theta_\alpha := \Theta^{\pi_{v_1}}$ /* rule (i) */
$\sigma^1$ := function projecting operators onto $\{v_2, \dots, v_n\}$
apply $\sigma^1$ to transition labels in $\Theta_\alpha$
for $i := 2, \dots, n$ do
    $\alpha' := \pi_{v_i}$, $\Theta_{\alpha'} := \Theta^{\pi_{v_i}}$ /* rule (i) */
    apply $\sigma^{i-1}$ to transition labels in $\Theta_{\alpha'}$
    $\alpha := \alpha \otimes \alpha'$, $\Theta_\alpha := \Theta_\alpha \otimes \Theta_{\alpha'}$ /* rule (iii) */
    $\sigma^i$ := function projecting operators onto $\{v_{i+1}, \dots, v_n\}$
    apply $\sigma^i$ to transition labels in $\Theta_\alpha$
    $\sim$:= coarsest bisimulation for $\Theta_\alpha$
    aggregate all states $s, t$ in $\alpha$ and $\Theta_\alpha$ where $s \sim t$ /* rule (ii) */
endfor
return $\alpha$ and $\Theta_\alpha$
```

Figure 2: Overview of M&S-bop algorithm.

For example, in a transportation domain, if we consider only the position of a truck, then any states s, t equally distant from the truck's initial and target position can be aggregated: locally, the difference is irrelevant. Globally, however, there are transportable objects to which the difference in truck positions does matter, and thus aggregating s and t results in information loss. Therefore, the heuristic computed is *not* perfect although, theoretically, it could be.

M&S-bop Overview

Figure 2 outlines our algorithm computing perfect heuristics, **M&S-bop** (M&S with **b**isimulation and **op**erator pruning).

The variable ordering $v_1, \dots, v_n$ will be defined by the merging strategy (a simple heuristic, cf. Section). Note that the merging strategy is **linear**, i.e., α' is always atomic in the application of rule (iii). Iterating over $v_1, \dots, v_n$, the algorithm maintains an abstraction α, along with the transition system Θ_α, as described earlier. In this way, M&S-bop is a straightforward instance of the M&S framework. Its distinguishing features are (a) the label reduction functions σ^i that remove operator preconditions/effects pertaining to variables indexed $\leq i$, and (b) coarsest bisimulation over the label-reduced abstract state spaces. (b) defines the shrinking strategy, which is influenced by (a) because reducing labels changes the bisimulation. Reducing labels may also change the outcome of the synchronized product (rule (iii)), therefore (a) endangers correctness of Θ_α relative to Θ^α.

In the following three sections, we fill in the formal details on (a) and (b), proving (in particular) that the abstraction heuristic h^α of α as returned by M&S-bop is perfect, and can be extracted from the returned transition system Θ_α. We begin by defining bisimulation, and pointing out some basic facts about its behavior in the M&S framework. We then formally define label reduction and the conditions under which it preserves the correctness of Θ_α. We finally point out that the two techniques can be combined fruitfully.

Bisimulation

Bisimulation is a well-known criterion under which abstraction does not significantly change the system behavior. Let $\Theta = (S, L, T, s_0, S_\star)$ be a transition system. An equivalence relation $\sim$ on S is a **bisimulation** for Θ if: (1) $s \sim t$ implies that either $s, t \in S_\star$ or $s, t \notin S_\star$; (2) for every pair of states $s, t \in S$ so that $s \sim t$, and for every transition label $l \in L$, if $(s, l, s') \in T$ then there exists t' s.t. $(t, l, t') \in T$ and $s' \sim t'$. Intuitively, (1) s and t agree on the status of the goal, and (2) whatever operator applies to s applies also to t, leading into equivalent states. An abstraction α is a bisimulation iff the induced equivalence relation $\sim^\alpha$ is.

We first consider a well-known criterion under which abstraction does not significantly change the system behavior. Let $\Theta = (S, L, T, s_0, S_\star)$ be a transition system. An equivalence relation $\sim$ on S is a **bisimulation** for Θ if: (1) $s \sim t$ implies that either $s, t \in S_\star$ or $s, t \notin S_\star$; (2) for every pair of states $s, t \in S$ so that $s \sim t$, and for every transition label $l \in L$, if $(s, l, s') \in T$ then there exists t' s.t. $(t, l, t') \in T$ and $s' \sim t'$. If α is an abstraction, then we say that α is a bisimulation iff the induced equivalence relation $\sim^\alpha$ is.

For a comprehensive treatment of bisimulation, see the work by Milner (1990). There always exists a unique **coarsest bisimulation**, i.e., a bisimulation that contains all other bisimulations. The coarsest bisimulation can be computed efficiently, in a bottom-up process starting with a single equivalence class containing all states, then iteratively separating non-bisimilar states. Using this process as a shrinking strategy preserves solution distance at the local level:

Proposition 3 *Let Θ be a transition system, and let α be a bisimulation for Θ. Then h^α is perfect.*

This holds because *abstract solution paths are real solution paths*. Consider some state t, and an abstract solution for t. Say the abstract solution starts with the transition (A, l, A'), where A and A' are abstract states. Since (A, l, A') is a transition in Θ^α, there exist states $s \in A$ and $s' \in A'$ so that (s, l, s') is a transition in Θ. By construction, we have $\alpha(t) = A = \alpha(s)$ and thus $s \sim^\alpha t$. Thus, by bisimulation property (2), we have a transition (t, l, t') in Θ where $s' \sim^\alpha t'$, i.e., $\alpha(t') = \alpha(s') = A'$. In other words, there exists a transition from t taking us into the desired state subset A'. Iterating the argument, we can execute all transitions on the abstract solution for t. The last state t' reached this way must be a solution state because of bisimulation property (1) and the fact that $\alpha(t')$ is a solution state in Θ^α.

Note that the argument we just made is much stronger than what is needed to prove that h^α is perfect – we preserve the actual solutions, not only their length. That is exactly what we will exploit further below. First, note that Proposition 3 tells us nothing about what happens if we interleave bisimulation with merging steps. This works, too:

Corollary 4 *Let Θ be the state space of a planning task with variables $\mathcal{V}$. Let α be an M&S abstraction over $V \subseteq \mathcal{V}$ constructed so that, in any application of rule (ii), γ is a bisimulation for Θ^β. Then α is a bisimulation for Θ^{π_V}.*

This can be shown by induction over rules (i)-(iii). The claim is obvious for atomic α. Induction over rule (ii) is easy by exploiting a transitivity property of bisimulations. For rule (iii), assume that α_1 is a bisimulation for $\Theta^{\pi_{V_1}}$, and α_2 is a bisimulation for $\Theta^{\pi_{V_2}}$. Since $\Theta^{\pi_{V_1}}$ and $\Theta^{\pi_{V_2}}$ are orthogonal, by Theorem 2 their synchronized product is equal to $\Theta^{\pi_{V_1} \otimes \pi_{V_2}}$ which is isomorphic with $\Theta^{\pi_{V_1 \cup V_2}}$. We can apply α_1 and α_2 to the states in $\Theta^{\pi_{V_1 \cup V_2}}$ (applying their component atomic projections to partial states), and it is easy to see that the bisimulation property is preserved.[2]

By Corollary 4 and Proposition 3, if we always stick to bisimulation during step (ii), then we obtain a perfect heuristic. Alas, in practical examples there rarely exist compact bisimulations. For example, in Gripper, the perfect abstraction of Figure 1 is *not* a bisimulation. If s and t agree on the number of objects in a room, this does not imply that they agree on the operators needed to transport them. For example, s and t as indicated in Figure 1 require to pick up O_2 (s) vs. O_1 (t). Thus the transition labels into the equivalent states (right neighbors in Figure 1) are different, violating property (2). Indeed, it is easy to see that the size of bisimulations in Gripper is exponential in the number of objects.

Motivated by the size of bisimulations, Dräger et al. (2006) propose a more approximate shrinking strategy that we will call the **DFP shrinking strategy**. When building the coarsest bisimulation, the strategy keeps separating states until the size limit N is reached. The latter may happen before a bisimulation is obtained, in which case we may lose information. The strategy prefers to separate states close to the goal, thus attempting to make errors only in more distant states where the errors will hopefully not be as relevant.

The DFP shrinking strategy is not a bad idea; some of the strategies we experiment with herein are variants of it. However, before resorting to approximations, there is a more fundamental issue we can improve. As noted in the proof of Proposition 3, bisimulation preserves solution paths exactly. This suits its traditional purpose in model checking. For computing a perfect heuristic, however, it suffices to preserve solution *length*. In bisimulation property (2), we can completely ignore the transition labels. This makes all the difference in Gripper. States s and t in Figure 1 *can* reach the same equivalence classes, but using different labels. Ignoring the latter, s and t are bisimilar. The perfect abstraction of Figure 1 is such a *fully label-reduced* bisimulation.

Label Reduction

Unfortunately, full label reduction cannot be applied within the M&S framework, at least not without significant information loss during the merging step. Figure 3 shows the atomic projections of our running example from Figure 1. For simplicity, we omit F and show the two variables O_1 and O_2 in terms of a single generic variable O. Consider the transition label $l = (\{R = A, O = A\}, \{O = R\})$ picking up the object in room A. This transitions O from A to R in Θ^{π_O} (right hand side), but does not affect R and hence labels only a self-loop in Θ^{π_R} (left hand side). Hence, in the synchronized system $\Theta^{\pi_R} \otimes \Theta^{\pi_O}$, as desired the robot does not move while loading the object. If we ignore the difference between l and the label $l' = (\{R = A\}, \{R = B\})$ moving R, however, then the synchronized system may apply l and l' together, acting as if they were the result of applying the same operator. Thus we may move and pick-up at the same time – a spurious transition not present in the original task.

[2]Dräger et al. (2006) mention a result similar to Corollary 4. The result is simpler in their context because, there, the overall state space is *defined* in terms of the synchronized product operation.

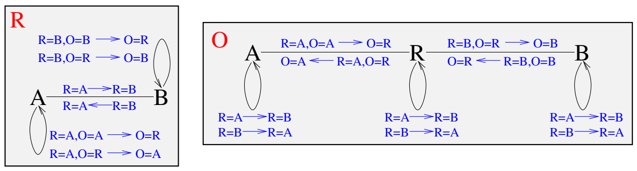


Figure 3: Atomic projections in (simplified) Gripper.

We now derive a way of reducing subsets of labels during M&S, so that no spurious transitions are introduced, and thus the correctness of Θ_α relative to Θ^α is preserved. We remark that a simpler version of the technique, pertaining only to linear merging strategies, was implemented already in the initial version of M&S reported by Helmert et al. (2007); the technique has not yet been described anywhere.

Let $\Theta = (S, L, T, s_0, S_\star)$ be a transition system. A **label reduction** is a function τ mapping L to a label set L^τ. We associate τ with the **reduced transition system** $\Theta|_\tau := (S, L^\tau, T|_\tau, s_0, S_\star)$ where $T|_\tau := \{(s, \tau(l), s') \mid (s, l, s') \in T\}$. Labels $l^1, l^2 \in L$ are **equivalent in** Θ if, for every pair of states $s, s' \in S$, $(s, l^1, s') \in T$ if and only if $(s, l^2, s') \in T$. We say that τ is **conservative for** Θ if, for all $l^1, l^2 \in L$, $\tau(l_1) = \tau(l_2)$ only if l_1 and l_2 are equivalent. Such label reduction is distributive with the synchronized product:

Lemma 5 *Let L be a set of labels, let Θ^1 and Θ^2 be transition systems using L, and let τ be a label reduction on L. If τ is conservative for Θ^2, then $\Theta^1|_\tau \otimes \Theta^2|_\tau = (\Theta^1 \otimes \Theta^2)|_\tau$.*

For illustration, consider the labels $l = (\{R = A, O = A\}, \{O = R\})$ and $l' = (\{R = A\}, \{R = B\})$ discussed at the start of this section. These are not equivalent in Θ^{π_O} (Figure 3 right), and if $\tau(l) = \tau(l')$ then, as discussed, $\Theta^{\pi_R}|_\tau \otimes \Theta^{\pi_O}|_\tau \neq (\Theta^{\pi_R} \otimes \Theta^{\pi_O})|_\tau$ in contrast to Lemma 5.

We apply conservative label reduction within M&S by, when considering an abstraction over variable subset V, applying a label reduction conservative for the other variables $\mathcal{V} \setminus V$. A key issue here is that one cannot "reduce in opposite directions". To illustrate, say Θ^1 (variables V) has the single label l^1 and Θ^2 (variables $\mathcal{V} \setminus V$) has the single label $l^2 \neq l^1$. Then $\Theta^1 \otimes \Theta^2$ contains no transitions at all. However, mapping all labels to some unique symbol r is conservative for each of Θ^1 and Θ^2. If we synchronize the systems after this reduction, we obtain a transition between every pair of states.

To avoid said difficulties, we apply label reduction upwards in a sequential ordering of the variables (given by the merging strategy). Let $V \subseteq \mathcal{V}$, and let $v_1, \dots, v_n$ be an ordering of V. We say that an M&S abstraction α over V **allows** $v_1, \dots, v_n$ if α is constructed so that, in any application of rule (iii), there exist i, j, k so that $V_1 = \{v_i, \dots, v_j\}$ and $V_2 = \{v_{j+1}, \dots, v_k\}$. In other words, the construction of α corresponds to a tree whose leaves are ordered $v_1, \dots, v_n$.

Let $\sigma^n \circ \dots \circ \sigma^1$ be a chain of label reductions for Θ. We denote $\tau^{>i} := \sigma^i \circ \dots \circ \sigma^1$. We say that $\sigma^n \circ \dots \circ \sigma^1$ is **conservative for** $v_1, \dots, v_n$ if, for each $1 \leq i < n$, $\tau^{>i}$ is conservative for each of $\Theta^{\pi_{v_{i+1}}}, \dots, \Theta^{\pi_{v_n}}$. Note that the last reduction σ^n is not restricted at all, and may thus be a full label reduction mapping all labels to the same symbol.

In practice, the label reductions σ^i we use are those from M&S-bop (cf. Figure 2), projecting operators onto $\{v_{i+1}, \dots, v_n\}$, i.e., removing any preconditions/effects pertaining to the variables $v_1, \dots, v_i$ that have already been merged. This chain of label reductions is conservative:

Proposition 6 *Let Θ be the state space of a planning task with variables $\mathcal{V}$. Let $V \subseteq \mathcal{V}$, and let $v_1, \dots, v_n$ be an ordering of V. Then operator projection is conservative for $v_1, \dots, v_n$, and is maximal among all chained label reductions with this property.*

For the first part of the claim, we need $\tau^{>i} = \sigma^i$ to be conservative – map only equivalent labels $(pre, eff), (pre', eff')$ to the same label – for each of $\Theta^{\pi_{v_{i+1}}}, \dots, \Theta^{\pi_{v_n}}$. This holds because *(*) (pre, eff) and (pre', eff') are equivalent in $\Theta^{\pi_{v_j}}$ if and only if their projection onto v_j is identical*. The second part of the claim, maximality, holds in that $\tau^{>i}$ maps *all* equivalent labels to the same label. This follows from the "only if" direction in (*).

Say that in our running example (omitting F) the variable order is $R, O_1, O_2, \dots, O_n$. Consider again the labels $l = (\{R = A, O = A\}, \{O = R\})$ and $l' = (\{R = A\}, \{R = B\})$. For $i = 1$, the robot move l' is projected onto $\tau^{>1}(l') = (\emptyset, \emptyset)$, ignoring the robot's position. However, $\tau^{>1}(l) = (\{O = A\}, \{O = R\})$, so $\tau^{>1}(l) \neq \tau^{>1}(l')$ as desired. On the other hand, for $i > 1$ and every $j \leq i$, all labels in $\Theta^{\pi_{O_j}}$ will be mapped to $(\emptyset, \emptyset)$ (compare Figure 3 right), so the differences between any "already merged" objects will effectively be ignored. For $i = n$, *all* labels are mapped to $(\emptyset, \emptyset)$ so the label reduction is full.

With conservative label reductions, for α that allows $v_1, \dots, v_n$, we can maintain Θ_α during M&S similarly as before. Namely, the **label-reduced transition system associated with** α, written Θ^τ_α, is constructed using these rules:

(i) If $\alpha = \pi_{v_i}$ for a variable $v_i \in V$, then $\Theta^\tau_\alpha := \Theta^{\pi_{v_i}}$ if $i \neq 1$, and $\Theta^\tau_\alpha := \Theta^{\pi_{v_i}}|_{\tau^{>i}}$ if $i = 1$.

(ii) If $\alpha = \gamma \circ \beta$ where β is an M&S abstraction and γ is a function on S^β, then $\Theta^\tau_\alpha := (\Theta^\tau_\beta)^\gamma$.

(iii) If $\alpha = \alpha_1 \otimes \alpha_2$ where α_1 (α_2) is an M&S abstraction of Θ over $V_1 = \{v_i, \dots, v_j\}$ ($V_2 = \{v_{j+1}, \dots, v_k\}$), then $\Theta^\tau_\alpha := \Theta^\tau_{\alpha_1} \otimes \Theta^\tau_{\alpha_2}$ if $i \neq 1$, and $\Theta^\tau_\alpha := (\Theta^\tau_{\alpha_1} \otimes \Theta^\tau_{\alpha_2}|_{\tau^{>j}})|_{\sigma^k \circ \dots \circ \sigma^{j+1}}$ if $i = 1$.

In words, we apply label reduction only if the underlying variable set V starts at the first variable v_1, and we choose the reduction pertaining to the last variable in V. This construction is correct relative to Θ^α, in the following sense:

Theorem 7 *Let Θ be the state space of a planning task with variables $\mathcal{V}$. Let $V \subseteq \mathcal{V}$, and let $v_1, \dots, v_n$ be an ordering of V. Let $\sigma^n \circ \dots \circ \sigma^1$ be a chained label reduction for Θ that is conservative for $v_1, \dots, v_n$, and let α be an M&S abstraction over V that allows $v_1, \dots, v_n$. Then $\Theta^\tau_\alpha = \Theta^\alpha|_{\tau^{>n}}$.*

Note here that "the correctness of Θ_α relative to Θ^α" is now interpreted in a different way. The transition system Θ^τ_α we maintain is no longer equal to the abstract state space Θ^α, but instead to the label-reduced version $\Theta^\alpha|_{\tau^{>n}}$ of that abstract state space. Since, obviously, the labels are irrelevant for h^α, the heuristic computed is the same.

The proof of Theorem 7 is a bit technical, but essentially simple. We prove by induction over the construction of α that, for any intermediate abstraction β over $\{v_i, \dots, v_j\}$, $\Theta^\tau_\beta = \Theta^\beta$ if $i \neq 1$, and $\Theta^\tau_\beta = \Theta^\beta|_{\tau^{>j}}$ if $i = 1$. The key step is induction over rule (iii) when $i = 1$. Since $\tau^{>j}$ is conservative for each of $\Theta^{\pi_{v_{j+1}}}, \dots, \Theta^{\pi_{v_k}}$, we can easily conclude that $\tau^{>j}$ is conservative for $\Theta^\tau_{\alpha_2}$. The claim then follows by applying Lemma 5 and Theorem 2.

Summing up, when during M&S we face an abstraction over variables $v_1, \dots, v_j$, we can project the operators labeling the transitions onto remaining variables $v_{j+1}, \dots, v_n$. We then still obtain the correct (label-reduced) abstract state space. In the initial implementation of Helmert et al. (2007), the motivation for doing so was the reduced *number* of labels – planning tasks often contain many operators, so this was time- and space-critical. Here, we observe that label reduction favorably interacts with bisimulation.

Bisimulation *and* Label Reduction

Label reduction obviously preserves Proposition 3:

Proposition 8 *Let Θ be a transition system, τ be a label reduction, and α a bisimulation for $\Theta|_\tau$. Then h^α is perfect.*

Proposition 8 is important because, as shown by our running example, label reduction may make a big difference:

Proposition 9 *There exist families $\mathcal{F}$ of transition systems Θ with associated label reductions τ so that the coarsest bisimulation for $\Theta|_\tau$ is exponentially smaller than the coarsest bisimulation for Θ.*

Corollary 4 tells us that, if the shrinking strategy sticks to bisimulation of the original (non label-reduced) abstract state spaces, then the final outcome will be a bisimulation. Does a similar result hold if we stick to bisimulation of the label-reduced abstract state spaces? Unsurprisingly, the answer is "yes". Given variables $v_1, \dots, v_n$ and a chained label reduction $\sigma^n \circ \dots \circ \sigma^1$, we say that α is **constructed by label-reduced bisimulation** if it is constructed so that, in any application of rule (ii) where β is over the variables $\{v_i, \dots, v_j\}$: if $i \neq 1$ then γ is a bisimulation for Θ^β; if $i = 1$ then γ is a bisimulation for $\Theta^\beta|_{\tau^{>j}}$. By combining the proofs of Corollary 4 and Theorem 7, we get:

Theorem 10 *Let Θ be the state space of a planning task with variables $\mathcal{V}$. Let $V \subseteq \mathcal{V}$, and let $v_1, \dots, v_n$ be an ordering of V. Let $\sigma^n \circ \dots \circ \sigma^1$ be a chained label reduction for Θ that is conservative for $v_1, \dots, v_n$, and let α be an M&S abstraction constructed by label-reduced bisimulation. Then α is a bisimulation for $\Theta^{\pi_V}|_{\tau^{>n}}$.*

By Theorem 10 and Proposition 8, if α is constructed by label-reduced bisimulation, then h^α is perfect. With Proposition 6, this hold for α as returned by M&S-bop. By Theorem 7 we can maintain the suitable abstract state spaces. In particular, h^α can be extracted from M&S-bop's returned transition system Θ_α. We will see next that M&S-bop has polynomial runtime in quite a number of benchmarks.

Domain-Specific Performance Bounds

We measure the performance of M&S-bop in terms of bounds on abstraction size, i.e., the number of abstract states. Runtime is a polynomial function of abstraction size. We express the bounds as functions of domain parameters like the number of objects. Polynomial bounds on abstraction size are possible only in domains with polynomial-time optimal solution algorithms. Helmert (2006) identifies six such domains in the IPC benchmarks: Gripper, Movie, PSR, Schedule, and two variants of Promela. Helmert et al. (2007) state that suitable merging and shrinking strategies exist for each of these except PSR, but do not show how to produce such strategies automatically. We begin with an easy result:

Proposition 11 *Let $\mathcal{P} = \{\Pi_n\}$ be the family of Gripper respectively Ext-Movie planning tasks, where n is the number of objects respectively snacks. Then, for any merging strategy, abstraction size for M&S-bop in $\mathcal{P}$ is bounded by a cubic respectively linear function in n.*

Ext-Movie is an extended version of Movie, allowing to scale the number of snacks. Both results hold because the M&S-bop shrinking strategy aggregates states that agree on the relevant object counts (number of objects in a room, number of snacks already obtained). It should be noted that label reduction is really needed here – in both domains, non-label-reduced bisimulations are exponentially large. The same holds true for all domains discussed below.

We next consider *scheduling-like* domains, where each task consists of some machines used to change the features $f(o)$ of processable objects o. The relevant property is that, for $o \neq o'$, $f(o)$ and $f(o')$ are mutually independent, i.e., not affected by any common operator – processing an object may affect the status of the machines but does not have immediate consequences for any other object. It is easy to see that the STRIPS variant Schedule-Strips of IPC'00 Schedule is a scheduling-like domain. We thus have:

Proposition 12 *Let $\mathcal{P} = \{\Pi_n\}$ be the family of Schedule-Strips planning tasks, where n is the number of processable objects. There exists a merging strategy so that abstraction size for M&S-bop in $\mathcal{P}$ is bounded by a polynomial in n.*

For Promela, this investigation is difficult because M&S-bop depends directly on task syntax, and the IPC'04 Promela domains are syntactically very complicated (compiled from Promela into an expressive PDDL dialect). For a straightforward direct encoding of one of the domains (Dining Philosophers), we found that M&S-bop exhibits exponential behavior. However, a variant that we tentatively named **greedy bisimulation** gives a polynomial bound. Greedy bisimulation demands bisimulation property (2) only for transitions (s, l, s') where $sd(s') \leq sd(s)$. Under certain additional conditions on the task – which hold in Dining Philosophers – greedy bisimulation results in a perfect heuristic.

Of course, M&S-bop is not omnipotent. For example, say we extend Gripper by scaling the number of robot hands. Then, for any merging strategy, there exists a shrinking strategy so that abstraction size is polynomially bounded. However, M&S-bop does not have such a bound. Robot hands H appear as *object-variable values* ("$O = H$") in the effect of pick-up operators. So, unless all object variables were already merged, operator projection doesn't remove these distinctions, and states using different hands are not bisimilar.

Experiments

In most benchmark domains, coarsest bisimulations are still large even under operator projection. We thus designed a family of (mostly) more approximate shrinking strategies. The family is characterized by 3 parameters: (1) *overall scheme*, (2) *bisimulation variant*, (3) *label reduction on/off*. For (1), the options are DFP shrinking strategy with abstraction size limit N vs. coarsest bisimulation without size limit. The former will be indicated by "DFP" in the strategy's name, the latter will be indicated by "M&S-". For (2), the options are bisimulation vs. greedy bisimulation. The strategy name is extended with "b" respectively "g". In (3), we do vs. do not use operator projection. If we do, "op" is attached to the name. In the DFP-g options, we use greedy bisimulation only if bisimulation would break the size limit. Our merging strategy is linear, and follows Fast-Downward's "level heuristic". This orders variables "closest to the root of the causal graph" up front, so that the most influential variables are projected away earlier on.

We experiment with: the previous M&S heuristic *HHH* (Helmert *et al.* 2007); *LM-cut* (Helmert and Domshlak 2009), the currently leading heuristic for optimal planning; and *structural patterns (SP)* (Katz and Domshlak 2009), a competitive heuristic related to ours in that it is based on (implicit) abstract state spaces. Our implementation is on top of Fast-Downward, with the same A* implementation for all heuristics. We ran all IPC benchmarks supported by LM-cut and HHH. The experiments were performed on dual-CPU Opteron 2384 machines, running eight experiments simultaneously in order to fully utilize the available (eight-core) machines. Each planner instance ran on a single core with a time limit of 30 minutes and a memory limit of 2 GB.

Different shrinking strategies sometimes result in complementary strengths (better performance in different domains). Thus we experimented also with all hybrid planners running any two of these strategies, sequentially with a 15 minute limit for each. The best, in terms of total coverage, of these 2-option combinations runs M&S-gop, and DFP-gop with N=200K. This is simply called "Hybrid" in what follows.

Table 1 gives the coverage data. Consider first the "no bound on N" columns on the right hand side. Comparing M&S-b with M&S-bop, we see that operator projection improves performance significantly. Recall that the heuristic in both cases is guaranteed to be perfect, so no actual search is needed in the 192 tasks where M&S-bop succeeds. The effect of using greedy bisimulation (which in general forfeits this guarantee) is dramatic. Note in particular that the number of instances where the abstraction can be built completely – without any size bound – goes up to 802. Interestingly, the effect of the parameter changes is reversed when using DFP: there, operator projection has a much larger impact on performance than greedy bisimulation.

DFP-bop dominates DFP-b (solves at least as many tasks) in 65 of the 66 combinations of domain and N value. DFP-gop dominates HHH in the total, and in 50 of these combinations; Hybrid dominates HHH in 64 combinations. SP is more competitive, but is inferior in the total to Hybrid as well as to DFP-bop and DFP-gop with $N = 10K$. LM-cut clearly dominates the total, but this is largely due to Miconic-STRIPS, where LM-cut delivers an exceptionally high-quality heuristic. Disregarding this domain, Hybrid solves 1 more task than LM-cut. In 11 of the 22 domains, Hybrid is one of the top performers, and in 4 more domains, only one task separates it from the top performer.

Coverage is a function of the trade-off between the quality of a heuristic, and the effort needed for computing it. Due to the multitude of domains and algorithms tested, it is beyond the scope of this paper to give detailed data. We provide a summary using Richter and Helmert (2009)'s **expansions score** (E) and **total-runtime score** (T). Both range between 0 and 100, for each individual instance. $E = 100$ if ≤ 100 expansions were made, $E = 0$ if $\geq 1,000,000$ expansions were made. In between, E interpolates logarithmically, so that an additive difference of 7.53 in scores corresponds to a factor 2. Similarly, $T = 100$ for runtimes ≤ 1 second, $T = 0$ for time-outs, and doubling the runtime decreases the score by about 9.25. The advantage of these scores is that they are absolute, i.e., there is no need to restrict the set of instances considered to those solved by all planners.[3]

Table 2 considers 7 pairs of heuristics X, Y. The 3 pairs in the left part illustrate the differences between the shrinking strategies proposed herein. We do not include M&S-b and M&S-bop because their expansions behavior is not interesting: they either have perfect expansions, or fail. Comparing $X =$ DFP-bop to $Y =$ DFP-b, we clearly see the advantage of operator projection. $E(Y) - E(X)$ is negative for 20 of 22 domains, meaning that X has fewer expansions. The picture is not quite as clear for runtime because DFP-bop sometimes generates more overhead (taking more time to reach the size limit N). Comparing $X =$ DFP-bop to $Y =$ DFP-gop, we clearly see that the performance difference is very small. By contrast, $X =$ DFP-bop vs. $Y =$ M&S-gop exhibits a huge variance, showing their complementary strengths. This reflects the fact that these two strategies are quite different. M&S-gop enforces bisimulation only on "$sd(s') \leq sd(s)$" transitions, but on all of those; DFP-variants enforce bisimulation everywhere but drop it completely if N is exceeded. The former has the edge in heuristic quality ($E(Y) - E(X)$ is negative for 18 domains), the latter is better in total runtime since building the abstraction generates less overhead (e.g., consider the pipesworld-notankage domain).

The middle and right parts of Table 2 show the competition with HHH respectively LM-cut. DFP-gop clearly beats HHH in expansions, and it largely keeps this advantage in runtime. M&S-gop most often produces worse heuristics than HHH, but with less overhead and thus better runtime. As for LM-cut, without any doubt this remains the most informative heuristic here – only in a few domains does DFP-

[3]Experimenting with such summaries, we found that they often misrepresented the results. For example, on instances solved by both, M&S-gop beats LM-cut even in domains where LM-cut actually scales better – but only because M&S-gop's cheap heuristic solves the small tasks very quickly, timing out on larger ones.

Domain	Hybrid	LM-cut	SP	N=10K				N=100K				N=200K				No bound on N		
				DFP-b	DFP-bop	DFP-gop	HHH	DFP-b	DFP-bop	DFP-gop	HHH	DFP-b	DFP-bop	DFP-gop	HHH	M&S-b	M&S-bop	M&S-gop
airport	22	**28**	21	23	23	23	19	15	15	15	13	11	11	11	12	1	1	22
blocks	21	**28**	21	21	21	21	18	18	18	18	19	18	18	18	19	6	6	21
depots	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**	3	5	6	6	3	1	1	**7**
driverlog	13	13	13	12	13	13	12	12	12	13	13	12	12	13	**14**	4	5	12
freecell	**16**	15	**16**	15	**16**	**16**	15	4	6	6	9	4	3	3	6	3	3	**16**
grid	**3**	2	1	1	2	2	2	1	**3**	**3**	2	0	**3**	**3**	0	0	0	2
gripper	**20**	7	7	7	11	11	7	7	**20**	**20**	7	7	**20**	**20**	7	6	**20**	7
logistics00	20	20	**22**	20	20	20	16	20	20	20	21	20	20	20	**22**	10	10	16
logistics98	5	**6**	**6**	4	4	4	5	5	5	5	5	5	5	5	5	2	1	4
miconic	67	**141**	53	55	56	56	55	55	65	65	55	55	67	67	56	40	56	50
mprime	**23**	22	**23**	18	19	19	21	8	12	12	14	4	9	9	9	1	1	**23**
mystery	15	**16**	15	13	13	13	14	7	8	8	11	6	6	6	7	2	3	15
openstacks	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
pathways	4	**5**	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
pipesworld-notank	15	**17**	14	15	**17**	**17**	9	9	9	9	2	6	6	6	0	2	2	15
pipesworld-tank	**16**	11	10	13	14	14	13	7	8	8	7	4	7	7	5	2	2	**16**
psr	**50**	49	49	49	49	49	**50**	49	49	49	50	49	49	**50**	**50**	43	45	**50**
rovers	**8**	7	6	6	7	7	7	7	**8**	**8**	7	6	**8**	**8**	7	4	4	6
satellite	**7**	**7**	6	6	6	6	6	6	**7**	**7**	6	6	**7**	**7**	6	4	6	6
tpp	**7**	6	6	6	6	6	6	6	**7**	**7**	6	6	**7**	**7**	6	5	5	6
trucks	7	**10**	7	6	7	7	6	5	7	7	6	5	7	7	6	4	4	6
zenotravel	11	**13**	11	9	11	11	11	9	12	12	11	8	11	11	11	5	6	9
Total	364	**437**	325	317	333	333	310	268	309	310	278	248	293	295	262	156	192	320
w/o miconic	**297**	296	272	262	277	277	255	213	244	245	223	193	226	228	206	116	136	270
Total M&S built				621	736	735	699	368	518	518	493	321	461	458	441	156	192	**802**

Table 1: Comparison of solved tasks over 22 IPC benchmark domains. Best results are highlighted in bold. "Total M&S built": total number of tasks for which computing the M&S abstraction did not exceed the available time/memory.

X = DFP-bop									X = HHH						X = LM-cut					
Y = DFP-b			Y = DFP-gop			Y = M&S-gop			Y = DFP-gop			Y = M&S-gop			Y = DFP-gop			Y = M&S-gop		
D	E	T	D	E	T	D	E	T	D	E	T	D	E	T	D	E	T	D	E	T
blocks	3.7	-1.7	driverlog	1.9	1.2	mystery	9.2	33.6	gripper	40.6	25.7	pipesnota	10.2	48.2	gripper	41.9	30.2	freecell	0.5	33.6
mystery	0.4	5.2	mystery	1.2	-0.2	blocks	7.9	5.5	satellite	23.7	8.8	blocks	8.6	14.7	openstack	21.9	9.7	gripper	-0.7	5.3
log98	-0.6	0.1	psr	0.8	0.2	depots	1.6	10.4	pipesnota	21.7	28.1	mystery	3.5	27.0	pipesnota	8.1	14.6	psr	-1.1	4.5
openstack	-0.8	-0.4	rovers	0.5	0.7	mprime	1.1	38.2	pathways	13.7	3.9	airport	2.2	21.5	freecell	4.1	5.9	pipesnota	-2.1	36.4
log00	-0.9	-0.9	freecell	0.5	0.4	psr	-2.0	3.0	pipestank	9.3	21.5	depots	1.6	14.1	log00	3.6	2.7	openstack	-6.6	11.5
psr	-1.0	0.9	log98	0.4	0.7	freecell	-3.1	28.1	airport	8.4	0.0	freecell	-0.5	27.0	satellite	2.4	-5.8	blocks	-14.0	-10.1
airport	-1.0	-0.7	satellite	0.3	-0.5	airport	-6.2	22.2	grid	7.3	10.3	pipestank	-0.7	43.2	psr	1.6	1.7	tpp	-15.3	-4.5
rovers	-1.3	-5.4	pipestank	0.3	-0.8	log98	-8.4	-5.6	miconic	6.1	2.0	mprime	-1.0	27.7	tpp	-2.2	-9.4	pipestank	-17.0	3.7
depots	-1.6	0.1	pipesnota	0.2	-0.1	driverlog	-9.0	4.2	zenotravel	4.9	3.0	gripper	-2.0	0.8	pipestank	-5.6	-16.3	depots	-19.8	14.9
trucks	-4.1	-4.9	tpp	0.1	0.5	rovers	-9.5	-7.3	trucks	3.4	4.5	grid	-3.8	5.8	driverlog	-9.3	-1.8	driverlog	-20.3	1.2
freecell	-4.2	3.7	trucks	0.1	-0.1	pipestank	-10.0	21.7	freecell	3.1	-0.6	trucks	-6.8	0.1	grid	-9.5	9.9	grid	-20.6	5.4
tpp	-4.4	-1.9	log00	0.1	-0.2	trucks	-10.1	-4.5	rovers	1.1	1.9	zenotravel	-6.9	-6.4	rovers	-13.5	-7.7	mystery	-21.5	2.5
zenotravel	-6.8	-13.9	zenotravel	0.1	-0.5	grid	-11.1	-4.0	blocks	0.2	8.7	miconic	-7.2	-3.1	zenotravel	-15.6	-1.8	mprime	-23.1	9.0
driverlog	-6.9	-2.4	airport	0.0	0.7	pipesnota	-11.2	19.2	openstack	0.1	1.4	rovers	-8.9	-6.2	pathways	-15.8	-16.1	rovers	-23.5	-15.8
miconic	-7.7	-2.2	grid	0.0	0.6	zenotravel	-11.6	-9.9	depots	-0.3	4.1	pathways	-12.4	8.8	depots	-21.7	4.8	log00	-24.9	-9.5
pipestank	-7.9	-2.3	miconic	0.0	0.1	tpp	-12.9	-3.5	tpp	-0.4	-0.1	psr	-13.0	0.6	blocks	-22.4	-16.1	zenotravel	-27.4	-11.2
pipesnota	-9.4	2.9	gripper	0.0	0.0	miconic	-13.4	-5.0	mprime	-2.1	-10.6	log98	-13.3	-8.8	mprime	-24.2	-29.3	satellite	-38.9	-16.5
mprime	-10.4	0.2	mprime	0.0	-0.1	log00	-21.1	-12.5	driverlog	-4.4	-3.6	tpp	-13.4	-4.2	mystery	-29.5	-31.3	airport	-39.2	-15.3
pathways	-11.0	-1.7	openstack	0.0	-0.1	pathways	-26.1	4.7	mystery	-4.5	-6.9	driverlog	-15.4	-0.6	trucks	-31.9	-20.6	pathways	-41.9	-11.2
grid	-11.2	-23.8	pathways	0.0	-0.1	openstack	-28.5	1.8	log98	-4.9	-3.4	satellite	-17.5	-1.9	airport	-33.0	-36.8	trucks	-42.1	-25.0
satellite	-28.1	-10.5	depots	-0.3	0.4	satellite	-41.0	-11.2	log00	-6.8	4.2	log00	-28.1	-8.1	log98	-47.5	-24.4	log98	-55.9	-29.8
gripper	-37.6	-25.7	blocks	-0.5	-0.5	gripper	-42.6	-24.9	psr	-10.3	2.2	openstack	-28.5	2.1	miconic	-66.5	-56.8	miconic	-80.0	-61.9
Total	-6.9	-3.6	Total	0.3	0.1	Total	-11.7	4.8	Total	5.0	4.6	Total	-7.0	9.3	Total	-12.4	-8.4	Total	-24.3	-3.8

Table 2: Difference $Y - X$ between per-domain averaged expansions (E) and total-runtime (T) scores (as per Richter and Helmert, see text), for selected pairs of heuristics X, Y. Columns ordered by decreasing $E(Y) - E(X)$. $N = 10K$ throughout.

gop manage to beat it. Everywhere else, the question is whether the faster heuristic calls for M&S (amortizing the cost for creating the abstraction in the first place) make up for the larger search space. For DFP-gop, this is the case only in depots. For M&S-gop, this happens in 11 of the domains, and partly to a considerable extent (e.g. freecell, blocks, depots, mprime).

Future Work

Our immediate future work concerns greedy bisimulation. First results suggest that there are many other interesting bisimulation variants along these lines, and corresponding conditions under which they yield perfect heuristics. The guiding idea is to test conditions on the degree and form of interactions between the current variable and the remaining ones, using the outcome to fine-tune the subset of transitions for which bisimulation property (2) is demanded.

Acknowledgements

Part of this work was performed while Raz Nissim was visiting INRIA, Nancy, France. The work was partially conducted as part of the BARQ project, supported by the French Agence Nationale de la Recherche (ANR), under Decision ANR-10-CEXC-003-01. Raz Nissim was supported in part by the Lynn and William Frankel Center for Computer Science.

References

Klaus Dräger, Bernd Finkbeiner, and Andreas Podelski. Directed model checking with distance-preserving abstractions. In *Proc. SPIN 2006*, pages 19–34, 2006.

Stefan Edelkamp. Planning with pattern databases. In *Proc. ECP 2001*, pages 13–24, 2001.

Malte Helmert and Carmel Domshlak. Landmarks, critical paths and abstractions: What's the difference anyway? In *Proc. ICAPS 2009*, pages 162–169, 2009.

Malte Helmert and Robert Mattmüller. Accuracy of admissible heuristic functions in selected planning domains. In *Proc. AAAI 2008*, pages 938–943, 2008.

Malte Helmert, Patrik Haslum, and Jörg Hoffmann. Flexible abstraction heuristics for optimal sequential planning. In *Proc. ICAPS 2007*, pages 176–183, 2007.

Malte Helmert. *Solving Planning Tasks in Theory and Practice*. PhD thesis, University of Freiburg, 2006.

Michael Katz and Carmel Domshlak. Structural-pattern databases. In *Proc. ICAPS 2009*, pages 186–193, 2009.

Robin Milner. Operational and algebraic semantics of concurrent processes. In *Handbook of TCS*, pages 1201–1242. 1990.

Silvia Richter and Malte Helmert. Preferred operators and deferred evaluation in satisficing planning. In *Proc. ICAPS 2009*, pages 273–280, 2009.

Incremental Lower Bounds for Additive Cost Planning Problems

Patrik Haslum and **John Slaney** and **Sylvie Thiébaux**
Australian National University & NICTA
firstname.lastname@anu.edu.au

Abstract

We define an incremental lower bound function for additive cost planning, based on repeatedly solving and strengthening the delete-relaxation of the problem. We show that it is monotonically increasing, and thus in the limit will produce an optimal plan. In most domains, however, it is not effective compared to alternatives such as limited search.

Introduction

Many potential applications of planning require planners to produce plans of high quality, according to a metric like cost, makespan, net benefit, or other. Even when generating guaranteed optimal plans is not computationally feasible, there is a need to be able to measure, in absolute terms, how good the plans found by non-optimal planners are. Current planning technology does not offer many tools suitable for this purpose. A lower bound function (i.e., an admissible heuristic) gives an absolute assessment, since the optimal cost is known to lie between the lower bound and the cost of the best known plan. But if this gap is too large to give confidence in the quality of the solution, and further search fails to turn up a better plan, there is not much that can be done. What is needed in this situation is an incremental lower bound: a function that can produce increasingly higher admissible estimates, given more time and memory.

Given any admissible heuristic h (including the "blind" heuristic $h = 0$) such an incremental lower bound can be obtained by running A*, IDA*, or any search algorithm that approaches the optimum from below, with h for a limited time, taking the highest proven f-value. But this is not always effective. As an example in point, columns (b) and (c) in table 1 shows, for a set of problems, the value of the LM-Cut heuristic (Helmert and Domshlak 2009) in the initial state, and the highest lower bound proven by running A* with this heuristic (Fast Downward implementation) until it exhausts memory. In only one problem does this result in a bound higher than the initial, and in no problem does the search uncover a plan.

In this paper, we propose a different approach to incremental lower bound computation, by repeatedly finding optimal solutions to relaxations of the planning problem, which are increasingly less and less relaxed. If we find a relaxed plan that is also a plan for the original, unrelaxed problem, we know it is optimal. If not, we use clues from the failure of the relaxed plan to construct the next relaxation, in such a way that the same relaxed plan will not be found again. The relaxation we consider is the standard delete-relaxation, in which negative effects of actions are ignored. (This means that we solve the NP-hard problem of finding an optimal delete-relaxed plan in each iteration.) The delete-relaxation has the property that a plan achieving atoms p and q also achieves the conjunction $\{p, q\}$. The way in which we make it less relaxed is by constructing a modified problem, in which certain conjunctions are made explicit, in the form of new atoms.[1] This carries information about negative interactions between subgoals (achievement of conjuncts) into

	(a)	(b)	(c)	(d)	(e)	(f)
chunk-089	137	104			104	108
chunk-091	12	10		12⋆	10	12⋆
window-331	42	31			31	32⋆
window-332	52	40			40	42
window-333	68	47			47	50
window-334	68	47			47	50
window-335	63	45			45	48
window-336	42	35			35	37
window-337	32	27			27	29
window-338	25	22	23		22	24
window-339	25	22			22	23
window-340	25	21			21	22
window-341	22	20			20	21
window-409	10	9		10⋆	9	10⋆

Table 1: Comparison of lower bounds on problems from a PDDL encoding of DES diagnosis. (All systems run with a memory limit of 3Gb, and no time limit.)
(a) Cost of best solution found by a non-optimal planner.
(b) LM-Cut-value of the initial state.
(c) Highest lower bound (f-value) proven by A* search with LM-Cut (where higher than (b)).
(d) Optimal solution costs found by Gamer (Edelkamp and Kissmann 2008).
(e) h^+-value of the initial state.
(f) Highest lower bound proven by h^{++} A star indicates that the bound is optimal, i.e., that the final relaxed plan is valid.

[1] It is basically the $P^m_\star$ construction of Haslum (2009), applied to a select set of conjunctions instead of all of size m.

the delete-relaxation of the modified problem. We choose the conjunctions to encode by analysing the failure of the current relaxed plan, in a way that guarantees we find a different relaxed plan in the next iteration. This incremental lower bound function is tentatively named h^{++}.

Columns (e) and (f) in table 1 show the h^{+} and h^{++} values computed, respectively. As can be seen, it achieves better bounds than A*/LM-Cut, and in a few cases even finds an optimal solution. It must, however, be noted that this is an exception rather than a rule. We have found few other domains where h^{++} is as effective, compared to the A*/LM-Cut combination. (Two examples will be presented later.)

Related Work

The idea of incrementally refining relaxations is by no means new. It is widely used in optimisation algorithms, for example in the form of incremental generation of valid cuts in integer-linear programming (e.g. Cornuéjols 2008). An instance of the idea that is closer to planning is *counterexample-guided abstraction refinement*, in which the relaxation is an abstraction of the problem. An optimal solution (plan) for the abstract problem is checked for failures wrt to the parts of the original problem ignored by the current abstraction, and those parts that cause it to fail are candidates for inclusion in the next abstraction. This idea has been applied in verification (model checking), and to planning in adversarial, probabilistic domains (Chatterjee et al. 2005), but, as far as we are aware, not to deriving lower bounds for planning with additive cost. The most closely related instance of the idea is perhaps in the work of van den Briel et al. (2007), who formulate a relaxation of planning problems as an integer programming model of a flow problem. (The problem is further simplified by solving the LP relaxation of the IP.) It is a relaxation because certain ordering constraints, due to non-deleted action preconditions, are ignored. They use composition of state variables to refine the relaxation, though not in an incremental fashion.[2]

Background

We adopt the standard definition of a propositional STRIPS planning problem, without negation in action preconditions or the goal (see, e.g., Ghallab, Nau, and Traverso 2004, chapter 2). We assume that action effects are consistent, meaning that for each action a, $\mathrm{del}(a) \cap \mathrm{add}(a) = \emptyset$.[3] As usual, a sequence of actions (or *plan*) *achieves* condition c from state s iff the sequence is executable in s and leads to a state where c holds. We assume an additive cost objective, i.e., each action a has a non-negative cost, $\mathrm{cost}(a)$, and the cost of a plan is the sum of the cost of actions in it. The initial state is denoted by s_I and the goal by G.

[2]An incremental refinement method has recently been developed for this relaxation, and used to create a quite effective cost-optimal planner. (Menkes van den Briel, pers. comm.)

[3]The semantics of PDDL permit actions that both delete and add the same atom. The net effect of such an action is to make the atom true. The transient delete effect implies that the action cannot be concurrent with any other action requiring or adding the atom, but since we consider only sequential (or sequentialisable) plans this is of no significance.

The Delete-Relaxation

The delete-relaxation of a planning problem P, denoted P^{+}, is a problem exactly like P except that $\mathrm{del}(a) = \emptyset$ for each a, i.e., no action makes any atom false. The delete-relaxation heuristic, $h^{+}(s, c)$, is defined as the minimum cost of any plan achieving c from s in the delete-relaxed problem.

Let A be a set of actions in P. We denote by $R^{+}(A)$ the set of all atoms that are reachable in P^{+}, starting from the initial state, using only actions in A. We say a set of actions A is a *relaxed plan* iff the goal $G \subseteq R^{+}(A)$. An actual plan for the delete-relaxed problem is of course a sequence of actions. That $G \subseteq R^{+}(A)$ means there is at least one sequencing of the the actions in A that reaches a goal state. When we need to distinguish a particular sequence of actions, that is valid a plan for P^{+}, we will call it a *sequenced relaxed plan*, or a *(relaxed) valid sequencing of A*.

We assume throughout that $G \subseteq R^{+}(A)$ for some set of actions A (which may be the set of all actions in P^{+}), i.e., that the goal is relaxed reachable. If it is not, the problem is unsolvable and we already have the highest possible lower bound ∞.

Computing Minimum-Cost Relaxed Plans

Our lower bound procedure depends on being able to compute a minimum-cost relaxed plan for a planning problem. It does not matter, for our purposes, how this is done. We would of course like to do it as efficiently as possible, but as solving P^{+} optimally is NP-complete, we cannot expect any method to be efficient in general. A possible approach is to treat P^{+} as a planning problem, like any other, and solve it using any cost-optimal planning algorithm. Betz and Helmert (2009) present polynomial-time algorithms for a few specific planning domains.

We will, however, use an algorithm based on the correspondance between relaxed plans and disjunctive action landmarks, established by Bonet and Helmert (2010). The algorithm is iterative in nature, and inspired by Slaney and Thiébaux's (2001) optimal planning algorithm for the Blocksworld domain.[4]

The Iterative Landmark-Based Algorithm

A *disjunctive action landmark* (landmark, for short) of a problem P is a set of actions such that at least one action in the set must be included in any valid plan for P. For any collection of landmarks L for P, the set of actions in any valid plan for P is a hitting set for L, i.e., contains one action from each set in L (by definition).

For any set of actions A such that $G \not\subseteq R^{+}(A)$, the complement of A (w.r.t. the whole set of actions) is a disjunctive action landmark for P^{+}. If A is a maximal (w.r.t. set inclusion) such "relaxed non-plan", the landmark is minimal

[4]Because we are only interested in optimal delete-relaxed plans, we also use a stronger form of relevance analysis to reduce the size of the problem: only actions that can achieve a relevant atom for the first time are relevant.

(w.r.t. set inclusion). This leads to the following algorithm for computing h^+:

Initialise the landmark collection L to $\emptyset$. Repeatedly (1) find a minimum-cost hitting set H for L; (2) test if $G \subseteq R^+(H)$; and if not, (3) extend H, by adding actions, to a maximal set H' such that $G \not\subseteq R^+(H')$, and add the complement of H' to the landmark collection.

The algorithm terminates when the hitting set H passes the test in step (2). This set is by definition a relaxed plan. There is no lower-cost relaxed plan, because any plan for P^+ must contain an action from every landmark in L, and H is a minimum-cost hitting set. Finally, the algorithm eventually terminates, because each landmark generated at step (3) is distinct from every landmark currently in L (every landmark in L contains at least one action in H, which the new landmark does not) and there is only a finite set of minimal landmarks for P^+. Two steps in this algorithm are frequently repeated, and therefore important to do efficiently: testing if the goal is relaxed reachable with a given set of actions, and finding a hitting set with minimum cost. Their implementations are detailed below. In spite of all tricks, however, computing an optimal relaxed plan is costly, in many problems criplingly so.

Testing Relaxed Reachability The set of atoms relaxed reachable with a given set of actions, i.e., $R^+(A)$, can be computed in linear time by a variant of Djikstra's algorithm:[5] Keep track of the number of unreached preconditions of each action, and keep a queue of newly reached atoms, initialised with the initially true atoms, dequeueing one at a time until the queue is empty. When dequeueing an atom, decrement the precondition counter for the actions that have this atom as a precondition, and when the counter reaches zero, mark atoms added by the action as reached and place any previously unmarked ones on the queue.

When generating a new landmark, we will perform a series of reachability computations, with mostly increasing sets of actions, starting from H, i.e., $H \cup \{a_1\}$, $H \cup \{a_1, a_2\}$, etc. Therefore, each reachability test (except the first) can be done incrementally. Suppose we have computed $R^+(A)$, by the algorithm above, and now wish to compute $R^+(A \cup \{a\})$. If $\mathrm{pre}(a) \not\subseteq R^+(A)$, $R^+(A \cup \{a\})$ equals $R^+(A)$, and the only thing that must be done is to initialise a's counter of unreached preconditions. If not, mark and enqueue any previously unreached atoms in $\mathrm{add}(a)$, and resume the main loop until the queue is again empty. If the goal becomes reachable, we must remove the last added action (a) from the set, and thus must restore the earlier state of reachability. This can be done by saving the state of $R^+(A)$ (including precondition counters) before computing $R^+(A \cup \{a\})$, copying it back if needed, instead of recomputing $R^+(A)$ from scratch.

Finding a Minimum-Cost Hitting Set Finding a (provably) minimum-cost hitting set over the collection of landmarks L is an NP-hard problem. We solve it using a recursive branch-and-bound algorithm with caching. Given a set $L = \{l_1, \ldots, l_m\}$ of landmarks to hit, pick a landmark $l_i \in L$: the minimum cost of a hitting set for L is

$$H^\star(L) = \min_{a \in l_i} H^\star(L - \{l \mid a \in l\}) + \mathrm{cost}(a)$$

A lower bound on $H^\star(L)$ can be obtained by selecting any subset $L' \subset L$ such that $l \cap l' = \emptyset$ for any $l, l' \in L'$, i.e., a set of pair-wise disjoint landmarks, and summing the costs of their cheapest actions, i.e., $\sum_{l \in L'} \min_{a \in l} \mathrm{cost}(a)$. Finding the set L' that yields the maximum lower bound amounts to solving a weighted independent set problem, but there are reasonably good and fast approximation algorithms (e.g. Halldórsson 2000).

Because the branch-and-bound algorithm is invoked recursively on subsets of L, it may prove increased lower bounds on the cost of hitting these subsets. Improved bounds are cached, and used in place of the lower bound calculation whenever a subset is encountered again. In the course of computing h^+, we will be solving a series of hitting set problems, over a strictly increasing collection of landmarks. Cached lower bounds may be used not only if a subset of L is encountered again within the same search, but also if it is encountered while solving a subsequent hitting set problem. This makes the caching mechanism quite important.

When finding a hitting set for $L \cup \{l_{i+1}\}$, we have an optimal hitting set for L. $H^\star(L)$ is clearly a lower bound on $H^\star(L \cup \{l_{i+1}\})$, and an initial upper bound can be found by taking $H^\star(L) + \min_{a \in l_{i+1}} \mathrm{cost}(a)$. These bounds are often very tight, which limits the amount of search. In particular, if l_{i+1} contains any zero-cost action, the initial upper bound is matched by the lower bound, and thus already optimal.

Finally, as long as the hitting set is not a relaxed plan, we do not require it to be optimal, since any hitting set will do as a starting point for generating another landmark. Thus, we can use any approximation algorithm to find a non-optimal hitting set, and invoke the optimal branch-and-bound procedure only when a relaxed plan is found. This scheme not always more efficient, however, because when the branch-and-bound algorithm is called, it is typically with a larger gap between the upper and lower bounds, and with fewer sets cached, resulting in more search.

The Relaxed Plan Dependency Graph

Although we have defined a relaxed plan as an unordered set of actions, there are some necessary ordering relations between actions in this set. These, and the reasons for them, are captured by the *relaxed plan dependency graph* defined below. This graph plays a central role in the identification of conflicts in a failed relaxed plan.

Let A be relaxed plan, i.e., a set of actions such that $G \subseteq R^+(A)$. We say that A is *non-redundant* if no strict subset of A is a relaxed plan. A relaxed plan can be checked for redundancy by simply testing for each action a in turn whether $G \subseteq R^+(A - \{a\})$, and made non-redundant (though not necessarily in a unique way) by greedily removing actions found to be redundant. The landmark-based algorithm can produce optimal plans containing redundant actions, although naturally only when those actions have a cost of zero. From now on, we will assume all relaxed plans are non-redundant.

[5]To the best of our knowledge, this method of computing relaxed reachability was first implemented in Fast Downward, but has not been described in any publication.

Definition 1 *Let A be a non-redundant relaxed plan. Construct a graph G' with nodes $\{n_a \mid a \in A\} \cup \{n_G\}$, i.e., the nodes of G' correspond to actions in A plus an additional node which represents the goal. With some abuse of notation, we write $\mathrm{pre}(n)$ for the precondition of node n, which is $\mathrm{pre}(a)$ for a node n_a and G for node n_G. G' has an edge from n_a to n' iff $\mathrm{pre}(n') \not\subseteq R^+(A - \{a\})$. This edge is labelled with $\mathrm{pre}(n') - R^+(A - \{a\})$.*

The relaxed plan dependency graph, $\mathrm{RPDG}(A)$, is the transitive reduction[6] of G'.

As will be shown shortly, the graph G' is acyclic, and therefore its transitive reduction is unique. (It is also computable in polynomial time; cf. Aho, Garey, and Ullman 1972.) Thus, the RPDG is well-defined.

Intuitively, an edge from node n to node n' in the RPDG means that the action associated with n is necessary for $\mathrm{pre}(n')$ to be relaxed reachable, and the edge label documents why that is, i.e., which atoms in $\mathrm{pre}(n')$ are added by that action only. However, the RPDG does not capture disjunctive dependencies: if several actions in A add atom p, there will be no edge with p in its label, and the fact that at least one of those actions is necessary to reach p will not be visible in the graph.

If there is a path from node n to node n' in $\mathrm{RPDG}(A)$, we say that the nodes are *ordered*. Conversely, if there is no path from n to n', nor from n' to n, we say they are *unordered*. This terminology is justified by properties (2) and (3) in the theorem below.

Theorem 2
(1) The graph G' in definition 1 is acyclic, and hence so is $\mathrm{RPDG}(A)$.
(2) If there is a path from n_a to n_b in $\mathrm{RPDG}(A)$, a appears before b in every valid sequencing of A.
(3) Let n_a and and n_b be two unordered nodes, i.e., such that neither n_a is reachable from n_b nor n_b from n_a in $\mathrm{RPDG}(A)$. Then there exists a valid sequencing of A in which a appears before b.
(4) If atom p appears in the label of an outgoing edge from node n_a in $\mathrm{RPDG}(A)$, then $p \in \mathrm{add}(a)$.
(5) For any two action nodes n_a and n_b in $\mathrm{RPDG}(A)$, the labels of any pair of outgoing edges from a and b, respectively, are disjoint.
(6) Any two unordered nodes n and n' in $\mathrm{RPDG}(A)$ have a common descendant, n''.

Proof: (1) follows directly from property (2) and the fact that A is non-redundant.

(2) Consider first the case where the path is a single edge from n_a to n_b, and suppose that there is a valid sequencing with b before a. This means that the preconditions of b are reachable using only those actions that appear before b in the sequence, which do not include a. This contradicts the fact that $\mathrm{pre}(b)$ is not contained in $R^+(A - \{a\})$. The general case follows by induction.

(3) Construct the sequence as follows: Let A' be the set of all actions except a, b and their descendants. Begin the sequence by applying all actions in A' (in any valid order). After this, apply a, then b, then the remaining actions in any valid order. Suppose that this fails, because $\mathrm{pre}(a)$ does not hold after applying all actions in A'. Let $p \in \mathrm{pre}(a)$ be an unsatisfied precondition. Since p does not hold initially (if it did, it would still hold after applying the actions in A'), there must be at least one action in A with $p \in \mathrm{add}(a)$ which is not a descendant of a, and no such action is in A'. Thus, it must be b or a descendant of b. But by definition this implies a path from b to this action, and thus from b to a.

(4) That p belongs to the label of an edge from n_a to some other node n' means that $p \in \mathrm{pre}(n')$ and becomes unreachable without action a. This cannot be because p is added by some other action, b, and $\mathrm{pre}(b)$ becomes unreachable without a, because if so, there would be edges from n_a to n_b and from n_b to n, and thus the edge from n_a to n would not be in the transitive reduction. Hence, p must be belong to $\mathrm{add}(a)$.

(5) Suppose there are two nodes, n_a and n_b, in $\mathrm{RPDG}(A)$, that both have outgoing edges with labels that include p (both must be action nodes, since the goal node has no outgoing edges). By property (4), p must belong to both $\mathrm{add}(a)$ and $\mathrm{add}(b)$. If there is a path from n_a to n_b, then a appears before b in any sequencing of A (by property (2)), and removing b cannot make p unreachable. If n_a and n_b are unordered, either a or b can appear first in a valid sequencing (by property (3)). Thus, removing just one of a or b cannot make p unreachable.

(6) Observe first that from every action node n_a there is a path to the goal node n_G. Since the graph is acyclic, every path must end in a leaf (node with no outgoing edges). A leaf node that is not the goal node is a redundant action, since removing this action does not make the goal unreachable. Thus, any pair of unordered nodes have a common descendant, the goal node if no other. □

Note that property (3) holds only for pairs of nodes. The reason is that the RPDG does not capture disjunctive precedences. Suppose, for example, that actions a and b both add atom p, and that p is a precondition of action c. (a and b both also add some other relevant atoms, as otherwise at least one of them would be redundant.) There are valid sequencings with c before a (if c is preceded by b) and c before b (if preceded by a), but not with c before both a and b. Because of this, every valid sequencing of A is a topological sort of $\mathrm{RPDG}(A)$, but not every topological sort of $\mathrm{RPDG}(A)$ is a valid sequencing of A.

Incrementally Strengthening the Relaxation

The idea behind the incremental lower bound function h^{++} is as follows: Given a minimal-cost sequenced relaxed plan A, if it is valid also for the non-relaxed problem P, then the optimal plan cost for P is equal to the cost of A (and we have a plan to prove it). If not, then we can identify a set of "conflicts", C, which are conjunctive conditions that are required but fail to hold at some point in the execution of the plan. We construct a new problem, called P^C, in which these conjunctions are explicitly represented by new atoms. The new problem has the property that A is not a relaxed plan for it (or, to be precise, no sequence of representatives

[6]The transitive reduction of a graph is the smallest edge-subgraph that has the same transitive closure.

of each action in A is a relaxed plan). The minimum cost of a relaxed plan for the new problem is a lower bound also on the cost of solving P. There is no guarantee that it will be higher than the h^+ value for the original problem, but since the set of non-redundant minimal-cost sequenced relaxed plans is finite, repeated application of this procedure will eventually result in a higher lower bound. In the limit it will even result in an optimal plan, unless time or memory limits halt the process.

The P^C Construction

Let $C = \{c_1, \ldots, c_n\}$ be a set of (non-unit) conjunctions over the atoms in P. We construct a problem P^C, in which these distinguished conjunctions are explicitly represented by new atoms, called "meta-atoms". The set of actions in P^C is also modified, so that the truth of the meta-atoms will accurately reflect the truth of the conjunctions they represent. This is essentially the P^m_* construction (Haslum 2009), applied to a specific set of conjunctive conditions.

Definition 3 *The set of atoms of P^C contains all atoms of P, and for each $c \in C$ an atom π_c. π_c is initially true iff c holds in the initial state of P, and is a goal iff $c \subseteq G$ in P. For each action a in P and for each subset C' of C such that for each $c \in C'$ (i) $\mathrm{del}(a) \cap c = \emptyset$ and $\mathrm{add}(a) \cap c \neq \emptyset$; and (ii) any $c' \in C$ such that $c' \subset c$ and c' satisfies (i) is also in C', P^C has an action $\alpha_{a,C'}$ with*

$$\begin{aligned}
&\mathrm{pre}(\alpha_{a,C'}) = \mathrm{pre}(a) \cup \textstyle\bigcup_{c \in C'} (c - \mathrm{add}(a)) \cup \\
&\qquad \{\pi_c \mid c \subseteq (\mathrm{pre}(a) \cup \textstyle\bigcup_{c \in C'} (c - \mathrm{add}(a))), c \in C\} \\
&\mathrm{add}(\alpha_{a,C'}) = \mathrm{add}(a) \cup \{\pi_c \mid c \in C'\} \cup \\
&\qquad \{\pi_c \mid c \subseteq \mathrm{add}(a), c \in C\} \\
&\mathrm{del}(\alpha_{a,C'}) = \mathrm{del}(a) \cup \{\pi_c \mid c \cap \mathrm{del}(a) \neq \emptyset, c \in C\}
\end{aligned}$$

and $\mathrm{cost}(\alpha_{a,C'}) = \mathrm{cost}(a)$.

Each action $\alpha_{a,C'}$ in P^C is constructed from an action a in P. We call this the *original action* for $\alpha_{a,C'}$. Conversely, for each action a in P we call the actions in P^C whose original action is a the *representatives of* a. Note that P^C always has at least one representative of each action a, namely $\alpha_{a,\emptyset}$.

Intuitively, meta-action $\alpha_{a,C'}$ corresponds to applying a in a state such that each conjunction $c \in C'$ will be made true by applying a, which is why $\alpha_{a,C'}$ adds π_c. This means that a does not delete any part of c, and the part of c not made true by a is already true. For any conjunction c that is made true by a alone (i.e., such that $c \subseteq \mathrm{add}(a)$), π_c is added by every representative of a.

The size of P^C is (potentially) exponential in the size of C, i.e., the number of conditions, but not in their size.

Theorem 4 *Given any plan for P^C, the corresponding sequence of original actions is a plan for P. Conversely, given any plan for P, there is a plan for P^C made up of a representative of each action in the plan for P (and hence of equal cost).*

Proof: The first claim follows directly from that the preconditions and effects of each action in P^C on atoms present in P are identical to those of the original action in P.

For the second, we choose by induction a representative of each action in the plan such that in each state resulting from the execution of the plan for P^C, π_c is true whenever c is true. It is then easy to see that each action in this plan will be executable, since the precondition of an action in P^C includes a meta-atom π_c if and only if it includes all of c, and that the goal condition will hold at the end.

The correspondence holds in the initial state by definition. Suppose the current state is s, and that a is the next action in the plan for P. Let s' be the next state in the execution of the plan in P, i.e., the state that results from applying a in s. Let C' be the subset of conditions in C that hold in s'. For each $c \in C'$, one of the following cases must hold: (1) c holds in s and $c \cap \mathrm{del}(a) = \emptyset$; (2) $c \subseteq \mathrm{add}(a)$; or (3) $c \not\subseteq \mathrm{add}(a)$, $c \cap \mathrm{add}(a) \neq \emptyset$, $c - \mathrm{add}(a)$ holds in s and $c \cap \mathrm{del}(a) = \emptyset$. Let $C'' = C' - (\{c \mid c \subseteq \mathrm{add}(a)\} \cup \{c \mid c \text{ holds in } s\})$, i.e., C'' is exactly the subset of conditions in C' that hold in s' because of case (3), and choose the representative $\alpha_{a,C''}$. There is such an action in P^C because condition (*i*) of the definition is implied by case (3), and condition (*ii*) by the choice of C'' as the set of all $c \in C$ that hold in s' by this case. If c holds in s' by case (1), π_c holds in the current state of execution in P^C by inductive assumption, and it is not deleted by the chosen representative. If c holds in s' by cases (2) or (3), it is added by the chosen representative. Finally, for each c that holds by case (3), $c - \mathrm{add}(a)$ must hold in s: thus, all atomic preconditions of the chosen representative are true in the current state, and therefore, by inductive assumption, so are any precondition meta-atoms representing conjunctions of the atomic preconditions. □

As of corollary, any lower bound on the cost of solving P^C is also a lower bound on the cost of any plan for P.

Compared to P, P^C contains no additional information. The reason why this construction is nevertheless useful is that the delete-relaxation of P^C may be more informative than the delete-relaxation of P. If we have any additional source of information about unreachability in P, such as static mutexes or invariants, this can be used in the construction of P^C to further strengthen $(P^C)^+$, by not including π_c in the add effects of any action for any condition $c \in C$ that is known to be unreachable, in P. (The correspondance shown in theorem 4 holds also with this modification, since the chosen representatives only add meta-atoms for conditions made true by the plan.) Our current implementation uses static mutexes found by h^2.

Conflict Extraction for a Sequenced Relaxed Plan

Consider a sequenced relaxed plan A. Deciding if it is a valid plan for the real problem, P, is easily done by simulating its execution. If the sequence is not a valid plan, then simulating it will at some point fail, meaning that the precondition of the next action to be applied does not hold in the current state. Call this the *failed condition*, and let n_f be the corresponding node in $\mathrm{RPDG}(A)$. (Note that the failed condition may also be the goal.) Let p be some unsatisfied atom in the failed condition. Since the sequence is valid in the relaxed sense, p was either true initially or added by some action preceding the point of failure. Thus, p was true in some earlier state also in the real execution of the plan, but deleted by some action taking place between that point and the point of failure. Call that action the *deleter*, and let n_d be its node in $\mathrm{RPDG}(A)$. We distinguish two cases:

Case 1: There is a path from n_d to n_f in $\mathrm{RPDG}(A)$. In this case, for each edge on the path from n_d to n_f choose an atom q from the edge label and form the conflict $\{q, p\}$.

Case 2: There is no path from n_d to n_f in $\mathrm{RPDG}(A)$. Clearly there cannot be a path from n_f to n_d, since n_d appeared before n_f in the sequence, which means the nodes are unordered. Let n_c be the one of their nearest common descendants that appears first in the sequence, and let L_d and L_f be atom sets constructed by choosing one atom from each edge label on the path from n_d to n_c and the path from n_f to n_c, respectively. Form the set of conflicts $\{\{q, q'\} \mid q \in L_d, q' \in L_f \cup \{p\}\}$. By property (5) of theorem 2, each conflict in the set is proper, in the sense that each in pair q and q' are distinct atoms.

Theorem 5 *Let $A = a_1, \ldots, a_n$ be a non-redundant sequenced relaxed plan for P which is not a real plan for P, and let C be a set of conflicts extracted as described above. No sequence $A' = \alpha_1, \ldots, \alpha_n$, where each α_i is a representative of a_i, is a relaxed plan for P^C.*

Proof: In the execution of A, we have the two nodes n_d and n_f, as described above. We examine the two possible cases:

Case 2 (n_d and n_f are unordered): Let n_c be the nearest common descendant that is first in the sequence. The edges of the path from n_d to n_c are labeled with atoms $q_1, \ldots, q_m$ (possibly among others), and the edges of the path from n_f to n_c are labeled with atoms $q'_1, \ldots, q'_l$. Since p is a precondition of the failed action, we can prepend it to the second path. The following picture illustrates:

$$\begin{tikzcd}
 & n_d \arrow[r, "q_1"'] & n_1 \arrow[r, "q_2"'] & \cdots \arrow[r, "q_{m-1}"'] & n_{m-1} \arrow[dr, "q_m"] & \\
 & & & & & n_c \\
{} \arrow[r, "p"'] & n_f \arrow[r, "q'_1"'] & n'_1 \arrow[r, "q'_2"'] & \cdots \arrow[r] & n'_{l-1} \arrow[ur, "q'_l"'] &
\end{tikzcd}$$

Each pair of nodes where one is taken from the first path and the other from the second (including n_f) are unordered, and each conjunction $\{q_i, q'_j\}$ is represented by a meta-atom in P^C. The final condition, $\{q_m, q'_l\}$ is contained in $\mathrm{pre}(n_c)$, and thus $\pi_{\{q_m, q'_l\}}$ belongs to the precondition of any representative of n_c (resp. to the goal, if n_c is the goal node). We show that none of these meta-atoms hold at any point prior to n_c in the relaxed execution of the sequence. This implies it is not a valid sequenced relaxed plan for P^C.

Let a_d be the deleter (the action associated with n_d). Immediately after a_d, $\pi_{\{q_1, p\}}$ is false, since it was not true before (q_1 was not true before a_d, as otherwise it could not label the outgoing edge from n_d) and no representative of a_d adds it (by definition, since $p \in \mathrm{del}(a_d)$). Suppose we have taken some steps along both paths: the next actions are a_i and a'_j, respectively. By inductive assumption, $\pi_{\{q_i, q'_j\}}$ does not hold. Every representative of a_i in P^C has q_i in its precondition. No representative of a_i adds q'_j. (Some actions on the first path might add p, but if so, they come after a_f in the sequence. This is ensured by the way the conflict was extracted: a_d lies between the last point where p was true and a_f.) Thus, any representative that adds $\pi_{\{q_{i+1}, q'_j\}}$ must be of the form $\alpha_{a_i, C'}$ with $\{q_{i+1}, q'_j\} \in C'$, and thus its precondition must include q'_j. Therefore, its precondition also includes $\pi_{\{q_i, q'_j\}}$ and no such representative is relaxed applicable. Hence, $\pi_{\{q_{i+1}, q'_j\}}$ is false after taking the next step along the first path, i.e., after executing a_i. By the same argument, after taking a step along the second path, i.e., after executing a_j, $\pi_{\{q_i, q'_{j+1}\}}$ is false. Thus, by induction, $\pi_{\{q_m, q'_l\}}$ is false after executing the last in sequence of the actions represented by n_{m-1} and n'_{l-1}.

Next, suppose that $\pi_{\{q_m, q'_l\}}$ is added by some meta-action, say α, between the last of n_{m-1} and n'_{l-1}, and n_c. α must also add at least one of q_m and q'_l. (This is true also if q_m and q'_l are meta-atoms, due to condition (*ii*) in definition 3.) Suppose it adds q_m: then n_α and n_{m-1} cannot be unordered, and neither can n_α be ordered before n_{m-1}, because if so, q_m would not label the edge from n_{m-1} to n_c. (Recall the meaning of an edge in the RPDG: removing the action associated with n_{m-1} makes the precondition q_m of n_c relaxed unreachable.) Thus, n_α follows n_{m-1}. Similarly, if α adds q'_l, n_α must follow n_{l-1}. If α does not add q'_l, it must have q'_l as a precondition. This implies that there is an edge, labeled with q'_l, from n_{l-1} to n_α, because there is no other action adding q'_l before n_c (if there were q'_l would not label the edge from n_{l-1} to n_c), and thus not before n_α. In summary, whether α adds either one of q_m and q'_l or both, n_α is a common descendant of n_{m-1} and n_{l-1}. Since α must appear before n_c in the sequence, this contradicts the choice of n_c as the first-in-sequence nearest common descendant.

Case 1 (n_d precedes n_f): This case is similar to the previous, but simpler in that we have only one path. The edge labels along this path include atoms $q_1, \ldots, q_m$, and the conflict set contains $\{q_i, p\}$ for each q_i, so P^C has a corresponding meta-atom $\pi_{\{q_i, p\}}$. The precondition of the failed node contains $\{q_m, p\}$, and thus the precondition of any representative of this action (or the goal, if the failed node is n_G) in P^C includes $\pi_{\{q_m, p\}}$. As in the previous case, none of these meta-atoms will hold in the relaxed execution of the sequence, and thus the failed action is not relaxed applicable. $\pi_{\{q_1, p\}}$ does not hold immediately after a_d. No action taking place between a_d and a_f in the sequence adds p (again, this is because a_d lies between the last point where p was true and a_f) and thus any representative of any of these actions which adds $\pi_{\{q_{i+i}, p\}}$ requires $\pi_{\{q_i, p\}}$. □

Conflict Extraction for Non-sequenced Plans

A non-sequenced relaxed plan is valid for the real problem iff there exists a sequencing of it that is a valid plan. Enumerating the sequencings of a relaxed plan is straightforward: starting with an empty sequence, non-deterministically choose the next action to apply from those whose preconditions are initially true or added by previously applied actions, never applying an action more than once. Backtracking systematically over the non-deterministic choices yields all sequencings. (When choosing the next action to apply, we divide the candidates into sets of mutually commutative actions and branch only on which set to apply. Changing the order of actions in a commutative set will not change the relaxed or real validity of a sequence. It may, however, affect which conflicts are found, as described below.)

The conflict extraction procedure above is defined for sequenced relaxed plans, and there is a spectrum ways to apply it to a non-sequenced plan: At one end, we may choose one arbitrary sequencing and add only the conflicts geneated by this sequence. We may then find for the modified problem a relaxed plan comprised of the same set of actions, but if so, it will not permit them to be sequenced in the same way. At the other end of the spectrum, we may add the union of conflicts extracted from all sequencings of the relaxed plan, ensuring that the modified problem requires a different relaxed plan. Both methods have their drawbacks: Eliminating only one sequencing at a time can result in many iterations, in each of which we must compute a minimum-cost relaxed plan. On the other hand, adding a large number of conjunctions can cause the size of the modified problem to blow out.

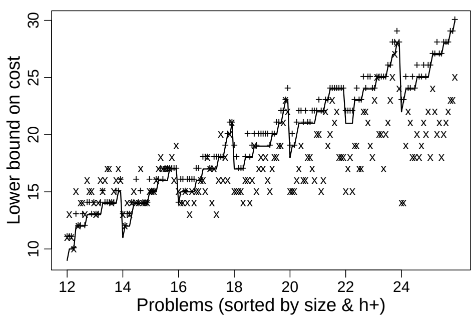
We adopt a middle ground: We enumerate all sequencings (modulo interleaving of commutative actions), and identify in each all triplets (n_d, n_f, p) of deleter, failed node and false atom $p \in \mathrm{pre}(n_f)$, then choose a set of such triplets containing at least one from every sequence, and take the union of conflict sets generated by each chosen triplet. The choice is made with the aim of minimising the size of the final conflict set. (This is again a weighted hitting set problem, which we solve with an approximation algorithm.) The number of conjunctions generated by (n_d, n_f, p) is estimated by the length of the shortest path from n_d to n_f, if they are ordered (case 1), and the smallest product of the length of their shortests paths to a nearest common descendant, if they are not (case 2). If the deleter and failed node are unordered and have several nearest common descendants, we choose only one. (The nearest common descendants are all unordered, so there exist sequencings with each one of them appearing first. This is where the order of commutative actions may affect which conflicts are generated.) By theorem 5, there is at least one choice that generates a conjunction not already represented by a meta-atom.

Iterating the Procedure

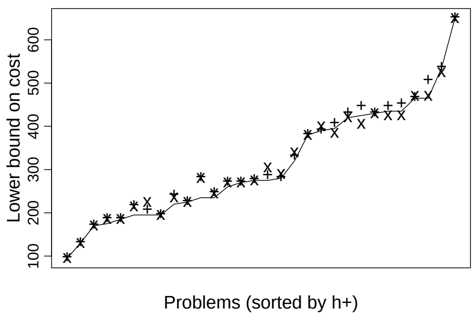
The h^{++} procedure iterates finding a relaxed plan, extracting a set of conflicts, and adding meta-atoms representing them to the problem, until the relaxed plan is also a real plan (or we run out of patience or memory). In the process, we construct a sequence of problems, P, P^{C_1}, $(P^{C_1})^{C_2}$, etc. The conflict extraction procedure described above generates only binary conjunctions, but from the second iteration on, the atoms in such a pair may themselves be meta-atoms.

In fact, what we do is slightly different: Instead of $(\cdots(P^{C_1})^{\cdots})^{C_k}$, we construct the problem $P^{(C_1 \cup \ldots \cup C_k)}$, which is not the same. That is, instead of creating meta-atoms representing conjunctions of meta-atoms, we take the union of the sets of original atoms they represent as the new conjunction. Let $\mathrm{atoms}(\pi_c) = c$, when π_c is a meta-atom, and $\mathrm{atoms}(p) = \{p\}$, when p is an original atom. When we generate a conflict $\{\pi, \pi'\}$, where π and π' may be either original or meta-atoms, the new conjunction is $\mathrm{atoms}(\pi) \cup \mathrm{atoms}(\pi')$.

There are two reasons for this: First, if we build up conjunctions by combining meta-atoms pair-wise, we can end up with several meta-atoms that in fact represent the same conjunction of original atoms, which is clearly redundant. Second, the delete-relaxation of $P^{(C_1 \cup \ldots \cup C_k)}$ is in fact a stronger relaxation of P than the delete-relaxation of $(\cdots(P^{C_1})^{\cdots})^{C_k}$. The reason for this is that in the second, "incremental", construction only meta-atoms representing conditions in the set C_i will be added to the precondition of new action representatives created at that step. (Note that in definition 3, $\mathrm{pre}(\alpha_{a,C'})$ contains π_c for *every* $c \in C$ contained in the set of original atoms in its precondition.) In practice, $P^{(C_1 \cup \ldots \cup C_k)}$ is of course constructed incrementally, not rebuilt from scratch each iteration. This can be done by keeping track of conjunctions already represented by meta-atoms, so that these can be added to the preconditions of new actions as required.



(a)



(b)

Figure 2: Comparison between h^{++} and A*/LM-Cut on problems from the (a) Blocksworld and (b) Woodworking (IPC 2008) domains. The graphs show the h^{+} value (–) and the highest lower bound proven by h^{++} (+) and A* (×), within a 1 hour CPU limit. Problems are sorted by increasing size and/or h^{+} value.

Additional Results

As noted in the introduction, we have found few planning domains in which h^{++} exceeds, or even comes close to, the performance of A* search with the LM-Cut heuristic, as measured by the highest lower bound proven within time and memory limits. Figure 2 shows results for two such domains: Blocksworld (3ops) and Woodworking (IPC 2008). We can observe an interesting trend: Small or easy problems are quickly solved by A*, but often not by h^{++}, but as problems grow larger and/or have costlier (presumably longer) plans, the relative efficiency of A*/LM-Cut drops, such that it is eventually dominated even by h^{+}, while h^{++} continues to sometime make a modest improvement over h^{+}. (Another comparison that should be made is of course with A* search using h^{+} as the heuristic.)

The properties of a problem that affect the efficiency of h^{++} are different from those that affect A*. For instance, a flat f-landscape, like that often caused by zero-cost actions, has little impact on h^{++} (an abundance of zero-cost actions is even helpful, since it makes finding an optimal hitting set easier). On the other hand, the efficiency of h^{++} is sensitive to the structure of the problem delete-relaxation: First, because the time taken to compute h^{++} is (nearly always) dominated by the h^{+} computation, which is in many problems prohibitively expensive. Second, if the delete-relaxation is inaccurate, the initial h^{+} value will be far from the optimal real plan cost, and if there are many alternative relaxed plans, or many ways to sequence them, the number of iterations of problem modification needed to boost the h^{++} value above the initial h^{+} value will be large. It is interesting to note that in the three problem domains we found where h^{++} is competitive with A*/LM-Cut, although they are not delete-free, optimal plans require most atoms to be made true only once.

Conclusions & Open Questions

Incremental lower bound functions are vital for inspiring confidence in the quality of plans produced by planners that do not offer any optimality guarantees, and thus important for acceptance of automated planning in many applications. The idea of repeatedly solving increasingly less and less relaxed problem relaxations seems an attractive approach to constructing such functions, but has not been widely explored. We have proposed one realisation of this idea, based on the common delete-relaxation. Its present incarnation, however, performs well, compared to alternatives such as bounded A* search, in only a few domains; for most problems, it is hopelessly inefficient.

There are many open questions, and options to explore, concerning the effectiveness h^{++}, and incremental lower bound functions more generally. First, in many problems not even a first optimal relaxed plan can be computed in reasonable time: is this due to the intrinsic hardness of h^{+}, or is the iterative landmark-based algorithm we use inefficient compared to other conceivable methods? Betz and Helmert (2009) present polynomial-time algorithms for computing h^{+} in a few specific domains: are there efficient h^{+} algorithms for larger classes of domains? (e.g., characterised by properties of the problems causal or domain transition graphs). Second, the size of P^{C} often grows exponentially with the number of conditions in C. A slightly different encoding of conjunctions can be done with only a linear size increase, using conditional action effects. This encoding has a weaker delete-relaxation, but still sufficient for theorem 5 to hold.[7] To make use of it, however, we need a method of computing optimal delete-relaxed plans for problems with conditional effects.

Finally, incremental lower bound functions can be devised in an analogous manner based on other relaxations, such as abstraction or the flow-based order relaxation used van den Briel et al. There are surely domains in which these will be more accurate than the delete-relaxation, just as there are a few in which the delete-relaxation is best.

References

Aho, A.; Garey, M.; and Ullman, J. 1972. The transitive reduction of a directed graph. *SIAM Journal on Computing* 1(2):131–137.

Betz, C., and Helmert, M. 2009. Planning with h^{+} in theory and practice. In *Proc. of the ICAPS'09 Workshop on Heuristics for Domain-Independent Planning*.

Bonet, B., and Helmert, M. 2010. Strengthening landmark heuristics via hitting sets. In *Proc. 19th European Conference on Artificial Intelligence (ECAI'10)*, 329–334.

Chatterjee, K.; Henzinger, T.; Jhala, R.; and Majumdar, R. 2005. Counterexample-guided planning. In *Proc. 21st International Conference on Uncertainty in Artificial Intelligence (UAI'05)*, 104–111.

Cornuéjols, G. 2008. Valid inequalities for mixed integer linear programs. *Mathematical Programming* 112(1):3–44. `http://dx.doi.org/10.1007/s10107-006-0086-0`.

Edelkamp, S., and Kissmann, P. 2008. GAMER: Bridging planning and general game playing with symbolic search. In *IPC 2008 Planner Abstracts*.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. Morgan Kaufmann Publishers. ISBN: 1-55860-856-7.

Halldórsson, M. 2000. Approximations of weighted independent set and hereditary subset problems. *Journal of Graph Algorithms and Applications* 4(1):1–16.

Haslum, P. 2009. $h^{m}(P) = h^{1}(P^{m})$: Alternative characterisations of the generalisation from $h^{\max}$ to h^{m}. In *Proc. of the 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*.

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)*.

Slaney, J., and Thiebaux, S. 2001. Blocks world revisited. *Artificial Intelligence* 125. `http://users.cecs.anu.edu.au/~jks/bw.html`.

van den Briel, M.; Benton, J.; Kambhampati, S.; and Vossen, T. 2007. An LP-based heuristic for optimal planning. In *Proc. 13th International Conference on Principles and Practice of Constraint Programming (CP'07)*, 651–665.

[7]Thanks to Joerg Hoffmann and Emil Keyder for pointing this out.

Planning with SAT, Admissible Heuristics and A*

Jussi Rintanen
The Australian National University, Canberra, Australia

Abstract

We study the relationship between optimal planning algorithms, in the form of (iterative deepening) A* with (forward) state-space search, and the reduction of the problem to SAT. Our results establish a strict dominance relation between the two approaches: any iterative deepening A* search can be efficiently simulated in the SAT framework, assuming that the heuristic has been encoded in the SAT problem, but the opposite is not possible as A* and IDA* searches sometimes take exponentially longer.

1 Introduction

We investigate the relations between two main approaches to finding optimal plans: state-space search with heuristic search algorithms such as A*, and the reduction to the propositional satisfiability problem SAT. Our optimality criterion is the minimality of the number of actions in a plan.

The theoretical limitations of the leading approaches to the optimal planning problem are well understood. A* [Hart *et al.*, 1968] is the best-known optimal search algorithm, which is guaranteed to expand the smallest possible number of states of any algorithm that does search state-by-state. The performance of A* is determined by the heuristic it uses. With a perfect heuristic it expands only the states corresponding to one optimal action sequence from the initial state to a goal state. With less informed heuristics the memory consumption is typically much higher. For example, with the trivial heuristic $h(s) = 0$ it will expand all states with a distance $< k - 1$ from the initial state when k is the length of the shortest path to a goal state. For short (polynomially long) plans A* still requires an exponential amount of memory in the worst case.

Planning by reduction to SAT, as proposed by Kautz and Selman [1992], has slightly different computational limitations. The sizes of the required formulas are linearly proportional to the length of action sequences considered. These action sequences may have an exponential length with respect to the representation of the problem instance, and, similarly to A*, will in this case require an exponential amount of memory. However, for short plans (polynomial in the size of the representation), even when A* expands an exponential number of states, the corresponding SAT problem can be solved in polynomial space (simply because SAT$\in$NP and NP$\subseteq$PSPACE.) This optimal space bound is achieved by well-known algorithms for the SAT problem, including the Davis-Putnam-Logemann-Loveland procedure [Davis *et al.*, 1962]. A counterpart of SAT in the state-space search domain is the iterative deepening A* (IDA*) algorithm [Korf, 1985] which stores in the memory, at any given time, only one bounded-length path in the state-space, and not all visited states like A* does. Under mild assumptions, IDA* does not perform many more search steps than A* [Korf, 1985, Theorem 6.4]. Although the worst-case resource requirements of SAT and IDA* are the same, the results of our work show that there is a simple implementation of SAT-based planning that is guaranteed to never do worse than state-space search with IDA*, and will sometimes perform exponentially better.

Our first result shows that if a given heuristic is encoded in the SAT problem, the Davis-Putnam-Logemann-Loveland procedure [Davis *et al.*, 1962] can simulate state-space search with IDA* [Korf, 1985], the iterative deepening variant of A* [Hart *et al.*, 1968], within the same time and space bounds. Our second result shows that the opposite does not hold: we construct a planning problem that requires an exponential amount of computation by A* and IDA*, but only a short resolution proof which is constructed by the unit propagation procedure in linear time.

The structure of the paper is as follows. In Section 2 we present the background of the work in SAT. Section 3 relates the notion of admissibility to the SAT framework and shows that one of the best known admissible heuristics is implicit in it. Section 4 presents a simulation of IDA* with the Davis-Putnam-Logemann-Loveland procedure for SAT. In Section 5 we show that both IDA* and A* are sometimes exponentially worse than SAT. We conclude by discussing related work in Section 6 and future research topics in Section 7.

2 Technical Background

Let Σ be a set of propositional variables. *Formulas* (over Σ) can be recursively constructed as follows. Every $a \in \Sigma$ is a formula. If ϕ is a formula, then so is $\neg\phi$. If ϕ_1 and ϕ_2 are formulas, then so are $\phi_1 \vee \phi_2$ and $\phi_1 \wedge \phi_2$. We also use $\top$ for the constant *true* and $\bot$ for *false*. For propositional variables $x \in \Sigma$, x and $\neg x$ are *literals*. The complement $\bar{l}$ of a literal l is defined by $\overline{x} = \neg x$ and $\overline{\neg x} = x$ when $x \in \Sigma$. A finite disjunction of literals is a *clause*.

A *valuation* of Σ is a (total) function $v : \Sigma \to \{0, 1\}$. A formula ϕ is *satisfied* by v (written $v \models \phi$) if the following holds. If $\phi = x$ for some $x \in \Sigma$, then $v \models \phi$ iff $v(x) = 1$. If $\phi = \neg\phi_1$, then $v \models \phi$ iff $v\neg \models \phi_1$. If $\phi = \phi_1 \wedge \phi_2$, then $v \models \phi$ iff $v \models \phi_1$ and $v \models \phi_1$. If $\phi = \phi_1 \vee \phi_2$, then $v \models \phi$ iff $v \models \phi_1$ or $v \models \phi_2$. The logical consequence relation $\phi \models \phi'$ holds if $v \models \phi$ implies $v \models \phi'$ for all valuations v. We use these notations also for sets of clauses. A formula ϕ (over Σ) is *satisfiable* if and only if $v \models \phi$ for some valuation v of Σ.

2.1 Planning

Planning problems are expressed in terms of a set F of facts (Boolean state variables) and a set of actions. Each action has a precondition $C \subseteq F$, consisting of the facts that have to be true for the action to be possible, and a set P of facts that become true (the *add* effects) and a set N of facts that become false (the *delete* effects). We define $\mathit{prec}(\langle C, P, N\rangle) = C$ and $\mathit{eff}(\langle C, P, N\rangle) = P \cup \{\neg f | f \in N\}$. An action is possible in state s (a valuation of F) if $s \models C$. The successor state $s' = \mathrm{succ}_a(s)$ of s with respect to $a = \langle C, P, N\rangle$ satisfies the following: $s' \models l$ for all $l \in \mathit{eff}(a)$ and $s'(f) = s(f)$ for all $f \in F\backslash(P \cup N)$.

A problem instance is $\langle F, I, A, G\rangle$, where F is a set of facts, I is a state, A is a set of actions, and G is a set of literals. The objective is to find a shortest sequence of actions such that $\mathrm{succ}_{a_n}(\mathrm{succ}_{a_{n-1}}(\mathrm{succ}_{a_2}(\mathrm{succ}_{a_1}(I)))) \models G$.

2.2 Planning in the Propositional Logic

Planning was first represented in the propositional logic by Kautz and Selman [1992]. The idea is to look at the bounded length planning problem with time points $0, \ldots, T$. A propositional formula encodes the possible plans and their executions by describing how the state can change between two consecutive time points.

In this paper, we use a simple and efficient encoding based on explanatory frame axioms. The propositional variables we use are $f@t$ for facts $f \in F$ and $t \in \{0, \ldots, T\}$, $a@t$ for actions $a \in A$ and $t \in \{0, \ldots, T-1\}$.

We translate the action $a = \langle C, P, N\rangle$ into the following propositional clauses for all $t \in \{0, \ldots, T-1\}$.

$$\neg a@t \vee f@t \text{ for all } f \in C \tag{1}$$

$$\neg a@t \vee f@(t+1) \text{ for all } f \in P \tag{2}$$

$$\neg a@t \vee \neg f@(t+1) \text{ for } f \in N \tag{3}$$

For every fact f and time $t \in \{0, \ldots, T-1\}$ we have *frame axioms* that state when facts remain unchanged.

$$f@t \vee \neg f@(t+1) \vee a_{k_1}@t \vee \cdots \vee a_{k_m}@t \tag{4}$$

$$\neg f@t \vee f@(t+1) \vee a_{n_1}@t \vee \cdots \vee a_{n_s}@t \tag{5}$$

Here $a_{k_1}, \ldots, a_{k_m}$ are all the actions that change f from false to true. and $a_{n_1}, \ldots, a_{n_s}$ are all the actions that change f from true to false.

Additionally, at most one action can take place at a time, which is expressed by the clauses

$$\neg a@t \vee \neg a'@t \text{ for all } a, a' \in A, t \in \{0, \ldots, T-1\}, \tag{6}$$

and at least one action must take place, expressed by

$$\bigvee_{a \in A} a@t \text{ for every } t \in \{0, \ldots, T-1\}. \tag{7}$$

For given sets of facts and actions and an integer T, we denote the set of all the above clauses by H_T. The action sequences can be restricted to *plans* that reach some goals from some initial state by using H_T with additional clauses.

For a given state $s : F \to \{0, 1\}$, we define a clause set representing s by $\mathit{lits}(s) = \{f \in F | s(f) = 1\} \cup \{\neg f | f \in F, s(f) = 0\}$. For a clause set S over F, we define the set $S@t$ in which every propositional variable $f \in F$ has been replaced by its time-tagged variant $f@t$.

Let I be the initial state and G a set of literals that describes the goal states. The optimal planning problem can be defined as finding a T so that $\mathit{lits}(I)@0 \cup H_T \cup G@T$ is satisfiable and $\mathit{lits}(I)@0 \cup H_{T-1} \cup G@(T-1)$ is unsatisfiable. An optimal plan can be easily constructed from a valuation that satisfies the first clause set.

2.3 Resolution Proof Systems

The resolution rule defines one of the simplest proof systems for the propositional logic. It is used for showing that a set of clauses is unsatisfiable.

Definition 1 (Resolution) *Let $c_1 = l_1 \vee \cdots \vee l_n$ and $c_2 = \overline{l_1} \vee l'_2 \vee \cdots \vee l'_m$ be two clauses. The resolution rule allows deriving the clause $c_3 = l_2 \vee \cdots \vee l_n \vee l'_2 \vee \cdots \vee l'_m$. We use the notation $c_1, c_2 \vdash c_3$ for this. As a special case, resolving the* unit clauses *l_1 and $\overline{l_1}$ results in the empty clause.*

Definition 2 (Derivation) *Let $S = \{c_1, \ldots, c_n\}$ be a set of clauses. A* resolution derivation *of the clause c from S is any clause sequence $c'_1, \ldots, c'_m$ with the following properties.*

1. *For every $i \in \{1, \ldots, m\}$, either $c'_i \in S$ or $c'_j, c'_k \vdash c'_i$ for some $1 \leq j < k < i$.*
2. *$c'_m = c$.*

Definition 3 (Refutation) *Let $S = \{c_1, \ldots, c_n\}$ be a set of clauses. A* resolution refutation *of S (which shows S to be unsatisfiable) is any resolution derivation of the empty clause from S.*

A simple special case of the resolution rule is *unit resolution*, which infers ϕ from a unit clause l and another clause $\overline{l} \vee \phi$, where ϕ is a disjunction of literals. This rule leads to a very simple and efficient, but incomplete, inference procedure, which performs all unit resolution steps with a given clause set. This procedure can be implemented in linear time in the size of the clause set, and because there is (only) a linear number of possible unit resolution steps (each clause need to be involved at most once), it can be performed exhaustively. The fact that the clause c can be derived from S by unit resolution is denoted by $S \vdash_{UP} c$. The unit propagation procedure is a component of virtually all systematic SAT algorithms, including the Davis-Putnam procedure [Davis *et al.*, 1962] or the conflict-driven clause learning (CDCL) procedure [Mitchell, 2005].

3 Admissible Heuristics and Planning as SAT

Lower bound functions (heuristics) in algorithms such as A* are used for search space pruning. We denote a given lower bound for the distance from a state s to a goal g by $h^g(s)$.

Any admissible heuristic $h^g(s)$ is implicit in a logic formalization of planning, because admissibility means that the information given by the heuristic is a logical consequence of the formula that represent the planning problem: $h^g(s)$ is a *true* lower bound for the distance for reaching the goals from state s, not just an estimate. In this abstract sense, as far as only logical consequence is considered, all admissible heuristics are redundant.

Proposition 4 *Let h be an admissible heuristic, s a state, g a formula, T and $t \leq T$ non-negative integers, and $h^g(s) = n$. Then $lits(s)@t \cup H_T \models \neg g@t'$ for all $t' \in \{t, \ldots, \min(T, t+n-1)\}$.*

The above is simply by the definition of admissibility and the properties of the formalization of action sequences in the propositional logic: If g cannot be reached by less than n actions, then g must be false in all time points before n.

Although logically redundant, the information in admissible heuristics may be useful for *algorithms* for SAT because of the pruning of the search space.

We propose a notion of an *implementation* of an admissible heuristic for the unit propagation procedure.

Definition 5 *Let f be a fact. A clause set χ_T (for the plan length T)* implements *the admissible heuristic $h^f(s)$ if for all $t \in \{0, \ldots, T\}$, all states s, and all $t' \in \{t, \ldots, \min(T, t + h^f(s) - 1)\}$, we have $lits(s)@t \cup H_T \cup \chi_T \vdash_{UP} \neg f@t'$.*

The heuristic and χ_T may depend on a given initial state, but we have not expressed this possible dependency in the above definition. We don't discuss this issue further here.

Next we will show that H_T, alone, encodes one of the best known (but by no means the strongest) admissible heuristics, with χ_T as the empty set. This is the h_{max} heuristic of Bonet and Geffner [2001]. It is implicit in the planning as SAT approach in the strong sense that it is inferred by the unit propagation procedure. The heuristic can be defined as follows.

Definition 6 (h_{max} [Bonet and Geffner, 2001]) *The h_{max} heuristic for all $f \in F$ is defined by the equation*

$$h^f_{max}(s) = \begin{cases} 0, \textit{if } s \models f \\ \min_{a \in A, f \in \mathit{eff}(a)} \left(1 + \max_{f' \in \mathit{prec}(a)} h^{f'}_{max}(s)\right) \end{cases}$$

for which a solution can be obtained as the least fixpoint of a procedure that starts with

$$h^f_{max}(s) = \begin{cases} 0 & \textit{if } s \models f \\ \infty & \textit{otherwise} \end{cases}$$

and repeatedly performs updates $h^f_{max}(s) := \min(h^f_{max}(s), \min_{a \in A, f \in \mathit{eff}(a)}(1 + \max_{f' \in \mathit{prec}(a)} h^{f'}_{max}(s)))$ for every $f \in F$ until no change takes place.

It is easier use the above fixpoint iteration to first identify all facts with lower bound 0, then those with lower bound 1, 2 and so on. We do this implicitly in the next proof.

Theorem 7 *The empty clause set $\chi_T = \emptyset$, for any $T \geq 0$, implements h^f_{max} for any fact f.*

Proof: We have to show that $lits(s)@t \cup H_T \vdash_{UP} \neg f@t'$ for any state s, any $T \geq 0$, any fact f, and any t and t' such that $t' \in \{t, \ldots, \min(T, t + h^f_{max}(s) - 1)\}$. The proof is by nested inductions, one with T which we leave implicit to simplify the presentation.

Base case $h^f_{max}(s) = 1$: $h^f_{max}(s) = 1$ implies $s \not\models f$. Hence $\neg f@t$ is one of the unit clauses in $lits(s)@t$, and we immediately have $lits(s)@t \cup H_T \vdash_{UP} \neg f@t$.

Inductive case $h^f_{max}(s) > 1$: We show by induction that $lits(s)@t \cup H_T \vdash_{UP} \neg f@(t+i)$ for all $i \in \{0, \ldots, \min(T - t, h^f_{max}(s) - 1)\}$.

> Base case $i = 0$: Proof of $lits(s)@t \cup H_T \vdash_{UP} \neg f@t$ for $i = 0$ is as in the base case.
>
> Inductive case $i \geq 1$: Since $h^f_{max}(s) > i$, for every action $a \in A$ with $f \in \mathit{eff}(a)$ we have $h^{f'}_{max}(s) > i - 1$ for at least one $f' \in \mathit{prec}(a)$. By the outer induction hypothesis we have $lits(s)@t \cup H_T \vdash_{UP} \neg f'@(t + i - 1)$. By Formula 1 we have $lits(s)@t \cup H_T \vdash_{UP} \neg a@(t + i - 1)$. As this holds for all actions that make f true, by the frame axiom 4 and the inner induction hypothesis $lits(s)@t \cup H_T \vdash_{UP} \neg f@(t + i - 1)$ we have $lits(s)@t \cup H_T \vdash_{UP} \neg f@(t + i)$.

□

All admissible heuristics commonly used in planning are computable in polynomial time. The question of whether an admissible heuristic can be encoded compactly is essentially the question of *circuit complexity* of the corresponding lower bound functions [Balcázar *et al.*, 1988; Papadimitriou, 1994]. Since every polynomial time function can be represented as a polynomial circuit [Papadimitriou, 1994], all of these heuristics can indeed be expressed "compactly" as formulas. For some admissible heuristics it is obvious that their representation is so compact that it is practically usable, for example pattern databases, but for others it is less obvious.

4 Simulation of IDA* with SAT search

Next we give a simulation of a bounded horizon search with IDA* in the DPLL procedure [Davis *et al.*, 1962], showing that with a very simple and natural branching rule, DPLL is at least as powerful as state-space search with IDA*. DPLL can be viewed as a resolution proof system, and it is one of the least powerful of such complete systems [Beame *et al.*, 2004]. Hence this result shows that several other resolution proof systems, including conflict-driven clause learning [Beame *et al.*, 2004; Pipatsrisawat and Darwiche, 2009], are more powerful than state-space search with IDA*.

To obtain the result, we limit the DPLL procedure to choose only action variables as branching variables, assign

```
1: procedure DPLL($S$)
2: $S$ := UP($S$);
3: if $\{x, \neg x\} \subseteq S$ for any $x$ then return false;
4: $x$ := any variable such that $\{x, \neg x\} \cap S = \emptyset$;
5: if DPLL($S \cup \{x\}$) then return true;
6: return DPLL($S \cup \{\neg x\}$)
```

Figure 1: The DPLL procedure

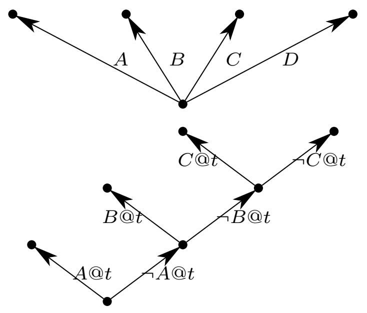


Figure 2: Branching in IDA* vs. DPLL

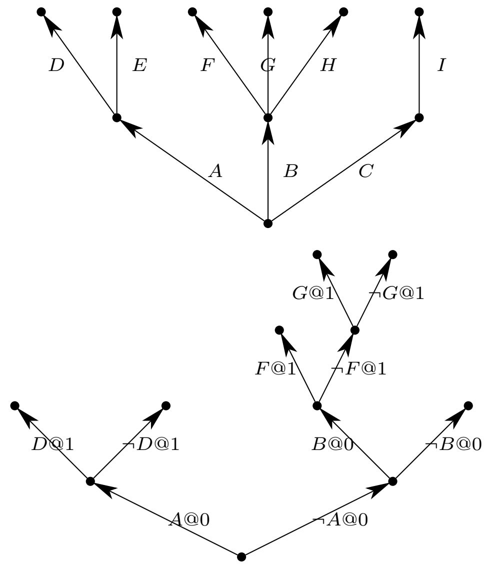


Figure 3: An IDA* search tree and a DPLL search tree

all variables for time t before proceeding with time $t+1$, and always choose the truth-value *true* first.

The next lemma shows how unit propagation computes successor states with the encoding from Section 2.2.

Lemma 8 *Let s and s' be states, $a_1, \ldots, a_k$ a sequence of actions so that $s' = succ_{a_k}(\cdots succ_{a_1}(s)\cdots)$. Let $k \leq n$. Let $L_a = \bigcup_{i=0}^{n-1} A@i$. Let $L_a^+ = \{a_1@0, \ldots, a_k@(k-1)\}$. Then $lits(s)@0 \cup H_n \cup L_a^+ \cup \{\neg a@t | a@t \in L_a \backslash L_a^+\} \vdash_{UP} lits(s')@k$.*

The main result of the section shows that DPLL search trees never have more nodes than IDA* search trees.

Theorem 9 *Let N be the size of a failed IDA* search tree with heuristic h and depth-bound T. Then there is a DPLL search tree of size less than $2N$ that shows $lits(I)@0 \cup H_T \cup \chi_T \cup g@T$ unsatisfiable.*

Proof: The DPLL procedure is given in Figure 1. It uses the unit propagation procedure UP(S), which resolves every unit clause l with clauses $\bar{l} \vee \phi$ to obtain ϕ (or the empty clause when resolving l and $\bar{l}$) which is added to S. When no further clauses can be derived, the resulting clause set is returned.

We prove that the number of nodes in the DPLL search tree is at most that of the IDA* search tree, when the variable on line 4 is always chosen so that it is an action variable $a@t$ and for all actions variables $a'@t'$ such that $t' < t$, either $a'@t' \in S$ or $\neg a'@t' \in S$.

The proof is by mapping IDA* search trees to DPLL search trees of slightly different structure, and showing that DPLL traverses the trees at most as far as IDA* does. The difference between the search trees of DPLL and state-space search with IDA* and DPLL is illustrated in Figure 2. The node in the IDA* tree on top has 4 successor nodes, corresponding to the actions A, B, C and D. The DPLL tree has binary branches only, with an action and its negation as the two arc labels, as required by the DPLL procedure. The last (rightmost) successor of any node does not have its action as a branching variables. In our example, $D@t$ is not a branching variable. Instead, it is inferred from $\neg A@t, \neg B@t, \neg C@t$ and the axiom 7 from Section 2.2. A node with n successors requires a total of $2n-1$ nodes in the corresponding DPLL search tree. Hence there are less than twice as many nodes in the DPLL tree. A full example of an IDA* search tree, with each branch cut-off after two actions, and the corresponding DPLL search tree are given in Figure 3.

When IDA* has reached a node in depth n corresponding to an action sequence $a_1, \ldots, a_n$, the corresponding state is $s' = \text{succ}_{a_n}(\cdots \text{succ}_{a_1}(I)\cdots)$. The clause set in the corresponding node in the DPLL search tree includes the same actions as literals $a_1@0, \ldots, a_n@(n-1)$ and the negations of all other action variables for time points $\leq n-1$. Hence by Lemma 8 DPLL has inferred the set $lits(s')@(n-1)$. If $n + h(s') > T$, the node will be a cut-off node for IDA*. Because χ_T implements the heuristic h, by Definition 5 we have $lits(s')@(n-1) \cup H_T \cup \chi_T \vdash_{UP} \bar{l}@T$ for at least one goal literal $l \in G$. Hence DPLL will backtrack in this node, if it has not backtracked earlier (and sometimes it has, as shown by Theorem 10.) Therefore the DPLL search tree will not be expanded further than the IDA* search tree, and it is – in terms of the number of nodes – less than twice the size. □

The above theorem assumes that the IDA* search does not detect cycles. Cycles can be eliminated from the DPLL search by encoding a constraint that says that the states at any two time points are different. Eliminating the cycles, for both IDA* and DPLL, is not necessary for the correctness or the termination of the search, but may improve performance.

The above result is not specific to DPLL. A similar simulation is possible with other systematic proof methods for SAT, for example the CDCL algorithm. A close correspondence between the branches of a search tree (paths from the root to a leaf, as in Figure 3) and the clauses learned by CDCL can be easily established. To sketch the proof idea, note that every branch in the DPLL tree in Figure 3 corresponds to a clause, for example $A@0 \vee \neg B@0 \vee F@1 \vee \neg G@1$. It can be *learned* by the CDCL algorithm, similarly to the unit resolution derivation in the proof of Lemma 8. Resolving these clauses, for example $A@0 \vee \neg B@0 \vee F@1 \vee \neg G@1$ with $A@0 \vee \neg B@0 \vee F@1 \vee G@1$, yielding $A@0 \vee \neg B@0 \vee F@1$, and this clause further with $A@0 \vee \neg B@0 \vee \neg F@1$, and so on, will eventually derive the empty clause.

5 Exponential Separation of A* and SAT

We show that some problems that are trivial for SAT (in the sense that a simple unit-resolution strategy will solve them) are very difficult for state-space search with both A* and IDA* when employing the same heuristic.

Theorem 10 *State-space search with A* and a heuristic h is sometimes exponentially slower than any algorithm for SAT that uses unit resolution, if the latter implements h.*

Proof: We give an example for which unit resolution immediately proves the lack of plans of length $n-1$ when n is the shortest plan length, and finds a plan of length n, but A* with h_{max} will visit an exponential number of states.

The idea is the following (illustrated in Figure 4.) The goals can be reached with the action sequence $y_1, \ldots, y_{k+2}$ of length $k+2$. Seemingly (as far as h_{max} is concerned), the goals can also be reached with a $k+1$ step plan consisting of k actions from $\{x_1, x'_1, \ldots, x_k, x'_k\}$ followed by bad. However, bad irreversibly falsifies one of the goals. The actions $x_1, x'_1, \ldots, x_k, x'_k$ induce a very large state space with $2^{k+1}-1$ states, which will be exhaustively searched by A*, making its runtime exponential in k.

On the other hand, unit propagation will immediately infer that action bad cannot be taken and that $y_1, \ldots, y_{k+2}$ is the only possible plan. If $T < k+2$, a refutation is immediately found. If $T = k+2$, the unit resolution procedure will find a satisfying assignment.

Next we formalize the example in detail. The state variables are $p_1, \ldots, p_{k+1}$ (of which exactly one is true in any reachable state), $r_2, \ldots, r_{k+2}$ (of which at most one is true in any reachable state), $b_1, \ldots, b_k$ (which can be independently true or false), and g_1 and g_2. Only g_1 and p_1 are initially true. All other variables are false. The goal consists of g_1 and g_2. The actions are given in Table 1.

The h_{max} estimate for the state reached by action y_1 from the initial state is $k+1$ (which is also the actual distance.)

The h_{max} estimate for all states reached from the initial state by a sequence of i actions from the set $\{x_1, \ldots, x_k, x'_1, \ldots, x'_k\}$ is $k-i+1$ (which always seems better than taking the action y_1 first), although the goals cannot be reached this way: the action bad fools h_{max} to think that goals can be reached with bad as soon as p_{k+1} has been made true, but bad actually irreversibly falsifies one of the goals.

action	precon	add	del
x_1	p_1	p_2	p_1
x'_1	p_1	p_2, b_1	p_1
x_2	p_2	p_3	p_2
x'_2	p_2	p_3, b_2	p_2
$\vdots$			
x_k	p_k	p_{k+1}	p_k
x'_k	p_k	p_{k+1}, b_k	p_k
bad	p_{k+1}	g_2	g_1
y_1	p_1	r_2	p_1
y_2	r_2	r_3	r_2
$\vdots$			
y_k	r_k	r_{k+1}	r_k
y_{k+1}	r_{k+1}	r_{k+2}	r_{k+1}
y_{k+2}	r_{k+2}	g_2	r_{k+2}

Table 1: The actions in the proof of Theorem 10

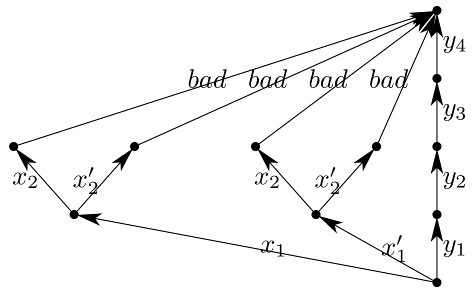


Figure 4: Illustration of the proof of Theorem 10 with $k = 2$

The A* algorithm generates 2^k states with p_{k+1} true and different values for $b_1, \ldots, b_k$, as determined by the choices between x_i and x'_i for $i \in \{1, \ldots, k\}$.

Next we show that $lits(I)@0 \cup H_{k+1} \cup \{g_1@k+1, g_2@k+1\}$ is found unsatisfiable by unit propagation.

The frame axiom for g_1 is $g_1@t \vee \neg g_1@(t+1)$ because none of the actions makes g_1 true. Hence from the goal literal $g_1@k+1$ we can infer $g_1@k, g_1@(k-1), \ldots, g_1@1$ with unit resolution, and therefore the bad action can be inferred to be never possible: we get $\neg bad@k, \ldots, \neg bad@0$.

From $\neg r_2@0$ (initial state) and $r_2@0 \vee \neg r_2@1 \vee y_2@0$ (frame axiom) and $\neg y_2@0 \vee r_2@0$ (precondition) we can infer $\neg r_2@1$ with unit resolution. Similarly we can infer $\neg r_i@1$ for all $i \geq 3$, and, more generally, $\neg r_i@j$ and $\neg y_i@j$ and $\neg g_2@(j+1)$ for all $1 < j < i \leq k+2$. Since $g_2@k+1$ belongs to the clause set, we have derived the empty clause. The number of unit resolution steps is quadratic in k and linear in the size of the clause set.

When there are $k+2$ time points, the same reasoning shows that unit propagation yields $y_1@0, \ldots, y_{k+2}@(k+1)$ and $\neg g_2@(k+1)$. As y_{k+2} is the only action that can make g_2 true, we get $y_{k+2}@(k+1)$ by unit resolution from the frame axiom. The rest of the plan is obtained analogously. $\square$

The above proof can be adapted to other admissible heuristics that use a delete-relaxation or other simplification that makes it seem that the action bad does not delete g_2.

6 Related Work

The power of different limited inference methods for approximate reachability (e.g. planning graphs) have been investigated earlier [Brafman, 2001; Geffner, 2004; Rintanen, 2008]. All three works investigate ways of strengthening inference with SAT to derive the invariants/mutexes in planning graphs. None of them attempts to relate SAT to state-space search, and the only result that is directly related to ours is Brafman's Lemma 1 which shows that for a sequential encoding (one action at each time point) Reachable-1 – which is closely related to h_{max} – is stronger than unit propagation. This seems to conflict with our Theorem 7, which shows unit propagation to be at least as strong. This discrepancy is due to our use of explanatory frame axioms. Brafman's Lemma 3 for parallel plans uses explanatory frame axioms, and then unit propagation is at least as strong as Reachability-1.

7 Conclusions

We have shown that IDA* search for state-space reachability problems can be efficiently simulated by DPLL search. Our results also show that unit resolution is sometimes exponentially more efficient than state-space search with A* or IDA*: A* and IDA* need exponential time, but short resolution proofs are immediately found with unit resolution. The work complements recent experimental results that suggest that SAT-based methods for non-optimal planning are at least as strong as state-space search [Rintanen, 2010].

One could view Theorems 9 and 10 as suggesting a simple IDA* style state-space search algorithm extended with some (unit) resolution inferences. Such a procedure would in some cases improve IDA*, but would not fully benefit from the power of modern SAT algorithms. Much of that strength comes from the less rigid structure of the proofs, which would mostly be lost in a pure forward-chaining state-space search. The general challenge is to utilize this strength and guarantee a performance that typically exceeds A* and IDA*, also for problems that would seem to be more suitable for forward-chaining state-space search.

A more direct avenue to useful implementations is to enhance SAT-based optimal planning with encodings of more powerful admissible heuristics than h_{max}. Unlike state-space search algorithms, SAT-based methods can take advantage of lower bounds for arbitrary facts (not only goals) as additional constraints. This suggests that the more powerful lower bounding methods have much more to contribute than what is currently utilized by state-space search methods. This is an important topic for future work.

References

[Balcázar *et al.*, 1988] José Luis Balcázar, Josep Díaz, and Joaquim Gabarró. *Structural Complexity I.* Springer-Verlag, Berlin, 1988.

[Beame *et al.*, 2004] Paul Beame, Henry Kautz, and Ashish Sabharwal. Towards understanding and harnessing the potential of clause learning. *Journal of Artificial Intelligence Research*, 22:319–351, 2004.

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

[Brafman, 2001] R.I. Brafman. On reachability, relevance, and resolution in the planning as satisfiability approach. *Journal of Artificial Intelligence Research*, 14:1–28, 2001.

[Davis *et al.*, 1962] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5:394–397, 1962.

[Geffner, 2004] Héctor Geffner. Planning graphs and knowledge compilation. In Didier Dubois, Christopher A. Welty, and Mary-Anne Williams, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Ninth International Conference (KR 2004)*, pages 662–672, 2004.

[Hart *et al.*, 1968] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum-cost paths. *IEEE Transactions on System Sciences and Cybernetics*, SSC-4(2):100–107, 1968.

[Kautz and Selman, 1992] Henry Kautz and Bart Selman. Planning as satisfiability. In Bernd Neumann, editor, *Proceedings of the 10th European Conference on Artificial Intelligence*, pages 359–363. John Wiley & Sons, 1992.

[Korf, 1985] R. E. Korf. Depth-first iterative deepening: an optimal admissible tree search. *Artificial Intelligence*, 27(1):97–109, 1985.

[Mitchell, 2005] David G. Mitchell. A SAT solver primer. *EATCS Bulletin*, 85:112–133, February 2005.

[Papadimitriou, 1994] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley Publishing Company, 1994.

[Pipatsrisawat and Darwiche, 2009] K. Pipatsrisawat and A. Darwiche. On the power of clause-learning SAT solvers with restarts. In I. P. Gent, editor, *Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming, CP 2009*, number 5732 in Lecture Notes in Computer Science, pages 654–668. Springer-Verlag, 2009.

[Rintanen, 2008] Jussi Rintanen. Planning graphs and propositional clause-learning. In Gerhard Brewka and Patrick Doherty, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Eleventh International Conference (KR 2008)*, pages 535–543. AAAI Press, 2008.

[Rintanen, 2010] Jussi Rintanen. Heuristics for planning with SAT. In David Cohen, editor, *Principles and Practice of Constraint Programming - CP 2010, 16th International Conference, CP 2010, St. Andrews, Scotland, September 2010, Proceedings.*, number 6308 in Lecture Notes in Computer Science, pages 414–428. Springer-Verlag, 2010.

Heuristics with Choice Variables: Bridging the Gap between Planning and Graphical Models

Emil Keyder
INRIA, France

Miquel Ramírez
Universitat Pompeu Fabra
08018 Barcelona, Spain

Héctor Geffner
ICREA & Universitat Pompeu Fabra
08018 Barcelona, Spain

Abstract

Motivated by the goal of applying inference techniques used in graphical models to planning, we introduce a novel heuristic based on the idea of choice variables, implicit multivalued variables to which no plan can assign more than one value. We adapt the recursive conditioning algorithm for calculating the probability of evidence in Bayesian networks to efficiently compute the values of this heuristic by considering problem structure and reasoning by cases about different assignments to these variables. The resulting algorithm is exponential in the treewidth of the graph describing the causal relationships between the choice variables of the problem. We present some examples of the computation of the heuristic, and discuss the issues faced in applying it to existing planning benchmarks, a goal which for now remains elusive.

Introduction

Many recently proposed techniques in planning make use of *invariants* such as landmarks and mutexes. Here, we consider a new type of invariant in planning: implicit multivalued variables to which a value can be assigned *at most once* by a plan. These variables, which we call *choice variables*, are different from standard multivalued variables in that they do not represent properties of the current state that may change in future states, but rather commitments made by the planner to solve the problem while accepting the constraints that a choice imposes upon the space of possible plans. Indeed, they are more similar in spirit to the variables used in graphical models such as constraint satisfaction problems and Bayesian networks, in which a single value that is consistent with the rest of the solution to the problem, or a single most likely instantiation, must be chosen for each variable.

Choice variables can be used as a tool for improving the quality of planning heuristics, by forcing them to respect features of a problem that otherwise might not be present in a relaxation. Consider a problem in which an agent must travel from an initial location to *one of* a set of markets and buy a set of items. In an optimal plan, the agent moves from its initial location to a market which minimizes the sum of the movement cost and the total cost of the items at that market. The delete relaxation of this problem, however, throws away information that restricts the agent to traveling to a *single* market. When movement costs are low, the optimal delete relaxation plan is then to move to all markets that carry some required item at the lowest price and buy each at a different market. The knowledge that *the set of markets constitutes a choice variable*, only one of whose values can be chosen, allows us to improve the value of a delete-relaxation heuristic by computing its value in several different versions of the same problem, in each of which the actions allowing the agent to move to all but one of the markets are *excluded*. Taking the *minimum* among these estimations then gives a strictly more informative estimate for the cost of the problem.

Since the number of possible assignments to a set of variables grows exponentially in the size of the set, the approach of enumerating all possible assignments becomes unfeasible as the number of variables increases. However, this effect can be mitigated when the interactions between choice variables are in some way *structured*. Consider a variant of the problem above in which there are several different types of market, so the agent must first go to one of a set of markets which sells food, then to one of a set of markets which sells drinks, etc. As long as the sequence in which the different types of markets must be visited is fixed, the choice of the nth market to visit is dependent only on the market chosen at step $n-1$. In effect, the graph describing the interactions between the choice variables of the problem forms a *chain*, a structure that various algorithms for graphical models can take advantage of to reduce the time complexity to linear in the number of variables. Here we adapt the recursive conditioning algorithm (Darwiche 2001) to the planning setting to take advantage of such problem structure.

The resulting heuristic is related to *factored* approaches to planning, which decompose a planning problem into a set of *factors* and attempt to find plans for each that can be interleaved with the plans for the others in order to obtain a global solution (Amir and Engelhardt 2003; Fabre et al. 2010). Such factors may share with each other either fluents or actions; our approach is similar to the former, with the shared fluents being the different values of choice variables. The number of times that the value of a variable shared between two factors must be changed has been investigated in this setting as a complexity parameter of the planning problem (Brafman and Domshlak 2006); our heuristic can be seen as having *a priori* knowledge that this parameter is limited to 1 for the set of choice variables. The behaviour of our heuristic when this assumption about the problem is not satisfied and constitutes a relaxation of the problem remains to be investigated. While factored planning methods attempt to find explicit plans for the global problem, we instead obtain heuristic estimates of the cost of solving each subproblem and combine these costs in order to obtain a heuristic for the global problem.

In this work, we focus on how to use choice variables to efficiently compute better heuristic estimates for planning problems, and assume that the choice variables themselves are given to the planner.

Background

We use the STRIPS formalization of planning, in which problem states and operators are defined in terms of a set of propositional variables, or fluents. Formally, a planning problem is a 4-tuple $\Pi = \langle F, O, I, G\rangle$, where F is such a set of variables, O is the set of operators, with each operator $o \in O$ given by a 3-tuple of sets of fluents $\langle \mathit{Pre}(o), \mathit{Add}(o), \mathit{Del}(o)\rangle$ and a cost $\mathit{cost}(o)$, $I \subseteq F$ is the initial state, and $G \subseteq F$ describes the set of goal states. A *state* $s \subseteq F$ is described by the set of fluents that are true in that state. An operator o is *applicable* in s when $\mathit{Pre}(o) \subseteq s$, and the result of applying it is $s[o] = (s \setminus \mathit{Del}(o)) \cup \mathit{Add}(o)$. The set of goal states is $\{s \mid G \subseteq s\}$, and a plan is a sequence of operators $\pi = o_1, \ldots, o_n$ such that applying it in I results in a goal state. The *cost* of π is $\sum_{i=1}^{n} \mathit{cost}(o_i)$, with an *optimal* plan π^* being a plan with minimal cost. The perfect heuristic $h^*(s)$ is the cost of an optimal plan for a state s, and an *admissible* heuristic h is one for which $h(s) \leq h^*(s)$ for all s.

The Choice Variables Heuristic

Choice variables in STRIPS can be defined as sets of fluents such that each operator adds at most one of them, and such that no plan applicable in the initial state contains more than one operator that adds some fluent in the set:

Definition 1 (Choice variable in STRIPS) *A choice variable $C_i = \{d_1, \ldots, d_n\}$ consists of a set of fluents such that $\max_{o \in O} |\mathit{Add}(o) \cap C_i| = 1$, and for any sequence of actions π applicable in I, $|\pi_{C_i}| \leq 1$, where $\pi_{C_i} = \{o \in \pi \mid \mathit{Add}(o) \cap C_i \neq \emptyset\}$.*

Given a set of choice variables $C = \{C_1, \ldots, C_n\}$, an assignment to C is a set $\mathbf{v}$ such that $|\mathbf{v} \cap C_i| \leq 1$ for $i = 1, \ldots, n$. Note that an assignment does not necessarily specify a value for each choice variable. For an assignment $\mathbf{v}$, a problem $\Pi^{\mathbf{v}}$ that *respects* $\mathbf{v}$ can be obtained by removing from the problem all operators that add a value that is inconsistent with $\mathbf{v}$:

Definition 2 ($\Pi^{\mathbf{v}}$) *Given a problem $\Pi = \langle F, I, O, G\rangle$, a set of choice variables C, and an assignment $\mathbf{v}$ to C, let $\Pi^{\mathbf{v}} = \langle F, I, O^{\mathbf{v}}, G\rangle$, where $O^{\mathbf{v}}$ is given by*

$$O^{\mathbf{v}} = \bigcap_{C_i \in C} \{o \in O \mid (C_i \setminus \mathbf{v}) \cap \mathit{Add}(o) = \emptyset\} \tag{1}$$

In other words, $O^{\mathbf{v}}$ consists of the set of operators that either add no fluent in C_i or add the value d such that $\mathbf{v} \cap C_i = \{d\}$. When $\mathbf{v} \cap C_i = \emptyset$, no operator adding any value of C_i is included in $O^{\mathbf{v}}$. Given a *base heuristic* h, we can define the *choice variable heuristic* h^c in terms of h and the problems $\Pi^{\mathbf{v}}$ for $\mathbf{v} \in \phi(C)$, where $\phi(C)$ denotes the set of possible assignments to C:

$$h^c = \min_{\mathbf{v} \in \phi(C)} h(\Pi^{\mathbf{v}}) \tag{2}$$

Proposition 3 (Admissibility and optimality of h^c) *Given a problem Π with a set of choice variables C and an admissible base heuristic h, h^c is also admissible. Furthermore, if h is the perfect heuristic h^*, then $h^c = h^*$.*

Proof sketch: Let $\mathbf{v}$ be the assignment made to C by an optimal plan π^*. Then π^* is also a plan for $\Pi^{\mathbf{v}}$, since it cannot contain any operators assigning values to C other than those in $\mathbf{v}$, and $h^*(\Pi^{\mathbf{v}}) = \mathit{cost}(\pi^*)$. The properties above follow from the fact that h^c is defined to be the minimum heuristic value for any $\Pi^{\mathbf{v}}$. □

While the value of h^c can be computed as implied by Equation 2, this requires that h be evaluated $|\phi(C)|$ times, which is exponential in $|C|$. For problems in which choice variables exhibit some degree of conditional independence, however, it is possible to reduce the number of evaluations by taking advantage of this structure, which we encode by means of the *choice variable graph*.

The Choice Variable Graph

The choice variable graph (CVG) is a directed graph $G_C = \langle C \cup C_G, E_C\rangle$, where $C_G = \cup_{g \in G}\{C_g\}$ is the set of unary *goal choice variables*, $C_g = \{g\}$. In the following, C_i may denote either one of the explicitly defined choice variables of the problem or some $C_g \in C_G$. To describe the construction of G_C, we first define the notion of *conditional relevance*:

Definition 4 (Conditional relevance) *For $p, q \in F$ and $B \subseteq F$, p is* conditionally relevant *to q given B if:*

- *$p = q$, or*
- *There exists $o \in O$ with $p \in \mathit{Pre}(o)$, $r \in (\mathit{Add}(o) \cup \mathit{Del}(o)) \setminus B$, and r is conditionally relevant to q given B.*

In other words, p is conditionally relevant to q given B if whether p is true in the current state or not changes the cost of achieving q, even when the truth values of fluents in B are held constant. We denote this by $\mathit{rel}(p, q, B)$. This definition can easily be extended to *sets* of fluents: $P \subseteq F$ is conditionally relevant to $Q \subseteq F$ given B if there exist $p \in P$, $q \in Q$ such that $\mathit{rel}(p, q, B)$. The set of edges E_C of G_C is then given by:

$$E_C = \{\langle C_i, C_j\rangle \mid \mathit{rel}(C_i, C_j, \cup_{C_k \in C \setminus \{C_i, C_j\}} C_k)\}$$

In words, there is an edge $e = \langle C_i, C_j\rangle$ in G_C if C_i is conditionally relevant to C_j given the values of all other choice variables. In what follows, we will assume that the CVG G_C given by this definition is *directed acyclic*.

G_C encodes a set of relationships between its nodes similar to those encoded by a Bayesian network, but rather than associating with each node C_i a *conditional probability* *table* (CPT) that specifies the probability of a node's taking different values given the values of its parents, it associates a *conditional cost table* (CCT) that specifies the cost of the subproblem of making true some $d \in C_i$ given an assignment to its parent nodes $Pa(C_i)$. For an assignment $\mathbf{v} \in \phi(C_i \cup Pa(C_i))$, this cost is that of a STRIPS problem with goal $G = \mathbf{v}[\{C_i\}]$ and initial state $s \cup \mathbf{v}[Pa(C_i)]$, where the notation $\mathbf{v}[C']$ denotes the set of values in $\mathbf{v}$ which belong to the domain of some $C_i \in C'$. The set of operators of the problem is given by $O^{\mathbf{v}}$, which is computed as in Definition 2.

Rather than calculating the costs of these subproblems and storing them in a table beforehand, the CCT can be represented implicitly as a heuristic that estimates the costs of these problems as they are needed. We denote the heuristic values for such a subproblem with $h(\mathbf{v}[C_i] \mid \mathbf{v}[Pa(C_i)])$. This notation parallels that used in Bayesian nets for the conditional probability of a node being instantiated to a value given the values of its parent nodes, and makes clear the relationship between the subproblems considered here and *factors* in Bayesian nets that consist of a node together with all of its parents. The *choice variable decomposition heuristic* h^{cd} can then be written as follows:

$$h^{cd}(\Pi) = \min_{\mathbf{v} \in \phi(C)} \left[\sum_{i=1}^{|G|} h(g_i \mid \mathbf{v}[Pa(C_{g_i})]) + \sum_{i=1}^{|C|} h(\mathbf{v}[C_i] \mid \mathbf{v}[Pa(C_i)]) \right] \qquad (3)$$

h^{cd} approximates the value of h^c under the assumption of acyclicity, discussed above, and the additional assumption of decomposability, which allows us to obtain heuristic estimates for the global problem by *summing* estimates for disjoint subproblems. It can be shown that under these two assumptions, the values of h^c and h^{cd} are equal for certain base heuristics, notably h^* and h^+. However, for non-optimal delete relaxation heuristics such as h^{add}, the h^{cd} heuristic can sometimes result in more accurate heuristic estimates than h^c, as partitioning the problem into subproblems allows the elimination of the some of the overcounting behaviour typically observed with h^{add}. Note that in Equation 3, while all possible assignments to non-goal choice variables are considered, the values of the goal choice variables are forced to take on their (unique) values g_i. This can be seen as analogous to the notion of *evidence* in Bayesian nets, which are nodes whose values have been observed and which therefore cannot take on other values.

To formalize the assumption of decomposability discussed above, we extend the definition of conditional relevance: an operator o is conditionally relevant to a set of fluents Q given another set of fluents B if $rel(Add(o) \cup Del(o), Q, B)$ holds. We define the *decomposability* of a problem with respect to a set of choice variables in terms of this idea:

Definition 5 (Decomposability) *A problem* Π *is* decomposable *with a set of choice variables* C *if each operator in* Π *is relevant to a* single $C_i \in C$ *given all the other choice variables in* C, *i.e. if for each* $o \in O$, $|\{C_i \mid rel(o, C_i, \cup_{C_k \in C \setminus \{C_i\}} C_k)\}| = 1$.

When this decomposability condition is not met, the cost of o may be counted in more than one subproblem, leading to an inadmissible heuristic even if the underlying heuristic used to compute the costs of subproblems is admissible.

Proposition 6 (Equivalence of h^c and h^{cd}) *Given a problem* Π *with a set of choice variables* C *such that the CVG* G_C *is directed acyclic and* Π *is decomposable with* C*, and base heuristic* h^* *or* h^+, $h^c = h^{cd}$.

Proof sketch: The proof follows from the fact that due to the definition of decomposability, an optimal (relaxed) plan can be partitioned into a set of subsets such that each constitutes a (relaxed) plan for a subproblem in h^{cd}, and optimal (relaxed) plans for each subproblem can be combined to obtain a global optimal (relaxed) plan. □

Proposition 7 (Admissibility and optimality of h^{cd}) *Given a problem* Π *with a set of choice variables* C *such that the CVG* G_C *is directed acyclic and* Π *is decomposable with* C*, and an admissible base heuristic* h, h^{cd} *is also admissible. Furthermore, if* $h = h^*$*, then* $h^{cd} = h^*$.

Proof sketch: The proof follows from that of Proposition 3, and the observation that each operator in an optimal plan π^* contributes cost to exactly one subproblem in h^{cd} due to decomposability. Optimal (admissible) estimates for each subproblem can therefore be summed to obtain optimal (admissible) estimates for the global problem. □

Efficient Computation of h^{cd}

We now turn our attention to how to take advantage of the conditional independence relations between choice variables encoded by G_C, showing how to adapt the recursive conditioning algorithm (Darwiche 2001) to the computation of h^{cd}. Given a Bayesian network $\mathcal{N}$ and evidence $\mathbf{e}$ in the form of an observed instantiation of a subset of the nodes of $\mathcal{N}$, the recursive conditioning algorithm operates by selecting a *cutset* C that when instantiated results in two connected components $\mathcal{N}^l, \mathcal{N}^r$ that are *conditionally independent* of one another given C. The method then enumerates the possible instantiations $\mathbf{c}$ of C, recording each and solving $\mathcal{N}^l$ and $\mathcal{N}^r$ by applying the same method recursively with the pertinent evidence. In enumerating the possible instantiations of the cutset, the observed values of nodes that constitute the evidence $\mathbf{e}$ are respected. When the network on which the method is called consists of a single node, the probability corresponding to the current instantiations of the node and its parents can be looked up directly in the associated CPT. Given two conditionally independent networks $\mathcal{N}^l, \mathcal{N}^r$, the joint probability of the evidence for an instantiation $\mathbf{c}$ of the cutset C is calculated as the product of the probability of the evidence for each, and these results for all possible instantiations of C are summed to obtain the total probability.

The recursive conditioning algorithm can be driven by a structure known as a *dtree*, a binary tree whose leaves correspond to the factors of $\mathcal{N}$ (Darwiche 2001). Each internal node d of the dtree corresponds to a subnetwork $\mathcal{N}^d$ of

$\mathcal{N}$, with the root node corresponding to the full network, and specifies a cutset consisting of those variables shared between its two children d^l and d^r which do not appear in its *acutset*, defined as the set of variables that appear in the cutsets of the ancestors of d. An instantiation of all of the variables that appear in a node's cutset and acutset then ensures that all variables shared between its two children are instantiated, resulting in two networks which are conditionally independent given the instantiation. A dtree node specifies that recursive conditioning be applied to the associated network by instantiating its cutset, and then applying recursive conditioning to the resulting two networks $\mathcal{N}^l$ and $\mathcal{N}^r$ by using d^l and d^r as dtrees for those.

To avoid repeated computations, each dtree node may also maintain a *cache*, which records for each instantiation $\mathbf{y}$ of its *context*, defined as the intersection of the set of variables of the corresponding subnetwork with the nodes of its acutset, the probability resulting from $\mathbf{y}$. While this increases the space requirements of the algorithm, in many settings the associated gains in time are of greater importance. With full caching, recursive conditioning driven with such a dtree is guaranteed to run in $O(n^w)$ time, where n is the number of nodes in the network and the *width* w is a property of the dtree. Given an *elimination ordering* of width w for a network, a dtree with width $\leq w$ can be constructed in linear time.[1] While finding the optimal elimination ordering for a graph which results in the lowest width dtree is an NP-hard problem, many greedy heuristics provide reasonable performance in practice (Dechter 2003).

In order to compute h^{cd} using recursive conditioning, we replace the CPTs associated with each factor with CCTs represented implicitly by a *heuristic estimator* h, which rather than calculating the probability of some instantiation of a node given an instantiation of its parent nodes, estimates the *cost* of achieving a value of a choice variable given values in the domains of its parent variables. To obtain the cost resulting from an instantiation of a cutset, the costs of the two components are *summed* rather than multiplied, and the *minimum* such cost is taken over all of the possible instantiations of the cutset. The fact that each of the goal choice variables C_g must be assigned its single possible value g, and that a choice variable may already be assigned a value in a state from which the h^{cd} is computed, can be seen as equivalent to evidence in the Bayesian networks setting.

The modifications described above result in Algorithm 1. Since the CVGs that we consider are typically small and the time taken by computing the value of a heuristic at each state is the largest factor in heuristic search planners' runtime, we use full caching of computed costs for each subproblem. We construct the dtree used in the algorithm from the elimination ordering suggested by the greedy *min-degree* heuristic, which orders the nodes last to first in increasing order of their degree. Finally, the values of choice variables in the state from which the heuristic is computed, as well as the single values of each of the goal choice variables, are given to the algorithm as evidence.

[1]Elimination orderings and width are beyond the scope of this paper. For an overview of the subject, see e. g. Bodlaender.

```
Input: A dtree node $D$
Input: An assignment $\mathbf{v}$ to $C$
Input: A base heuristic function $h$
Output: A heuristic estimate $h^{cd} \in \mathbb{R}_0^+$
function RC-h ($D, \mathbf{v}$) begin
    if $D$ is a leaf node then
        $C_i \leftarrow$ the choice variable associated with $D$
        return $h(\mathbf{v}[C_i] \mid Pa(C_i))$
    endif
    else if $cache_D[\mathbf{v}[context(D)]] \neq undefined$ then
        return $cache_D[\mathbf{v}[context(D)]]$
    endif
    else
        $h^{cd} \leftarrow \infty$
        for $\mathbf{c} \in \phi(cutset(D))$ do
            $h^l \leftarrow$ RC-h($D^l, \mathbf{v} \cup \mathbf{c}$)
            $h^r \leftarrow$ RC-h($D^r, \mathbf{v} \cup \mathbf{c}$)
            $h^{cd} \leftarrow \min(h^{cd}, h^l + h^r)$
        end
        $cache_D[\mathbf{v}[context(D)]] \leftarrow h^{cd}$
        return $h^{cd}$
    endif
end
```

Algorithm 1: Recursive conditioning for calculating h^{cd} in a planning problem with choice variables.

Proposition 8 (Correctness of RC-h) *If G_C is acyclic, the RC-h algorithm computes the values of the choice variable decomposition heuristic h^{cd}.*

We omit the proof due to lack of space.

Proposition 9 (Complexity of RC-h) *The number of calls made to the underlying heuristic estimator h by RC-h is $O(n^w)$, where n is the number of nodes in the CVG G_C and w is the width of the dtree used.*

Proof: Direct from results concerning the complexity of recursive conditioning (Darwiche 2001). □

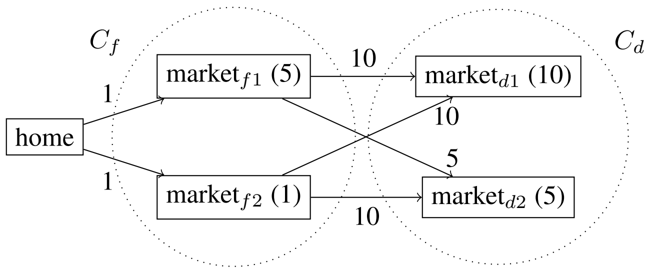


Figure 1: A market problem. Food is available at *market$_{f1}$* and *market$_{f2}$*, while drink is available at *market$_{d1}$* and *market$_{d2}$*. Numbers in parenthesis give the cost of each good at the specified market, and those next to each edge show the movement cost.

Example. Consider a problem in which an agent must buy food and drink, and two different markets are available selling each (Figure 1). The CVG for this problem is a tree, with the choice of the food market independent of all other choice variables, the choice of the drink market dependent only on the food market, and the choice variables for the two goals *have-food* and *have-drink* dependent only on the market chosen for each (Figure 2). The RC-h algorithm begins at the root node of the dtree (Figure 3), and must enumerate the possible instantiations of its cutset $\{C_f\}$. *market*$_{f1}$ is selected as the first value, and the recursion proceeds with a call to the right subtree. This is an internal node with an empty cutset, so no further variables are instantiated. The algorithm then recurses to the right child, which is a leaf node in which the base heuristic is used to estimate the cost of the goal *have-food* from the initial state $\{home, market_{f1}\}$, with actions adding other values of the choice variable C_f disallowed. There is a single possible plan for this component which consists of buying food at *market*$_{f1}$, and this plan has cost 5, which is returned to the parent node. The algorithm then proceeds to evaluate the cost of the left node, which is the cost of making *market*$_{f1}$ true from the initial state $\{home\}$, 1. Since the cutset of the node is empty, no further instantiations are required and the value $5 + 1 = 6$ is returned to the root node. The algorithm now estimates the cost of the node with cutset $\{C_d\}$, instantiating it first to *market*$_{d1}$. In the call to the right child, the cost of achieving *market*$_{d1}$ from initial state $\{home, market_{f1}\}$ is evaluated, to give cost 10, and in the left child, the cost of buying drink at *market*$_{d1}$ is evaluated, to give cost 10. The returned values are summed in the internal node with cutset C_d to give cost 20. The value resulting from $C_d = market_{d2}$ is calculated similarly, giving cost $5 + 5 = 10$, which is lower than the previous instantiation, and therefore returned to the root node, in which it is summed with the cost calculated for the right child to give a final cost of $10 + 6 = 16$ for the instantiation $C_f = market_{f1}$. The cost of choosing $C_f = market_{f2}$ is computed similarly and turns out to be 17, which is higher than the previous estimate, so 16 is returned as the final value. Note that the optimal delete relaxation plan for this problem is to move to *market*$_{f2}$ and buy food there, and to move from *home* to *market*$_{f1}$ and from there to *market*$_{d2}$ to buy drinks there, for a total cost of $1 + 1 + 1 + 5 + 5 = 13$.

The "sequential markets problem" problem can also be seen as the most probable explanation (MPE) problem on hidden Markov models (HMMs). HMMs are dynamic Bayesian processes defined by $H = \langle S, O, T, I, E\rangle$, where S is a set of states, O is a set of observations, $T(s' \mid s)$ for $s, s' \in S$ is the probability of transitioning from s to s', $I(s)$ for $s \in S$ is the probability that the initial state is s, and $E(o|s)$ for $o \in O$, $s \in S$ is the probability of observation o in state s. The MPE problem for HMMs is to compute a sequence of states $s_0, \ldots, s_t$ that maximizes the probability of a sequence of observations $o_1, \ldots, o_t$, where this probability is given by $I(s_0) \prod_{i=1}^{t} T(s_i \mid s_{i-1}) E(o_i \mid s_i)$. An MPE problem for an HMM can be encoded as a market problem with a sequence of t different market types, at each of which there is a choice of $|S|$ different markets from which the single required item of that type must be bought. The costs of buying the required item and moving between markets are obtained by taking the negative logarithm of the associated probabilities, so that finding a plan with minimal cost is equivalent to finding a state trajectory with maximum probability for the HMM.

$$\begin{tikzcd}
C_{g_f} & C_{g_d} \\
C_f \arrow[u] \arrow[r] & C_d \arrow[u]
\end{tikzcd}$$

Figure 2: The CVG for the two markets problem shown in Figure 1, with factors $C_f, C_f \to C_d, C_f \to C_{g_f}, C_d \to C_{g_d}$.

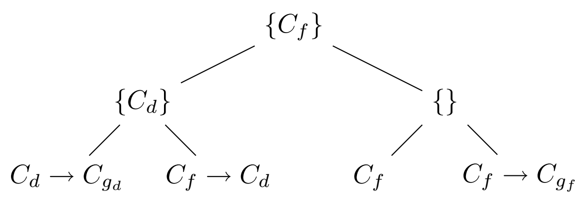


Figure 3: A dtree for the CVG shown in Figure 2, resulting from the min-degree elimination ordering $\langle C_{g_d}, C_{g_f}, C_d, C_f\rangle$. Cutsets are shown for internal nodes.

Implementation

The recursive conditioning algorithm naturally avoids redundant computations by caching for each node the values resulting from different assignments to its context. However it is designed to compute the probability of a single piece of evidence and therefore makes no attempt to preserve information between calls. We observe that the heuristic values $h^{\mathbf{v}}(C_i = \mathbf{v}[C_i] \mid Pa(C_i))$ do not change from state to state for components in which there is no non-choice fluent that is conditionally relevant to C_i given $Pa(C_i)$. These values can then be cached in each leaf node of the dtree the first time they are computed, and reused in later heuristic computations. We have implemented this optimization in the results described below.

Domains and Experimental Results

We have found that in practice few problems conform to the rather strict requirements that we have laid out for h^{cd}. Multivalued variables in planning problems typically represent properties of states, rather than choices made by plans, and their values tend to change many times during the execution of a plan, violating the fundamental property that we have used to define choice variables. One technique that we have investigated in certain problems is *stratification*, which consists of replacing a single multivalued variable X with several multivalued variables $X_0, \ldots, X_n$, with X_i representing the ith value assigned to X in the original problem. Each of these variables can than be treated as a choice variable in the new encoding, and the part of the CVG corresponding to these variables forms a chain in which each X_i is dependent only on X_{i-1}, leading to a width of 1. However this approach does not generally pay off as variables that were previously dependent only on X now depend on each of the variables X_i, increasing the width of the graph to the horizon used to obtain the new encoding.

We were also initially optimistic that h^{cd} could give informative estimates in domains such as Sokoban or Storage, in which a number of objects have to be placed in a set of goal locations. The final goal location chosen for an object can here be seen as a choice variable. However, in the absence of an ordering over the objects, the position of each object affects the available choices for the others, and even if an ordering is imposed, the cost of the last object to be placed is dependent on the locations of all other objects, leading to the CVG's width growing with problem size.

One area in which it has been easy to apply h^{cd}, however, is in planning encodings of problems from the graphical models setting. We now present two domains that we have adapted to the planning setting for which h^{cd} is able to compute optimal values. We compare the performance of h^{cd} to that of a standard delete relaxation heuristic, the cost of the relaxed plan obtained from the additive heuristic h^{add} (Keyder and Geffner 2008). We use the same heuristic within the framework of h^{cd} to obtain the heuristic estimates for the cost of each subproblem $h(C_i \mid Pa(C_i))$. The heuristics are used in greedy best-first search with delayed evaluation, with a second open queue for states resulting from helpful actions (Helmert 2006). Helpful actions for h^{cd} are those which are helpful in any one of the subproblems and which can be applied in the current state. All experiments were run on Xeon Woodcrest 2.33 GHz computers, with time and memory cutoffs of 1800 seconds and 2GBs, respectively.

Minimum Cost SAT. As a canonical example of a constraint satisfaction problem, we consider a version of the boolean satisfiability problem with three literals per clause in which a satisfying assignment with minimum cost must be found (MCSAT) (Li 2004). The natural choice variables for the problem consist of the sets $\{x_i, \neg x_i\}$ for each problem variable x_i. An encoding that results in an acyclic CVG is obtained by imposing an ordering over the set of variables of the problem, and using operators which can set variables which are not highest-ranked in any clause freely, but which ensure for the other variables that the clauses in which they are highest-ranked are already satisfied if the assignment made by the operator itself does not satisfy the clause. We omit some details of the encoding here due to lack of space, but the end result is that h^{cd} can be used to compute the optimal cost in this type of encoding. The number of evaluations of the base heuristic for a single call to h^{cd} is exponential in the width of the ordering over the variables that is used to generate the problem.

We generated random MCSAT problems with 4.3 times as many clauses as variables, a ratio that has been shown to produce problems for which satisfiability testing is hard (Mitchell, Selman, and Levesque 1992). The number of variables in the problems ranged from 5 to 34. The choice variable heuristic h^{cd} is optimal for this domain, while the additive heuristic h^{add} produces both overestimates and underestimates, with the error in the estimation growing as the number of variables increases. When used in search, h^{cd} is able to solve problems of up to 25 variables with orders of magnitude fewer node expansions than h^{add}. However, for the ratio of clauses to variables that we consider, the CVG is usually clique-like, and therefore has treewidth close to the number of variables in the problem. The computation of h^{cd} therefore rapidly becomes unfeasible, while h^{add} is able to scale up to larger instances. The higher informativeness of h^{cd} does not pay off in terms of plan cost either, with the plans found having roughly equal cost for both heuristics.

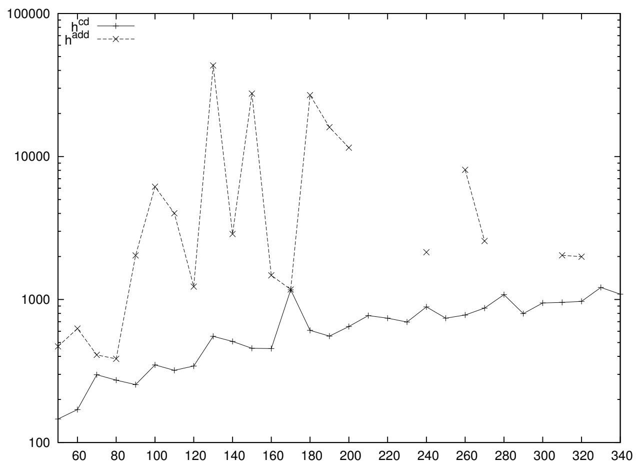


Figure 4: Node expansions on the Sequential Markets domain.

Sequential Markets Problem. We generated market problems such as those described above, with the number of markets of each type fixed at 15, and the number of different types of markets varying between 50 and 340. The h^{cd} heuristic computes optimal heuristic values for this domain that are 2-6 times cheaper than those computed by h^{add}, with the effect becoming more pronounced as the number of market types is increased. When the two heuristics are used in greedy best-first search, h^{cd} scales up well due to the constant width of the domain, solving all 30 problems with much fewer node evaluations than those required by h^{add} in the 21 problems it is able to solve (Figure 4). The computation time of h^{cd} is roughly 10 - 20 times slower than that of h^{add} in this domain, and due to the constant width of the CVG does not change as problem size is varied. The heuristic is also beneficial in terms of the costs of the plans that are found, with h^{cd} finding lower cost plans for all of the instances solved with both heuristics, and the difference in cost growing with the size of the problem.

Conclusions

We have introduced a new type of invariant in planning that we call choice variables, multivalued variables whose values can be set at most once by any plan. By reasoning by cases about different assignments to these variables and excluding operators that violate these assignments, we obtain the h^{cd} heuristic that goes beyond the delete relaxation in its estimates. The values of this heuristic can be computed by adapting inference techniques for graphical models to the planning setting, with the complexity of the heuristic computation then depending on the treewidth of the graph that describes the causal relationships between choice variables.

Our attempts to apply h^{cd} to benchmark planning problems have until now proved disappointing due to the lack of domains with variables that naturally exhibit the choice variable property. We have investigated the use of new encodings to induce this structure, however such transformations usually result in the treewidth of the CVGs increasing with domain size, and the computational overhead of the heuristic swiftly becoming impractical. We hope to eventually address this challenge by considering meaningful relaxations of these graphs that decrease the width parameter while respecting the essential features of the problem.

References

Amir, E., and Engelhardt, B. 2003. Factored planning. In *Proc. of the 18th IJCAI*, 929–935.

Bodlaender, H. L. 1993. A tourist guide through treewidth. *Acta Cybernetica* 11:1–23.

Brafman, R. I., and Domshlak, C. 2006. Factored planning: How, when, and when not. In *Proc. of the 21st AAAI*, 809–814.

Darwiche, A. 2001. Recursive conditioning. *Artificial Intelligence* 126(1-2):5–41.

Dechter, R. 2003. *Constraint Processing*. Morgan Kaufmann.

Fabre, E.; Jezequel, L.; Haslum, P.; and Thiébaux, S. 2010. Cost-optimal factored planning: Promises and pitfalls. In *Proc. of the 20th ICAPS*, 65–72.

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

Keyder, E., and Geffner, H. 2008. Heuristics for planning with action costs revisited. In *Proc. of the 18th ECAI*, 588–592.

Li, X. Y. 2004. *Optimization Algorithms for the Minimum-Cost Satisfiability Problem*. Ph.D. Dissertation, North Carolina State University.

Mitchell, D. G.; Selman, B.; and Levesque, H. J. 1992. Hard and easy distributions of SAT problems. In *Proc. of the 12th AAAI*, 459–465.

On Satisficing Planning with Admissible Heuristics

Roei Bahumi and **Carmel Domshlak** and **Michael Katz**
Faculty of Industrial Engineering & Management
Technion, Israel

Abstract

Heuristic forward search is at the state of the art of sequential satisficing planning. The heuristics in use are, however, inadmissible, and thus give no guarantees on the quality of the obtained solution. Although there is no theoretical limitation in adopting admissible heuristics for satisficing planning, in practice there are several obstacles, such as lack of definition of one important feature, called helpful actions or preferred operators. In this paper we present a definition of preferred operators for the fork-decomposition abstraction heuristics and perform an extensive empirical evaluation on a range of domains from International Planning Competitions. In addition, we examine a mixed setting of using fork-decomposition heuristics with preferred operators derived from delete-relaxation based h_{FF} machinery.

Introduction

Heuristic search, either through progression in the space of world states or through regression in the space of subgoals, is a common and successful approach to classical planning. These days, most leading planning systems for cost-oriented classical planning adopt the forward search approach, as well as several enhancements such as helpful actions or preferred operators (Hoffmann & Nebel, 2001; Helmert, 2006) and deferred evaluation (Helmert, 2006). The notion of helpful actions refers to the actions that lead toward a solution of a simplified task (Hoffmann & Nebel, 2001), and thus are preferred over others. Deferred evaluation is another enhancement, allowing to evaluate the node only when it is expanded, and not when generated. Together, these enhancements are the cornerstone of several state-of-the-art planning systems (Richter & Helmert, 2009). All these systems adopt this or another inadmissible (but purportedly well-informed) heuristic, both for node evaluation and for deriving preferred operators. The reason for adopting the same heuristic for both is simple - the heuristic calculation is usually allowing for deriving preferred operators in little or no additional effort. The fact that inadmissible heuristics rule the area of satisficing planning should probably be attributed to two factors. First, the major breakthroughs in developing domain-independent admissible heuristics have been achieved only in the recent few years. Second, as the focus of these developments was on optimal planning, no mechanisms for deriving preferred operators in the scope of these admissible heuristics have been suggested.

With the recent substantial advances in admissible heuristics for cost-optimal classical planning, employing them not only in optimal but also in satisficing search became appealing. In this paper we show how to efficiently compute the set of preferred operators for fork-decomposition abstraction heuristics. We then empirically evaluate the efficiency of satisficing planning with these admissible heuristics. We adopt the deferred evaluation approach and investigate various settings of preferred operators, both from the fork-decomposition abstractions, as well as from the delete-relaxation based mechanism of h_{FF} (Hoffmann & Nebel, 2001) that is in use by most state-of-the-art satisficing planners.

Preliminaries

We consider classical planning tasks corresponding to state models with a single initial state and only deterministic actions. Specifically, we consider state models captured by the SAS^+ formalism (Bäckström & Nebel, 1995) with nonnegative action costs. Such a *planning task* is given by a quintuple $\Pi = \langle V, A, I, G, cost\rangle$, where:

- V is a set of *state variables*, with each $v \in V$ being associated with a finite domain $\mathcal{D}(v)$. For a subset of variables $V' \subseteq V$, we denote the set of assignments to V' by $\mathcal{D}(V') = \times_{v \in V'} \mathcal{D}(v)$. Each complete assignment to V is called a *state*, and $S = \mathcal{D}(V)$ is the *state space* of Π. I is an *initial state*. The *goal* G is a partial assignment to V; a state s is a *goal state* iff $G \subseteq s$.
- A is a finite set of *actions*. Each action a is a pair $\langle \mathsf{pre}(a), \mathsf{eff}(a)\rangle$ of partial assignments to V called

preconditions and *effects*, respectively. By $A_v \subseteq A$ we denote the actions affecting the value of v. *cost* : $A \rightarrow \mathbb{R}^{0+}$ is a real-valued, nonnegative *action cost* function.

For a partial assignment p, $\mathcal{V}(p) \subseteq V$ denotes the subset of state variables instantiated by p. In turn, for any $V' \subseteq \mathcal{V}(p)$, by $p[V']$ we denote the value of V' in p; if $V' = \{v\}$ is a singleton, we use $p[v]$ for $p[V']$. For any sequence of actions ρ and variable $v \in V$, by $\rho_{\downarrow_v}$ we denote the restriction of ρ to actions changing the value of v; that is, $\rho_{\downarrow_v}$ is the maximal subsequence of ρ consisting only of actions in A_v.

An action a is applicable in a state s iff $s[v] = \mathsf{pre}(a)[v]$ for all $v \in \mathcal{V}(\mathsf{pre}(a))$. The set of all applicable in state s actions is denoted by $A(s)$. Applying a changes the value of $v \in \mathcal{V}(\mathsf{eff}(a))$ to $\mathsf{eff}(a)[v]$. The resulting state is denoted by $s[\![a]\!]$; by $s[\![\langle a_1, \dots, a_k\rangle]\!]$ we denote the state obtained from sequential application of the (respectively applicable) actions $a_1, \dots, a_k$ starting at state s. Such an action sequence is an *s-plan* if $G \subseteq s[\![\langle a_1, \dots, a_k\rangle]\!]$, and it is a *cost-optimal* (or, in what follows, *optimal*) s-plan if the sum of its action costs is minimal among all s-plans. The purpose of (optimal) planning is finding an (optimal) I-plan. For a pair of states $s_1, s_2 \in S$, by $cost(s_1, s_2)$ we refer to the cost of a cost-optimal plan from s_1 to s_2; $h^*(s) = \min_{s' \supseteq G} cost(s, s')$ is the custom notation for the cost of the optimal s-plan in Π. Finally, important roles in what follows are played by a pair of standard graphical structures induced by planning tasks.

- The *causal graph* $\mathit{CG}(\Pi)$ of Π is a digraph over nodes V. An arc (v, v') is in $\mathit{CG}(\Pi)$ iff $v \neq v'$ 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 this case, we say that (v, v') is *induced* by a. By $\mathsf{succ}(v)$ and $\mathsf{pred}(v)$ we respectively denote the sets of immediate successors and predecessors of v in $\mathit{CG}(\Pi)$.
- The *domain transition graph* $\mathit{DTG}(v, \Pi)$ of a variable $v \in V$ is an arc-labeled digraph over the nodes $\mathcal{D}(v)$ such that an arc (ϑ, ϑ') labeled with $\mathsf{pre}(a)[V \setminus \{v\}]$ and $cost(a)$ exists in the graph iff both $\mathsf{eff}(a)[v] = \vartheta'$, and either $\mathsf{pre}(a)[v] = \vartheta$ or $v \notin \mathcal{V}(\mathsf{pre}(a))$.

Heuristic functions are used by informed-search procedures to estimate the cost (of the cheapest path) from a search node to the nearest goal node. Our focus here is on *additive implicit abstraction heuristics* (Katz & Domshlak, 2010b, 2010a), that are based on two fragments of tractable cost-optimal planning for tasks with fork and inverted-fork structured causal graphs. In general, for a planning task $\Pi = \langle V, A, I, G, cost\rangle$ over states S, an additive implicit abstraction of Π is denoted by a set of triplets $\mathcal{AE} = \{\langle \Pi_i, \alpha_i, \beta_i\rangle\}_{i=1}^m$, where $\Pi_i = \langle V_i, A_i, I_i, G_i, cost_i\rangle$ over states S_i is called an *abstract task*, $\alpha_i : S \mapsto S_i$ is the *state abstraction function*, and $\beta_i : A_i \mapsto A$ is the function connecting the abstract actions to their origin. The state transition system of Π_i is an abstraction of the state transition system of Π, and the admissibility of the additive result is obtained by emposing the *additive action-cost partitioning* constraint

$$\forall a \in A : \sum_{i=1}^{m} \sum_{a' \in \beta_i^{-1}(a)} cost_i(a') \leq cost(a). \qquad (1)$$

Dominating Actions

Let $\Pi = \langle V, A, I, G, cost\rangle$ be a planning task over the states S. For each state $s \in S$, the set of *dominating actions* for s, denoted by $\mathrm{Pref}_{\Pi}(s)$, is the set of all actions $a \in A$ applicable in s, each starting some cost-optimal s-plan, that is,

$$\mathrm{Pref}_{\Pi}(s) = \{a \in A(s) \mid h^*(s) = cost(a) + h^*(s[\![a]\!])\}.$$

The notion of dominating actions complements the notion of *useless actions* (Wehrle, Kupferschmid, & Podelski, 2008); deciding whether an action is useless is in general as hard as planning itself.

Considering the family of abstraction heuristics, note that computing dominating actions for explicit abstractions (such as projection/pattern and merge-and-shrink abstractions) can straightforwardly be accomplished in time polynomial in the size of these abstractions. Next we show that the same holds for implicit fork-decomposition abstractions, though the corresponding procedures are not that straightforward.

Theorem 1 *Let $\Pi = \langle V, A, I, G, cost\rangle$ be a planning task with a fork causal graph rooted at a binary-valued variable r. For any set of states $S' \subseteq S$, the time and space complexity of computing the sets $Pref_{\Pi}(s)$ for all states $s \in S'$ is, respectively, $O(d^3 \cdot |V| + |A_r| + |S'| \cdot d \cdot |V|)$ and $O(d^2 \cdot |A|)$, where $d = \max_v \mathcal{D}(v)$.*

Proof: The proof is by a slight modification of the polynomial-time algorithm for computing $h^*(s)$ for a state s of such a task Π used in the proof of Theorem 7 (Tractable Forks) by Katz and Domshlak (2010b).

In what follows, for each of the two root's values $\vartheta \in \mathcal{D}(r)$, $\neg\vartheta$ denotes the opposite value $1 - \vartheta$, $\sigma(r|\vartheta)$ denotes the interchanging sequence of values from $\mathcal{D}(r)$ starting with ϑ, $\trianglerighteq[\sigma(r|\vartheta)]$ denotes the set of all valid prefixes of $\sigma(r|\vartheta)$, and $\mathit{DTG}_v^{\vartheta}$ denotes subgraph of $\mathit{DTG}(v, \Pi)$ obtained from the latter by removing arcs labelled with $\neg\vartheta$.

(1) For each of the two values $\vartheta_r \in \mathcal{D}(r)$ of the root variable, each leaf variable $v \in V \setminus \{r\}$, and each pair of values $\vartheta, \vartheta' \in \mathcal{D}(v)$, let $p_{\vartheta,\vartheta';\vartheta_r}$ be the cost of the cheapest sequence of actions changing v from ϑ to ϑ' *provided* $r : \vartheta_r$. In parallel, let $A_{\vartheta,\vartheta';\vartheta_r}$ be the set of all possible first actions of

such sequences. Both $\{p_{\vartheta,\vartheta';\vartheta_r}\}$ and $\{A_{\vartheta,\vartheta';\vartheta_r}\}$ for all the leaves $v \in V \setminus \{r\}$ can be computed by a straightforward variant of the all-pairs-shortest-paths, Floyd-Warshall algorithm on $\mathit{DTG}_v^{\vartheta_r}$ in time $O(d^3|V|)$.

(2) For each leaf variable $v \in V \setminus \{r\}$, $\vartheta \in \mathcal{D}(v)$, $1 \leq i \leq d+1$, and $\vartheta_r \in \mathcal{D}(r)$, let $\tilde{g}_{\vartheta;i}(\vartheta_r)$ be the cost of the cheapest sequence of actions changing ϑ to $G[v]$ *provided* the value changes of r induce a $0/1$ sequence of length i starting with ϑ_r. In parallel, let $\tilde{A}_{\vartheta;i}(\vartheta_r)$ be the set of first actions of all such sequences, provided that these actions prevailed either by ϑ_r or nothing at all. The set $\{\tilde{g}_{\vartheta;i}(\vartheta_r)\}$ is given by the solution of the recursive equation

$$\tilde{g}_{\vartheta;i}(\vartheta_r) = \begin{cases} p_{\vartheta,G[v];\vartheta_r}, & i=1 \\ \min\limits_{\vartheta'} \begin{bmatrix} p_{\vartheta,\vartheta';\vartheta_r} + \\ \tilde{g}_{\vartheta';i-1}(\neg\vartheta_r) \end{bmatrix}, & 1 < i \leq \delta_\vartheta \\ \tilde{g}_{\vartheta;i-1}(\vartheta_r), & \delta_\vartheta < i \leq d+1 \end{cases}, \tag{2}$$

which can be solved in time $O(d^3|V|)$, and then, $\tilde{A}_{\vartheta;i}(\vartheta_r)$ can be obtained recursively in time $O(d^3|V|)$ as

$$\tilde{A}_{\vartheta;i}(\vartheta_r) = \begin{cases} A_{\vartheta,G[v];\vartheta_r}, & i=1 \\ \bigcup\limits_{\vartheta' \in \mathrm{M}_{\vartheta;i}(\vartheta_r)} A_{\vartheta,\vartheta';\vartheta_r}, & 1 < i \leq \delta_\vartheta \\ \tilde{A}_{\vartheta;i-1}(\vartheta_r), & \delta_\vartheta < i \leq d+1 \end{cases},$$

where

$$\mathrm{M}_{\vartheta;i}(\vartheta_r) = \{\vartheta' \mid \tilde{g}_{\vartheta;i}(\vartheta_r) = p_{\vartheta,\vartheta';\vartheta_r} + \tilde{g}_{\vartheta';i-1}(\neg\vartheta_r)\}.$$

Note that this equation is independent of the evaluated state s, and yet $\{\tilde{g}_{\vartheta;i}(\vartheta_r)\}$ allow for computing $h^*(s)$ for a given state s via

$$h^*(s) = \min_{\sigma \in \trianglerighteq[\sigma(r|s[r])]} \left[\mathit{cost}(\sigma) + \sum_{v \in V \setminus \{r\}} \tilde{g}_{s[v];|\sigma|}(s[r]) \right] \tag{3}$$

where $\mathit{cost}(\sigma) = \sum_{i=2}^{|\sigma|} \mathit{cost}(a_{\sigma[i]})$, with $a_{\sigma[i]} \in A$ being some cheapest action changing the value of r from $\sigma[i-1]$ to $\sigma[i]$. Let $\mathrm{M}(s) \subseteq \trianglerighteq[\sigma(r \mid s[r])]$ denote the set of all sequences that obtain the minimum in Eq. 3. Now, if changing the root value first can be a part of some optimal plan, that is, $h^*(s) = h^*(s[\![a_{\sigma[2]}]\!]) + \mathit{cost}(a_{\sigma[2]})$, then the respective action is in $\mathrm{Pref}_\Pi(s)$. Note that using Eq. 2 we can rewrite this condition as $\tilde{g}_{s[v];|\sigma|}(s[r]) = \tilde{g}_{s[v];|\sigma|-1}(\neg s[r])$ for all $v \in V \setminus \{r\}$. Let

$$\mathrm{M}(\sigma, s) = \bigcap_{v \in V \setminus \{r\}} \{a_{\sigma[2]} \mid \tilde{g}_{s[v];|\sigma|}(s[r]) = \tilde{g}_{s[v];|\sigma|-1}(\neg s[r])\}$$

denote the set of all such cheapest actions. Thus, $\mathrm{Pref}_\Pi(s)$ can be computed as follows.

$$\mathrm{Pref}_\Pi(s) = \bigcup_{\sigma \in \mathrm{M}(s)} \left[\mathrm{M}(\sigma, s) \cup \bigcup_{v \in V \setminus \{r\}} \tilde{A}_{s[v];|\sigma|}(s[r]) \right]. \tag{4}$$

The only computation that has to be performed per search node, is the final minimization over $\trianglerighteq[\sigma(r|s[r])]$ in Eq. 3 and the union over $\mathrm{M}(s)$ in Eq. 4, and those are the lightest parts of the whole algorithm anyway. The major computations, notably those of $\{p_{\vartheta,\vartheta';\vartheta_r}\}$, $\{A_{\vartheta,\vartheta';\vartheta_r}\}$, $\{\tilde{g}_{\vartheta;i}(\vartheta_r)\}$, and $\{\tilde{A}_{\vartheta;i}(\vartheta_r)\}$, can be performed offline and shared between the evaluated states. The space required to store this information is $O(d^2|A|)$ as it contains only a $O(|A_v|)$ amount of information per pair of values of each variable. The time complexity of the offline computation is $O(d^3|V| + |A_r|)$; the $|A_r|$ component stems from precomputing the costs $\mathit{cost}(\sigma)$. The time complexity of the online computation per state is $O(d|V|)$; $|V|$ comes from the internal summation and d comes from the size of $\trianglerighteq[\sigma(r|s[r])]$. ■

Theorem 2 *Let* $\Pi = \langle V, A, I, G, \mathit{cost}\rangle$ *be a planning task with an inverted fork causal graph with sink* r *and* $|\mathcal{D}(r)| = b = O(1)$. *For any set of states* $S' \subseteq S$, *the time and space complexity of computing the sets* $\mathit{Pref}_\Pi(s)$ *for all states* $s \in S'$ *is, respectively,* $O(b|V||A_r|^{b-1} + d^3|V| + |S'||V||A_r|^{b-1})$ *and* $O(|V||A_r|^{b-1} + d^2|V|)$, *respectively, where* $d = \max_v \mathcal{D}(v)$.

Proof: Similarly to the proof of Theorem 1, the proof of Theorem 2 is by a slight modification of the polynomial-time algorithm for computing $h^*(s)$ used for the proof of Theorem 8 (Tractable Inverted Forks) by Katz and Domshlak (2010b).

(1) For each parent variable $v \in V \setminus \{r\}$, and each pair of its values $\vartheta, \vartheta' \in \mathcal{D}(v)$, let $p_{\vartheta,\vartheta'}$ be the cost of the cheapest sequence of actions changing ϑ to ϑ'. In parallel, let $A_{\vartheta,\vartheta'}$ be the set of all possible first actions of such sequences. Both $\{p_{\vartheta,\vartheta'}\}$ and $\{A_{\vartheta,\vartheta'}\}$ can be computed using the Floyd-Warshall algorithm on the domain transition graph of v in time $O(d^3|V|)$.

(2) For each $\vartheta_r \in \mathcal{D}(r)$ and each cycle-free path $\pi = \langle a_1, \ldots, a_m\rangle$ from ϑ_r to $G[r]$ in $\mathit{DTG}(r, \Pi)$, let $a_\pi = a_1$ be the first action of that path, and let a "proxy" state s_π be

$$s_\pi[v] = \begin{cases} \vartheta_r, & v = r \\ G[v], & v \notin \bigcup_{i=1}^m \mathcal{V}(\mathsf{pre}(a_i)) \\ \mathsf{pre}(a_i)[v], & i = \operatorname{argmin}_j \{v \in \mathcal{V}(\mathsf{pre}(a_j))\} \end{cases},$$

that is, the nontrivial part of s_π captures the first values of $V \setminus \{r\}$ required along π.[1] Given that, let g_π be the cost of the cheapest plan from s_π in Π based on π, and the cheapest paths $\{p_{\vartheta,\vartheta'}\}$ computed in (1). Each g_π can be computed as

$$g_\pi = \sum_{i=1}^{m} \left[cost(a_i) + \sum_{v \in V \setminus \{r\}} p_{\mathsf{pre}_i[v], \mathsf{pre}_{i+1}[v]} \right],$$

where, for each $v \in V \setminus \{r\}$, and $1 \leq i \leq m+1$,

$$\mathsf{pre}_i[v] = \begin{cases} s_\pi[v], & i = 1 \\ G[v], & i = m+1 \text{ and } v \in \mathcal{V}(G) \\ \mathsf{pre}(a_i)[v], & 2 \leq i \leq m \text{ and } v \in \mathcal{V}(\mathsf{pre}(a_i)) \\ \mathsf{pre}_{i-1}[v], & \text{otherwise} \end{cases}$$

Storing the triplets (g_π, s_π, a_π) accomplishes the offline part of the computation.

(3) Now, given a search state s, we can compute

$$h^*(s) = \min_{\substack{\pi \text{ s.t.} \\ s_\pi[r] = s[r]}} \left[g_\pi + \sum_{v \in V \setminus \{r\}} p_{s[v], s_\pi[v]} \right], \quad (5)$$

storing the paths obtaining the minimum. Let $\mathrm{M}(s)$ be the set of such paths, and for each path π, let $\mathrm{M}(s, \pi) = \{a_\pi \mid a_\pi \in A(s)\}$ denote the set containing the first action of such path, if it is applicable in state s. Thus, $\mathrm{Pref}_\Pi(s)$ can be computed as follows.

$$\mathrm{Pref}_\Pi(s) = \bigcup_{\pi \in \mathrm{M}(s)} \left[\mathrm{M}(s, \pi) \cup \bigcup_{v \in V \setminus \{r\}} A_{s[v], s_\pi[v]} \right]. \quad (6)$$

The number of cycle-free paths to $G[r]$ in $\mathit{DTG}(r, \Pi)$ is $\Theta(|A_r|^{b-1})$, and g_π for each such path π can be computed in time $O(b|V|)$. Hence, the overall offline time complexity is $O(b|V||A_r|^{b-1} + d^3|V|)$, and the space complexity (including the storage of the proxy states s_π and the first actions a_π) is $O(|V||A_r|^{b-1} + d^2|A|)$. The time complexity of the online computation per state via Eq. 5 is $O(|V||A_r|^{b-1})$; $|V|$ comes from the internal summation and $|A_r|^{b-1}$ from the upper bound on the number of cycle-free paths from $s[r]$ to $G[r]$. ∎

Experimental Evaluation

We have evaluated satisficing planning with fork-decomposition heuristics, using iterative lazy weighted A^* (Richter & Helmert, 2009) with weight schema 100, 10, 5, 3, 2, 1, with deferred evaluation and preferred operators. This setting was compared to two heuristic-search baselines, both using the same search scheme as ours, and employing the FF heuristic. The first baseline used no preferred operators at all. The second one used preferred operators from the FF heuristic (Hoffmann & Nebel, 2001). All the planners were run on one core of a 2.33GHz Intel Q8200 CPU with 4 GB memory, using 2 GB memory limit and 30 minute timeout. The summary of results is depicted in Table 1. The scoring method evaluates plan costs accordingly to the method used in International Planning Competition (IPC-2008).

Columns 2 and 3 depict the results for searching with the FF heuristic, without and with preferred operators. The evaluation covered three fork-decomposition heuristics, namely forks only ($h^{\mathfrak{F}}$), inverted forks only ($h^{\mathfrak{I}}$); and both forks and inverted forks. ($h^{\mathfrak{FI}}$). In order to overcome various issues caused by the 0-cost actions in IPC 2008 domains, in these domains for the purpose of heuristic evaluation all action costs were increased by 1.

The first evaluation was performed with no preferred operators, and the results are depicted in columns 4, 9, and 14, respectively. Then, preferred operators from these heuristics were added (columns 5, 10, and 15). Our first observation is that preferred operators from the implicit abstractions are not always helpful. In fact, for both $h^{\mathfrak{F}}$ and $h^{\mathfrak{FI}}$, in most of the domains, the plans obtained without any preferred operators are shorter than with all preferred operators obtained from these abstractions. Note that the abstract operators change the value of a variable with either in-degree 0 or out-degree 0. In the latter case the goal value of the variable is always defined, while in sooner it is not always the case. Thus, some of these operators may be more helpful in guidance towards the goal than other. In order to check this hypothesis, we evaluated two additional settings, one taking only preferred operators changing the value of some variable with in-degree 0 in some abstraction (columns 6, 11, and 16), and another one taking only preferred operators changing the value of some variable with out-degree 0 in some abstraction (columns 7, 12, and 17). This setting changes the picture for both $h^{\mathfrak{F}}$ and $h^{\mathfrak{FI}}$ heuristics, yet not for $h^{\mathfrak{I}}$ heuristic where the picture is reverse. In any case, all these settings of preferred operators in most of the domains are outperformed by FF heuristic with its preferred operators. One possible reason for that is that the abstractions do not cover all planning task variables, and usually concentrated around those with goal values defined. Hence, the guidance of the actions obtained from these abstractions is greedy towards achieving the goals, and disregards other variables and intermediate goals.

In order to evaluate this assumption, we evaluated employing the fork-decomposition heuristics with preferred operators derived from the relaxed plans responsible for the FF heuristic values. The respective results are shown in columns 8, 13, and 18. Although the per-node evaluation obviously becomes more expensive, the

[1]For ease of presentation, we omit here the case where v is required neither along π, nor by the goal; such variables should be simply ignored when accounting for the cost of π.

domain	h_{FF}		$h^{\mathcal{F}}$					$h^{\mathcal{I}}$					$h^{\mathcal{FI}}$				
	No Pref	All Pref	No Pref	All Pref	Up Pref	Lo Pref	FF Pref	No Pref	All Pref	Up Pref	Lo Pref	FF Pref	No Pref	All Pref	Up Pref	Lo Pref	FF Pref
blocks-aips2000	**34.81**	34.24	32.56	30.33	30.69	31.34	32.84	31.88	31.32	31.32	32.1	32.67	31.27	29.69	29.91	30.24	32.06
elevators-strips-seq-sat	27.26	**29.32**	11.2	9.87	16.94	14.32	12.74	8.33	8.58	8.58	8.28	13.57	24.29	19.74	20.62	24.81	26.43
logistics-AIPS98	22.55	**32.79**	20.52	14.3	18.53	20.08	28.28	20.15	26.96	18.77	30.88	31.35	20.24	14.18	18.7	21.89	28.49
openstacks-strips-seq-sat	**29.52**	29.27	29.03	25.62	27.94	21.74	29.07	23.55	23.99	23.99	22.73	29.13	29.08	25.02	28.19	21.1	28.96
pegsol-strips-seq-sat	**30**	29.85	29.95	28.86	29	28.77	29	29.95	29	29	29.95	29.95	29.95	28.67	28.75	29.53	29.9
woodworking-strips-seq-sat	12.43	**27.72**	5	6.89	5	6.89	13.11	5	6	6	5	15.67	5	6.97	5	7.74	13.08
logistics-aips2000	27.15	27.76	**27.96**	27.57	27.91	27.26	27.78	26.91	27.33	26.37	26.75	27.29	27.22	27.52	26.58	26.76	27.42
openstacks-adl-seq-sat	29.14	29.18	23.73	20.6	22.29	20.6	**29.22**	13.8	14.33	14.33	13.8	15	25.48	19.19	23.94	19.19	29.15
parcprinter-strips-seq-sat	14	14	12	22.63	20.73	20	22.8	13	26.95	26.95	13	23.93	13	25.97	26.97	26	**28.88**
scanalyzer-strips-seq-sat	24.38	25.15	22.53	21.35	21.81	22.91	21.65	22.36	**25.33**	**25.33**	22.25	22.28	21.43	21.63	21.7	22.98	22.47
sokoban-strips-seq-sat	26.83	26.88	23	22.53	24.98	20.62	23.93	**28.83**	27.73	27.73	**28.83**	27.96	24.75	21.94	23.96	21.89	24.91
transport-strips-seq-sat	12.16	18.29	19.44	14.4	17.44	15.42	**19.67**	8.3	8.34	8.34	8.3	8.94	13.22	11.93	12.61	10.87	17.82
	290.23	**324.45**	256.91	244.95	263.25	249.94	290.08	232.06	255.87	246.72	241.87	277.74	264.93	252.43	266.94	263.02	309.58

Table 1: A summary of the experimental results: plan cost. Per planner/domain, the cost of the best found plan is given by the sum of ratios over all tasks. Boldfaced results indicate the best performance within the corresponding domain. The last row summarizes the ratio over all domains.

domain	h_{FF}		$h^{\mathcal{F}}$					$h^{\mathcal{I}}$					$h^{\mathcal{FI}}$				
	No Pref	All Pref	No Pref	All Pref	Up Pref	Lo Pref	FF Pref	No Pref	All Pref	Up Pref	Lo Pref	FF Pref	No Pref	All Pref	Up Pref	Lo Pref	FF Pref
blocks-aips2000	9.58	**32.52**	5.72	3.97	3.63	8.61	27.06	2.35	1.64	1.64	2.35	15.68	3.9	3.5	2.8	9.87	24.63
elevators-strips-seq-sat	2.05	**25.27**	0.59	0.13	0.29	0.76	10.4	0.04	0.08	0.08	0.04	3.08	1.55	0.49	0.24	1.42	24.63
logistics-AIPS98	2.57	**34.16**	1.31	0.9	0.68	2.31	18.3	0.75	1.6	0.51	7.33	23.28	1.03	1.75	0.5	4.68	21.4
pegsol-strips-seq-sat	15.32	**20.99**	3.62	4.1	4.39	4.05	13.16	3.54	3.77	3.77	3.54	10.75	5.27	3.63	3.77	5.13	13.72
sokoban-strips-seq-sat	**20.41**	12.78	3.24	4.27	4.28	3.78	9.89	7.14	8.03	8.03	7.14	13.89	6.61	8.08	8.1	7.25	11.28
woodworking-strips-seq-sat	2.45	**26.47**	1.36	3.15	2.22	3.97	6.12	0.95	3.46	3.46	1.07	7.9	1.39	3.14	2.27	4.87	6.21
logistics-aips2000	9.37	22.63	13.21	10.66	3.92	15.33	**24.89**	8.19	15.71	3.02	9.77	22.78	8.92	21.09	2.11	9.95	22.89
openstacks-adl-seq-sat	14.04	13.75	4.05	3.16	5.34	3.16	**29.74**	0.58	0.64	0.64	0.58	3.19	2.82	2.55	3.21	2.55	7.41
openstacks-strips-seq-sat	8.53	8.52	18.62	4.09	27.71	5	**28.31**	1.88	5.6	5.6	1.88	6.75	7.09	3.55	13.21	4.52	9.24
parcprinter-strips-seq-sat	2.11	3.59	2.93	7.41	5.29	12.31	9.49	4.06	11.93	11.93	4.06	11.88	4.63	5.4	5.3	**17.86**	13.98
scanalyzer-strips-seq-sat	6.04	11.64	2.62	3.88	3.84	11.49	16.01	2.12	6.78	6.78	2.12	14.03	2.84	4.86	6.64	8.66	**16.57**
transport-strips-seq-sat	1.34	15.17	1.03	0.74	0.99	0.78	13.55	0.52	0.57	0.57	0.52	3.22	1.29	0.96	1.25	0.89	**16.04**
	93.8	**227.48**	58.31	46.44	62.57	71.55	206.93	32.12	59.8	46.02	40.4	136.41	47.34	59	49.41	77.64	188

Table 2: A summary of the experimental results: expanded nodes. Per planner/domain, the number of expanded nodes of the first search is given by the sum of ratios over all tasks. Boldfaced results indicate the best performance within the corresponding domain. The last row summarizes the ratio over all domains.

overall results improve considerably. The approach is still outperformed by the baseline with preferred operators, but the substantial improvement suggests further exploration of this direction.

Another way to evaluate the performance of our approach is to look at the number of expanded nodes. Since the approach runs iteratively, we compared the results obtained for the first iteration only, that is lazy weighted A^* with the weight set to 100. Table 2 describes the results in terms of expanded nodes; with the columns having the same role as before. Overall, the picture here appears similar to this in Table 1. The main difference is that in pairwise comparison, when looking on the number of domains in which fork-decomposition heuristics achieve better performance, these heuristics appear to be slightly more competitive in terms of expanded nodes of the first iteration than in terms of plan cost of the last one.

Summary

We considered heuristic search for sequential satisficing planning and introduced a way of obtaining a set of operators from fork-decomposition implicit abstractions, to be used as preferred operators. We then discussed possible implications of such sets, and performed an empirical evaluation of several settings. We show that even a most conservative setting significantly improves performance comparatively to using no preferred operators at all. In addition, we show that combining heuristic evaluation from one heuristic family and preferred operators from another may improve the overall performance despite the overhead in computing two heuristic functions per search node.

References

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

Helmert, M. (2006). *Solving Planning Tasks in Theory and Practice*. Ph.D. thesis, Albert-Ludwigs University, Freiburg.

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

Katz, M., & Domshlak, C. (2010a). Optimal admissible composition of abstraction heuristics. *Artificial Intelligence*, *174*, 767–798.

Katz, M., & Domshlak, C. (2010b). Implicit abstraction heuristics. *Journal of Artificial Intelligence Research*, *39*, 51 – 126.

Richter, S., & Helmert, M. (2009). Preferred operators and deferred evaluation in satiscing planning. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 273–280, Thessaloniki, Greece.

Wehrle, M., Kupferschmid, S., & Podelski, A. (2008). Useless actions are useful. In *In Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS 2008*, pp. 388–395. AAAI Press.

Cost Based Satisficing Search Considered Harmful

William Cushing and **J. Benton** and **Subbarao Kambhampati**[*]
Dept. of Comp. Sci. and Eng.
Arizona State University
Tempe, AZ 85281

Abstract

Recently, several researchers have found that cost-based satisficing search with A* often runs into problems. Although some "work arounds" have been proposed to ameliorate the problem, there has not been any concerted effort to pinpoint its origin. In this paper, we argue that the origins can be traced back to the wide variance in action costs that is easily observed in planning domains. We show that such cost variance misleads A* search, and that this is a systemic weakness of the very concept: "cost-based evaluation functions + systematic search + combinatorial graphs". We argue that purely size-based evaluation functions are a reasonable default, as these are trivially immune to cost-induced difficulties. We further show that cost-sensitive versions of size-based evaluation function — where the heuristic estimates the size of cheap paths provides attractive quality vs. speed tradeoffs.

1 Introduction

Much of the scale-up, as well as the research focus, in the automated planning community in the recent years has been on satisficing planning. Unfortunately, there hasn't been a concomitant increase in our understanding of satisficing search. Too often, the "theory" of satisficing search defaults to doing (W)A* with inadmissible heuristics. While removing the requirement of admissible heuristics certainly relaxes the guarantee of optimality, there is no implied guarantee of efficiency. A combinatorial search can be seen to consist of two parts: a "discovery" part where the (optimal) solution is found and a "proof" part where the optimality of the solution is verified. While an optimizing search depends crucially on both these phases, a satisficing search is instead affected more directly by the discovery phase. Now, standard A* search conflates the discovery and proof phases together and terminates only when it picks the optimal path for expansion. By default, satisficing planners use the same search regime, but relax the admissibility requirement on the heuristics.[1] This may not cause too much of a problem in domains with uniform action costs, but when actions can have non-uniform costs, the the optimal and second optimal solution can be arbitrarily far apart in depth. Consequently, (W)A* search with cost-based evaluation functions can be an arbitrarily bad strategy for satisficing search, as it waits until the solution is both discovered *and* proved to be (within some bound of) optimal.

To be more specific, consider a planning problem for which the cost-optimal and second-best solution to a problem exist on 10 and 1000 unspecified actions. *The optimal solution may be the larger one.* How long should it take just to find the 10 action plan? How long should it take to prove (or disprove) its optimality? In general (presuming PSPACE/EXPSPACE $\neq$ P):

1. Discovery should require time exponential in, at most, 10.
2. Proof should require time exponential in, at least, 1000.

That is, in principle, the only way to (domain-independently) prove that the 10 action plan is better or worse than the 1000 action one is to in fact go and discover the 1000 action plan. Thus, A* search with cost-based evaluation function will take time proportional to b^{1000} for either discovery or proof. Simple breadth-first search discovers a solution in time proportional to b^{10} (and proof in $O(b^{1000})$).

Using both abstract and benchmark problems, we will demonstrate that this is a systematic weakness of any search that uses cost-based evaluation function. In particular, we shall see that if ε is the smallest cost action (after all costs are normalized so the maximal cost action costs 1 unit), then the time taken to discover a depth d optimal solution will be $b^{\frac{d}{\varepsilon}}$. If all actions have same cost, then $\varepsilon \approx 1$ where as if the actions have significant cost variance, then $\varepsilon \ll 1$. We shall see that for a variety of reasons, most real-world planning domains do exhibit high cost variance, thus presenting an "*ε-cost trap*" that forces any cost-based satisficing search to dig its own ($\frac{1}{\varepsilon}$ deep) grave.

Consequently, we argue that satisficing search should resist the temptation to directly use cost-based evaluation functions (*i.e.*, f functions that return answers in cost units) even if they are interested in the quality (cost measure) of the resulting plan. We will consider two size-based branch-and-bound alternatives: the straightforward one which completely ignores costs and sticks to a purely size-based evaluation function, and a more subtle one that uses a cost-sensitive size-based evaluation function (specifically, the heuristic estimates the size of the cheapest cost path; see Section 2). We show that both of these outperform cost-based evaluation

[*]An extended abstract of this paper appeared in the proceedings of SOCS 2010. This research is supported in part by ONR grants N00014-09-1-0017 and N00014-07-1-1049, the NSF grant IIS-0905672, and by DARPA and the U.S. Army Research Laboratory under contract W911NF-11-C-0037.

[1]In the extreme case, by using an infinite heuristic weight: "greedy best-first search".

functions in the presence of ε-cost traps, with the second one providing better quality plans (for the same run time limits) than the first in our empirical studies.

While some of the problems with cost-based satisficing search have also been observed, in passing, by other researchers (*e.g.*, (Benton et al. 2010; Richter and Westphal 2010), and some work-arounds have been suggested, our main contribution is to bring to the fore its fundamental nature. The rest of the paper is organized as follows. In the next section, we present some preliminary notation to formally specify cost-based, size-based as well as cost-sensitive size-based search alternatives. Next, we present two abstract and fundamental search spaces, which demonstrate that *cost-based* evaluation functions are 'always' needlessly prone to such traps (Section 3). Section 4 strengthens the intuitions behind this analysis by viewing best-first search as flooding topological surfaces set up by evaluation functions. We will argue that of all possible topological surfaces (*i.e.*, evaluation functions) to choose for search, cost-based is the worst. In Section 5, we put all this analysis to empirical validation by experimenting with LAMA (Richter and Westphal 2010) and SapaReplan. The experiments do show that size-based alternatives can out-perform cost-based search. Modern planners such as LAMA use a plethora of improvements beyond vanilla A* search, and in the appendix we provide a deeper analysis on which extensions of LAMA seem to help it mask (but not fully overcome) the pernicious effects of cost-based evaluation functions.

2 Setup and Notation

We gear the problem set up to be in line with the prevalent view of state-space search in modern, state-of-the-art satisficing planners. First, we assume the current popular approach of reducing planning to graph search. That is, planners typically model the state-space in a causal direction, so the problem becomes one of extracting paths, meaning whole plans do not need to be stored in each search node. More important is that the structure of the graph is given *implicitly* by a procedure Γ, the child generator, with $\Gamma(v)$ returning the local subgraph leaving v; i.e., $\Gamma(v)$ computes the subgraph $(N^+[v], E(\{v\}, V-v)) = (\{u \mid (v,u) \in E\} + v, \{(v,u) \mid (v,u) \in E\})$ along with all associated labels, weights, and so forth. That is, our analysis depends on the assumption that *an implicit representation of the graph is the only computationally feasible representation*, a common requirement for analyzing the A* family of algorithms (Hart, Nilsson, and Raphael 1968; Dechter and Pearl 1985).

The search problem is to find a path from an initial state, i, to some goal state in $\mathcal{G}$. Let costs be represented as edge weights, say $c(e)$ is the cost of the edge e. Let $g_c^*(v)$ be the (optimal) cost-to-reach v (from i), and $h_c^*(v)$ be the (optimal) cost-to-go from v (to the goal). Then $f_c^*(v) := g_c^*(v) + h_c^*(v)$, the cost-through v, is the cost of the cheapest i-$\mathcal{G}$ path passing through v. For discussing smallest solutions, let $f_s^*(v)$ denote the smallest i-$\mathcal{G}$ path through v. It is also interesting to consider the size of the cheapest i-$\mathcal{G}$ path passing through v, say $\hat{f}_s^*(v)$.

We define a search node n as equivalent to a path represented as a linked list (of edges). In particular, we distinguish this from the state of n (its last vertex), $n\,.\,v$. We say $n\,.\,a$ (for action) is the last edge of the path and $n.\,p$ (for parent) is the subpath excluding $n\,.\,a$. Let $n' = na$ denote extending n by the edge a (so $a = (n.\,v, n'.\,v)$). The function $g_c(n)$ (g-cost) is just the recursive formulation of path cost: $g_c(n) := g_c(n\,.\,p) + c(n\,.\,a)$ ($g_c(n) := 0$ if n is the trivial path). So $g_c^*(v) \le g_c(n)$ for all i-v paths n, with equality for at least one of them. Similarly let $g_s(n) := g_s(n.\,p) + 1$ (initialized at 0), so that $g_s(n)$ is an upper bound on the shortest path reaching the same state ($n\,.\,v$).

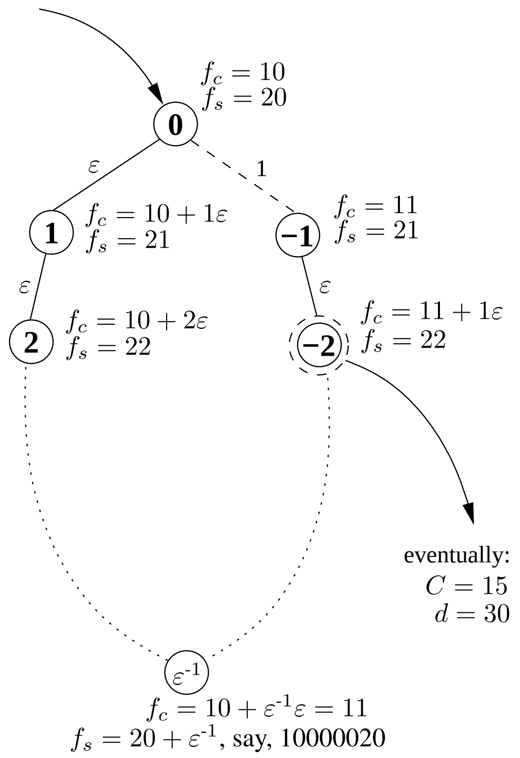
A goal state is a target vertex where a plan may stop and be a valid solution. We fix a computed predicate $\mathcal{G}(v)$ (a black-box), the *goal*, encoding the set of goal states. Let $h_c(v)$, the *heuristic*, be a procedure to estimate $h_c^*(v)$. (Sometimes h_c is considered a function of the search node, *i.e.*, the whole path, rather than just the last vertex.) The heuristic h_c is called *admissible* if it is a *guaranteed* lower bound. (An inadmissible heuristic lacks the guarantee, but might anyways be coincidentally admissible.) Let $h_s(v)$ estimate the remaining depth to the nearest goal, and let $\hat{h}_s(v)$ estimate the remaining depth to the cheapest reachable goal. Anything goes with such heuristics — an inadmissible estimate of the size of an inadmissible estimate of the cheapest continuation is an acceptable (and practical) interpretation of $\hat{h}_s(v)$.

We focus on two different definitions of f (the evaluation function). Since we study cost-based planning, we consider $f_c(n) := g_c(n) + h_c(n\,.\,v)$; this is the (standard, cost-based) *evaluation function* of A^*: cheapest-completion-first. We compare this to $f_s(n) := g_s(n) + h_s(n\,.\,v)$, the canonical size-based (or search distance) evaluation function, equivalent to f_c under uniform weights. Any combination of g_c and h_c is *cost-based*; any combination of g_s and h_s is *size-based* (*e.g.*, breadth-first search is size-based). The evaluation function $\hat{f}_s(n) := g_s(n) + \hat{h}_s(n\,.\,v)$ is also size-based, but nonetheless cost-sensitive and so preferable.

```
BEST-FIRST-SEARCH($i, \mathcal{G}, \Gamma, h_c$)
 1  initialize search
 2  **while** *open* not empty
 3      $n$ = *open*.remove()
 4      **if** BOUND-TEST($n, h_c$) **then continue**
 5      **if** GOAL-TEST($n, \mathcal{G}$) **then continue**
 6      **if** DUPLICATE-TEST($n$) **then continue**
 7      $s = n\,.\,v$
 8      *star* = $\Gamma(s)$                                   // Expand $s$
 9      **for** each edge $a = (s, s')$ from $s$ to a child $s'$ in *star*
10          $n' = na$
11          $\boxed{f = \text{EVALUATE}(n')}$
12          *open*.add($n', f$)
13  **return** *best-known-plan*                 // Optimality is proven.
```

Pseudo-code for best-first branch-and-bound search of implicit graphs is shown above. It continues searching after a solution is encountered and uses the current best solution value to prune the search space (line 4). The search is performed on a graph implicitly represented by Γ, with the assumption being that the explicit graph is so large that it is better to invoke expensive heuristics (inside of EVALUATE) during the search than it is to just compute the graph up front. The question considered by this paper is how to implement EVALUATE.

With respect to normalizing costs, we can let ε :=



$f_c = 10$
$f_s = 20$
0
ε
1
1
$f_c = 10 + 1\varepsilon$
$f_s = 21$
-1
$f_c = 11$
$f_s = 21$
ε
ε
2
$f_c = 10 + 2\varepsilon$
$f_s = 22$
-2
$f_c = 11 + 1\varepsilon$
$f_s = 22$
eventually:
$C = 15$
$d = 30$
ε^{-1}
$f_c = 10 + \varepsilon^{-1}\varepsilon = 11$
$f_s = 20 + \varepsilon^{-1}$, say, 10000020

Figure 1: A trap for cost-based search. The heuristic perceives all movement on the cycle to be irrelevant to achieving high quality plans. The state with label **-2** is one interesting way to leave the cycle, there may be (many) others. C denotes the cost of one such continuation from **-2**, and d its depth. Edge weights nominally denote changes in f_c: as given, locally, these are the same as changes in g_c. But increasing f_s by 1 at **-1** (and descendants) would, for example, model instead the special edge as having cost $\frac{1}{2}$ and being perceived as worst-possible in an undirected graph.

$\frac{\min_a c(a)}{\max_a c(a)}$, that is, ε is the least cost edge after normalizing costs by the maximum cost (to bring costs into the range $[0, 1]$). We use the symbol ε for this ratio as we anticipate actions with high cost variance in real world planning problems. For example: boarding versus flying (ZenoTravel), mode-switching versus machine operation (Job-Shop), and (unskilled) labor versus (precious) material cost.

3 ε-cost Trap: Two Canonical Cases

In this section we argue that the mere presence of ε-cost misleads cost-based search, and that this is no trifling detail or accidental phenomenon, but a systemic weakness of the very concept of "cost-based evaluation functions + systematic search + combinatorial graphs". We base this analysis in two abstract search spaces, in order to demonstrate the fundamental nature of such traps. The first abstract space we consider is the simplest, non-trivial, non-uniform cost, intractably large, search space: the search space of an enormous cycle with one expensive edge. The second abstract space we consider is a more natural model of search (in planning): a uniform branching tree. Traps in these spaces are just exponentially sized *and connected* sets of ε-cost edges: not the common result of, say, a typical random model of search. We briefly consider why planning benchmarks naturally give rise to such structure.

For a thorough analysis of models of search see (Pearl 1984); for a planning specific context see (Helmert and Röger 2008).

3.1 Cycle Trap

In this section we consider the simplest abstract example of the ε-cost 'trap'. The notion is that applying increasingly powerful heuristics, domain analysis, learning techniques, . . . , to one's search problem transforms it into a simpler 'effective graph' — the graph for which Dijkstra's algorithm (Dijkstra 1959) produces isomorphic behavior. For example, let c' be a new edge-cost function obtained by setting edge costs to the difference in f values of the edge's endpoints: Dijkstra's algorithm on c' is A* on f.[2] Similarly take Γ' to be the result of applying one's favorite incompleteness-inducing pruning rules to Γ (the child generator), say, helpful actions (Hoffmann and Nebel 2001); then Dijkstra's algorithm on Γ' is A* with helpful action pruning.

Presumably the effective search graph remains very complex despite all the clever inference (or there is nothing to discuss); but certainly complex graphs contain simple graphs as subgraphs. So if there is a problem with search behavior in an exceedingly simple graph then we can suppose that no amount of domain analysis, learning, heuristics, and so forth, will incidentally address the problem: such inference must specifically address the issue of non-uniform weights. Suppose not: none of the bells and whistles consider non-uniform costs to be a serious issue, permitting wildly varying edge 'costs' even in the effective search graph: $\varepsilon \approx \varepsilon' = \frac{\min_e c'(e)}{\max_e c'(e)}$. We demonstrate that that by itself is enough to produce very troubling search behavior: ε-cost is a fundamental challenge to be overcome in planning.

There are several candidates for simple non-trivial state-spaces (*e.g.*, cliques), but clearly the cycle is fundamental (what kind of 'state-space' is acyclic?). So, the state-space we consider is the cycle, with associated exceedingly simple metric consisting of all uniform weights but for a single expensive edge. Its search space is certainly the simplest non-trivial search space: the rooted tree on two leaves. So the single unforced decision to be made is in which direction to traverse the cycle: clockwise or counter-clockwise. See Figure 1. Formally:

ε-cost Trap: Consider the problem of making some variable, say x, encoded in k bits represent $2^k - 2 \equiv -2 \pmod{2^k}$, starting from 0, using only the operations of increment and decrement. There are 2 minimal solutions: incrementing $2^k - 2$ times, or decrementing twice. Set the cost of incrementing and decrementing to 1, except for transitioning between $x \equiv 0$ and $x \equiv -1$ costs, say, 2^{k-1} (in either direction). Then the 2 minimal solutions cost $2^k - 2$ and $2^{k-1} + 1$, or, normalized, $2(1 - \varepsilon)$ and $1 + \varepsilon$. Cost-based search loses: While both approaches prove optimality in exponential time ($O(2^k)$), size-based search discovered that optimal plan in constant time.

[2]Systematic inconsistency of a heuristic translates to analyzing the behavior of Dijkstra's algorithm with many *negative* 'cost' edges, a typical reason to assume consistency in analysis.

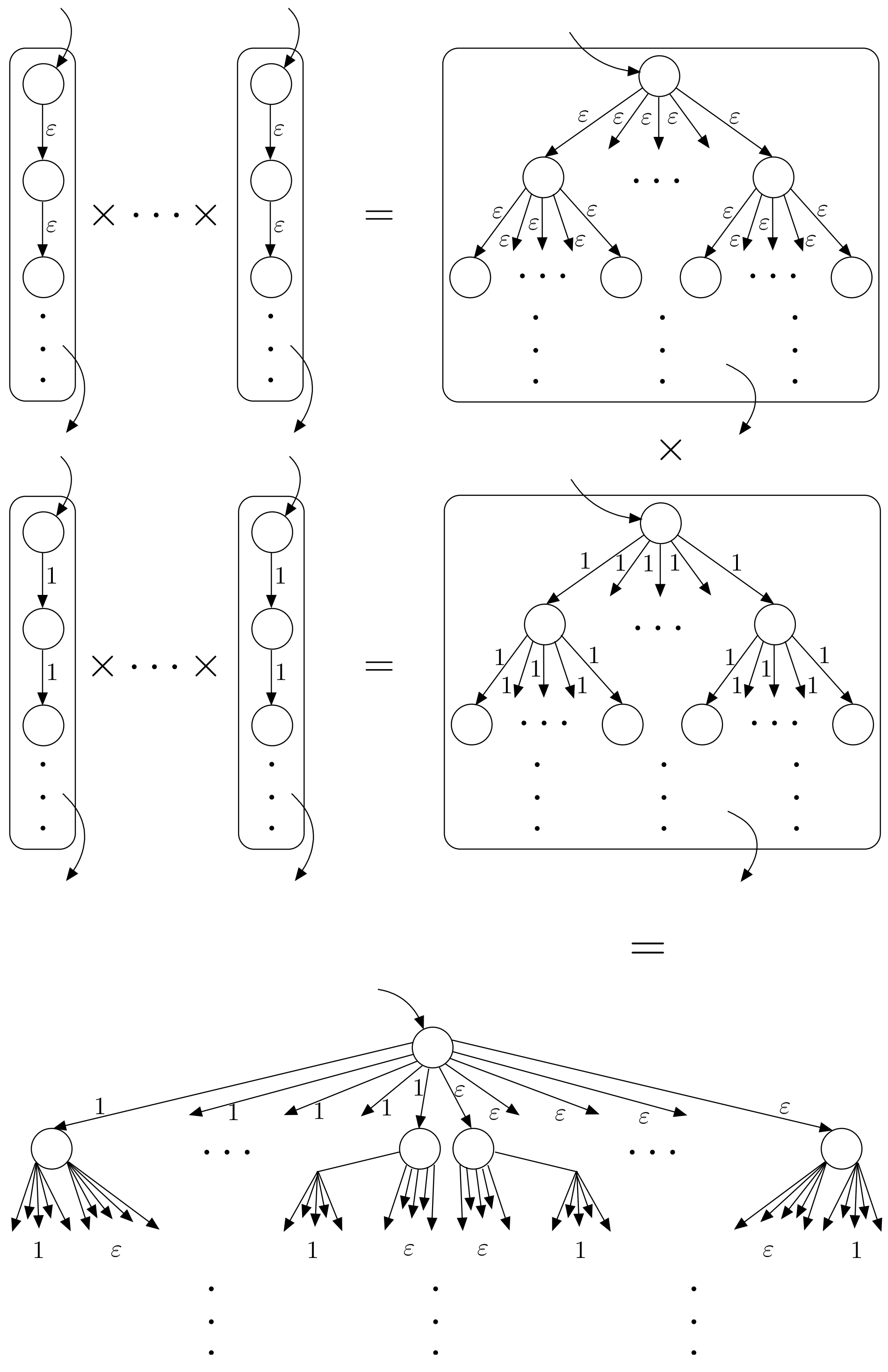


Figure 2: A trap for cost-based search. Two rather distinct kinds of physical objects exist in the domain, with primitive operators at rather distinct orders of magnitude; supposing uniformity and normalizing, then one type involves ε-cost and the other involves cost 1. So there is a low-cost subspace, a high-cost subspace, and the full space, each a uniform branching tree. As trees are acyclic, it is probably best to think of these as search, rather than state, spaces. As depicted, planning for an individual object is trivial as there is no choice besides going forward. Other than that no significant amount of inference is being assumed, and in particular the effects of a heuristic are not depicted. For cost-based search to avoid death, the heuristic would need to forecast every necessary cost 1 edge, so as to reduce its weight closer to 0. (Note that the aim of a heuristic is to drive all the weights to 0 along optimal/good paths, and to infinity for not-good/terrible/dead-end choices.) If any cut of the space across such edges (separating good solutions) is not foreseen, then backtracking into *all* of the low-cost subspaces so far encountered commences, to multiples of depth ε^{-1} — one such multiple for every unforeseen cost 1 cut. Observe that in the object-specific subspaces (the paths), a single edge ends up being multiplied into such a cut of the global space.

Performance Comparison: All Goals. Of course the goal $x \equiv -2$ is chosen to best illustrate the trap. So consider the discovery problem for other goals. With the goal in the interval $2^k \cdot [0, \frac{1}{2}]$ cost-based search is twice as fast. With the goal in the interval $2^k \cdot [\frac{1}{2}, \frac{2}{3}]$ the performance gap narrows to break-even. For the last interval, $2^k \cdot [\frac{2}{3}, 1\rangle$, the size-based approach takes the lead — *by an enormous margin*. There is one additional region of interest. Taking the goal in the interval $2^k \cdot [\frac{2}{3}, \frac{3}{4}]$ there is a trade-off: size-based search finds a solution before cost-based search, but cost-based search finds the optimal solution first. Concerning time till optimality is proven, the cost-based approach is monotonically faster (of course). Specifically, the cost-based approach is faster by a factor of 2 for goals in the region $2^k \cdot [0, \frac{1}{2}]$, not faster for goals in the region $2^k \cdot [\frac{3}{4}, 1\rangle$, and by a factor of $(\frac{1}{2} + 2\alpha)^{-1}$ (bounded by 1 and 2) for goals of the form $x \equiv 2^k(\frac{1}{2} + \alpha)$, with $0 < \alpha < \frac{1}{4}$.

Performance Comparison: Feasible Goals. Considering all goals is inappropriate in the satisficing context; to illustrate, consider large k, say, $k = 1000$. Fractions of exponentials are still exponentials — even the most patient reader will have forcibly terminated either search *long* before receiving any useful output. Except if the goal is of the form $x \equiv 0 \pm f(k)$ for some sub-exponential $f(k)$. Both approaches discover (and prove) the optimal solution in the positive case in time $O(f(k))$ (with size-based performing twice as much work). In the negative case, only the size-based approach manages to discover a solution (the optimal one, in time $O(f(k))$) before being killed. Moreover, while it will fail to produce a proof of such before death, we, based on superior understanding of the domain, can, and have, shown it to be posthumously correct. ($2^k - f(k) > 2^k \cdot \frac{3}{4}$ for any sub-exponential $f(k)$ with large enough k.)

How Good is Almost Perfect Search Control? Keep in mind that the representation of the space as a simple k bit counter is *not* available. In particular what 'increment' actually stands for is an inference-motivated choice of a single operator out of a large number of executable and promising operators at each state — in the language of Markov Decision Processes, we are allowing inference to be so close to perfect that the optimal policy is known at all but 1 state. Only one decision remains ... but no methods cleverer than search remain. Still the graph is intractably large. Cost-based search only explores in one direction: left, say. In the satisficing context such behavior is entirely inappropriate. What is appropriate? Of course explore left first, for considerable time even. But certainly not for 3 years before even trying just a few expansions to the right, for that matter, even mere thousands of expansions to the left before one or two to the right are tried is perhaps too greedy.

3.2 Branching Trap

In the counter problem the trap is not even *combinatorial*; the search problem consists of a single decision at the root, and the trap is just an exponentially deep path. Then it is abundantly clear that appending Towers of Hanoi to a planning benchmark, setting its actions at ε-cost, will kill cost-based search — even given the perfect heuristic for the puzzle! Besides Hanoi, though, exponentially deep paths are not typical of planning benchmarks. So in this section we demonstrate that exponentially large subtrees on ε-cost edges are also traps.

Consider $x > 1$ high cost actions and $y > 1$ low cost actions in a uniform branching tree model of search space. The model is appropriate up to the point where duplicate state checking becomes significant. (See Figure 2.) Suppose the solution of interest costs C, in normalized units, so the solution lies at depth C or greater. Then cost-based search faces a grave situation: $O((x + y^{\frac{1}{\varepsilon}})^C)$ possibilities will be explored before considering all potential solutions of cost C.

A size-based search only ever considers at most $O((x + y)^d) = O(b^d)$ possibilities before consideration of all potential solutions of size d. Of course the more interesting question is how long it takes to find solutions of fixed cost rather than fixed depth. Note that $\frac{C}{\varepsilon} \geq d \geq C$. Assuming the high cost actions are relevant, that is, some number of them are needed by solutions, then we have that solutions are not actually hidden as deep as $\frac{C}{\varepsilon}$. Suppose, for example, that solutions tend to be a mix of high and low cost actions in equal proportion. Then the depth of those solutions with cost C is $d = 2\,\frac{C}{1+\varepsilon}$ (*i.e.*, $\frac{d}{2} \cdot 1 + \frac{d}{2} \cdot \varepsilon = C$). At such depths the size-based approach is the clear winner: $O((x + y)^{\frac{2C}{1+\varepsilon}}) \ll O((x + y^{\frac{1}{\varepsilon}})^C)$ (normally).

Consider, say, $x = y = \frac{b}{2}$, then:

$$\begin{aligned} &\text{size effort/cost effort} \approx \\ &\quad b^{\frac{2C}{1+\varepsilon}} \big/ \left(x + y^{\frac{1}{\varepsilon}}\right)^C < b^{\frac{2C}{1+\varepsilon}} / y^{\frac{C}{\varepsilon}}, \\ &\qquad\qquad < 2^{\frac{C}{\varepsilon}} / b^{\frac{C}{\varepsilon}\frac{1-\varepsilon}{1+\varepsilon}}, \\ &\qquad\qquad < \left(\frac{2}{b^{\frac{1-\varepsilon}{1+\varepsilon}}}\right)^{\frac{C}{\varepsilon}}, \end{aligned}$$

and, provided $\varepsilon < \frac{1-\log_b 2}{1+\log_b 2}$ (for $b = 4$, $\varepsilon < \frac{1}{3}$), the last is always less than 1 and, for that matter, goes, quickly, to 0 as C increases and/or b increases and/or ε decreases.

Generalizing, the size-based approach is faster at finding solutions of any given cost, as long as (1) high-cost actions constitute at least some constant fraction of the solutions considered (high-cost actions are relevant), (2) the ratio between high-cost and low-cost is sufficiently large, (3) the effective search graph (post inference) is reasonably well modeled by an infinite uniform branching tree (*i.e.*, huge enough to render duplicate checking negligible, or at least not especially favorable to cost-based search), and most importantly, (4) the cost function in the effective search graph *still* demonstrates a sufficiently large ratio between high-cost and low-cost edges (no inference has attempted to compensate).

4 Search Effort as Flooding Topological Surfaces of Evaluation Functions

We view evaluation functions (f) as topological surfaces over search nodes, so that generated nodes are visited in, roughly, order of f-altitude. With non-monotone evaluation functions, the set of nodes visited before a given node is all those contained within some basin of the appropriate depth — picture water flowing from the initial state: if there are dams then such a flood could temporarily visit high altitude nodes before low altitude nodes. (With very inconsistent heuristics — large heuristic weights — the metaphor loses explanatory power, as there is nowhere to go but downhill.)

All reasonable choices of search topology will eventually lead to identifying and proving the optimal solution (*e.g.*, assume finiteness, or divergence of f along infinite paths). Some will produce a whole slew of suboptimal solutions along the way, eventually reaching a point where one begins to wonder if the most recently reported solution is optimal. Others report nothing until finishing. The former are *interruptible* (Zilberstein 1998), which is one way to more formally define satisficing.[3] Admissible cost-based topology is the least interruptible choice: the only reported solution is also the last path considered. Define the cost-optimal footprint as the set of plans considered. Gaining interruptibility is a matter of raising the altitude of large portions of the cost-optimal footprint in exchange for lowering the altitude of a smaller set of non-footprint search nodes — allowing suboptimal solutions to be considered. Note that interruptibility comes at the expense of total work.

So, somewhat confirming the intuition that interruptibility is a reasonable notion of satisficing: the cost-optimal approach is the worst-possible approach (short of deliberately wasting computation) to satisficing. Said another way, proving optimality is about increasing the lower bound on true value, while solution discovery is about decreasing the upper bound on true value. It seems appropriate to assume that the fastest way to decrease the upper bound is more or less the opposite of the fastest way to increase the lower bound — with the notable exception of the very last computation one will ever do for the problem: making the two bounds meet (proving optimality).

For size-based topology, with respect to any cost-based variant, the 'large' set is the set of *longer* yet *cheaper* plans, while the 'small' set is the *shorter* yet *costlier* plans. In general one expects there to be many more longer plans than shorter plans in combinatorial problems, so that the increase in total work is small, relative to the work that had to be done eventually (exhaust the many long, cheap, plans). The additional work is considering exactly plans that are costlier than necessary (potentially suboptimal solutions). So the idea of the trade-off is good, but even the best version of a purely size-based topology will not be the best trade-off possible — intuitively the search shouldn't be completely blind to costs: just defensive about possible ε-traps.

Actually, there is a common misconception here about *search nodes* and *states* due to considering uniformly branching trees as a model of state-space: putting states and search nodes in correspondence. Duplicate detection and re-expansion are, in practice, important issues. In particular not finding the cheapest path first comes with a price, re-expansion, so the satisficing intent comes hand in hand with re-expansion of states. So, for example, besides the obvious kind of re-expansion that IDA* (Korf 1985) performs between iterations, it is also true that it considers paths which A* never would (even subsequent to arming IDA* with a transposition table) — it is not really true that one can re-order consideration of paths however one pleases. In particular at least some kind of breadth-first bias is appropriate, so as to avoid finding woefully suboptimal plans to states early on, triggering giant cascades of re-expansion later on.

[3]Another way that Zilberstein suggests is to specify a *contract*; the 2008 planning competition has such a format (Helmert, Do, and Refanidis 2008).

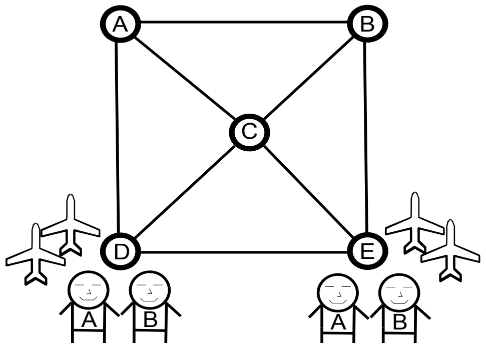


Figure 3: Rendezvous problems. Diagonal edges cost 7,000, exterior edges cost 10,000. Board/Debark cost 1.

For thorough consideration of blending size and cost considerations in the design of evaluation functions see (Thayer and Ruml 2010). Earlier work in evaluation function design beyond just simplistic heuristic weighting is in (Pohl 1973). Dechter and Pearl give a highly technical account of the properties of generalized best-first search strategies, focusing on issues of computational optimality, but, mostly from the perspective of search constrained to proving optimality in the path metric (Dechter and Pearl 1985).

5 ε-cost Trap in Practice

In this section we demonstrate existence of the problematic planner behavior in a realistic setting: running LAMA on problems in the travel domain (simplified ZenoTravel, zoom and fuel removed), as well as two other IPC domains. Analysis of LAMA is complicated by many factors, so we also test the behavior of SapaReplan on simpler instances (but in all of ZenoTravel). The first set of problems concern a rendezvous at the center city in the location graph depicted in Figure 3; the optimal plan arranges a rendezvous at the center city. The second set of problems is to swap the positions of passengers located at the endpoints of a chain of cities.

For thorough empirical analysis of cost issues in standard benchmarks see (Richter and Westphal 2010).

5.1 LAMA

In this section we demonstrate the performance problem wrought by ε-cost in a state-of-the-art (2008) planner — LAMA (Richter and Westphal 2010), the leader of the cost-sensitive (satisficing) track of IPC'08 (Helmert, Do, and Refanidis 2008). With a completely trivial recompilation (set a flag) one can make it ignore the given cost function, effectively searching by f_s. With slightly more work one can do better and have it use $\hat{f}_s$ as its evaluation function, *i.e.*, have the heuristic estimate $\hat{d}$ and the search be size-based, but still compute costs correctly for branch-and-bound. Call this latter modification LAMA-size. Ultimately, the observation is that LAMA-size outperforms LAMA — an astonishing feat for such a trivial change in implementation.

Domain	LAMA	LAMA-size
Rendezvous	70.8%	83.0%
Elevators	79.2%	93.6%
Woodworking	76.6%	64.1%

Table 1: IPC metric on LAMA variants.

LAMA[4] defies analysis in a number of ways: *landmarks*, *preferred operators*, *dynamic evaluation functions*, *multiple open lists*, and *delayed evaluation*, all of which effect potential search plateaus in complex ways. Nonetheless, it is essentially a cost-based approach.

Results.[5] With more than about 8 total passengers, LAMA is unable to complete any search stage except the first (the greedy search). For the same problems, LAMA-size finds the same first plan (the heuristic values differ, but not the structure), but is then subsequently able to complete further stages of search. In so doing it sees marked improvement in cost; on the larger problems this is due only to finding better variants on the greedy plan. Other domains are included for broader perspective, woodworking in particular was chosen as a likely counter-example, as all the actions concern just one type of physical object and the costs are not wildly different. For the same reasons we would expect LAMA to out-perform LAMA-size in some cost-enhanced version of Blocksworld.

5.2 SapaReplan

We also consider the behavior of SapaReplan on the simpler set of problems.[6] This planner is much less sophisticated in terms of its search than LAMA, in the sense of being much closer to a straight up implementation of weighted A* search. The problem is just to swap the locations of passengers located on either side of a chain of cities. A plane starts on each side, but there is no actual advantage to using more than one (for optimizing either of size or cost): the second plane exists to confuse the planner. Observe that smallest and cheapest plans are the same. So in some sense the concepts have become only superficially different; but this is just what makes the problem interesting, as despite this similarity, still the behavior of search is strongly affected by the nature of the evaluation function. We test the performance of $\hat{f}_s$ and f_c, as well as a hybrid evaluation function similar to $\hat{f}_s + f_c$ (with costs normalized). We also test hybridizing via tie-breaking conditions, which ought to have little effect given the rest of the search framework.

Results.[7] The size-based evaluation functions find better cost plans faster (within the deadline) than cost-based evaluation functions. The hybrid evaluation function also does

	2 Cities		3 Cities	
Mode	Score	Rank	Score	Rank
Hybrid	88.8%	1	43.1%	2
Size	83.4%	2	43.7%	1
Size, tie-break on cost	82.1%	3	43.1%	2
Cost, tie-break on size	77.8%	4	33.3%	3
Cost	77.8%	4	33.3%	3

Table 2: IPC metric on SapaReplan variants in ZenoTravel.

relatively well, but not as well as could be hoped. Tie-breaking has little effect, sometimes negative.

We note that Richter and Westphal (2010) also report that replacing cost-based evaluation function with a pure size-based one improves performance over LAMA in multiple other domains. Our version of LAMA-size uses a cost-sensitive size-based search ($\hat{h}_s$), and our results, in the domains we investigated, seem to show bigger improvements over the size-based variation on LAMA obtained by *completely* ignoring costs (h_s, *i.e.*, setting the compilation flag). Also observe that one need not accept a tradeoff: calculating $\log_{10} \varepsilon^{-1} \leq 2$ (3? 1.5?) and choosing between LAMA and LAMA-size appropriately would be an easy way to improve performance simultaneously in ZenoTravel (4 orders of magnitude) and Woodworking ($<$ 2 orders of magnitude).

Finally, while LAMA-size outperforms LAMA, our theory of ε-cost traps suggests that cost-based search should fail even more spectacularly. In the appendix, we take a much closer look at the travel domain and present a detailed study of which extensions of LAMA help it temporarily mask the pernicious effects of cost-based search. Our conclusion is that both LAMA and SapaReplan manage to find solutions to problems in the travel domain despite the use of a cost-based evaluation function by using various tricks to induce a limited amount of *depth-first behavior* in an A*-framework. This has the potential effect of delaying exploration of the ε-cost plateaus slightly, past the discovery of a solution, but still each planner is ultimately trapped by such plateaus before being able to find really good solutions. In other words, such tricks are mostly serving to mask the problems of cost-based search (and ε-cost), as they merely delay failure by just enough that one can imagine that the planner is now effective (because it returns a solution where before it returned none). Using a size-based evaluation function more directly addresses the existence of cost plateaus, and not surprisingly leads to improvement over the equivalent cost-based approach — even with LAMA.

6 Conclusion

The practice of combinatorial search in automated planning is (often) *satisficing*. There is a great call for deeper theories of satisficing search (*e.g.,* a formal definition agreeing with practice is a start), and one perhaps significant obstacle in the way of such research is the pervasive notion that perfect problem solvers are the ones giving only perfect solutions. Actually implementing cost-based, systematic, combinatorial, search reinforces this notion, and therein lies its greatest harm. (Simply defining search as if "strictly positive edge weights" is good enough in practice is also harmful.)

In support of the position we demonstrated the technical difficulties arising from such use of a cost-based evaluation

[4]Options: 'fFlLi'.

[5]New best plans for Elevators were found (largely by LAMA-size). The baseline planner's score is 71.8% against the better reference plans.

[6]Except that these problems are run on all of ZenoTravel.

[7]The results differ markedly between the 2 and 3 city sets of problems because the sub-optimal relaxed plan extraction in the 2-cities problems coincidentally produces an essentially perfect heuristic in many of them. One should infer that the solutions found in the 2-cities problems are sharply bimodal in quality and that the meaning of the average is then significantly different than in the 3-cities problems.

function, largely by arguing that the size-based alternative is a notably more effective default strategy. We argued that using cost as the basis for plan evaluation is a purely exploitative/greedy perspective, leading to least interruptible behavior. Being least interruptible, it follows that implementing cost-based search will typically be immediately harmful to that particular application. Exceptions will abound, for example if costs are *not* wildly varying. The lasting harm, though, in taking cost-based evaluation functions as the default approach, failing to document any sort of justification for the risk so taken, is in reinforcing the wrong definition of *satisficing* in the first place. In conclusion, as a rule: Cost-based search is harmful.

References

Barto, A. G.; Bradtke, S. J.; and Singh, S. P. 1995. Learning to act using real-time dynamic programming. *AIJ* 81–138.

Benton, J.; Talamadupula, K.; Eyerich, P.; Mattmüller, R.; and Kambhampati, S. 2010. G-value plateaus: A challenge for planning. In *ICAPS*. AAAI Press.

Bonet, B. 2003. Labeled RTDP: Improving the convergence of real-time dynamic programming. In *ICAPS*, 12–21.

Dechter, R., and Pearl, J. 1985. Generalized best-first search strategies and the optimality of A*. *ACM* 32(3):505–536.

Dijkstra, E. W. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik* 1:269–271.

Dijkstra, E. W. 1968. Go to statement considered harmful. *Commun. ACM* 11:147–148.

Edelkamp, S., and Kissmann, P. 2009. Optimal symbolic planning with action costs and preferences. In *IJCAI*, 1690–1695.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* 4:100–107.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1972. Correction to "a formal basis for the heuristic determination of minimum cost paths". *SIGART* 28–29.

Helmert, M., and Röger, G. 2008. How good is almost perfect. In *AAAI*, 944–949.

Helmert, M.; Do, M.; and Refanidis, I. 2008. The deterministic track of the international planning competition. `http://ipc.informatik.uni-freiburg.de/`.

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

Kearns, M.; Capital, S.; and Singh, S. 1998. Near-optimal reinforcement learning in polynomial time. In *Machine Learning*, 260–268. Morgan Kaufmann.

Kocsis, L., and Szepesvari, C. 2006. Bandit based monte-carlo planning. In *ECML*.

Korf, R. E. 1985. Depth-first iterative-deepening: An optimal admissible tree search. *AIJ* 27:97–109.

Mcmahan, H. B.; Likhachev, M.; and Gordon, G. J. 2005. Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees. In *ICML*, 569–576.

Pearl, J. 1984. *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley.

Pohl, I. 1973. The avoidance of (relative) catastrophe, heuristic competence, genuine dynamic weighting and computational issues in heuristic problem solving. In *IJCAI*.

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

Sanner, S.; Goetschalckx, R.; Driessens, K.; and Shani, G. 2009. Bayesian real-time dynamic programming. In *IJCAI*.

Streeter, M., and Smith, S. F. 2007. New techniques for algorithm portfolio design. In *UAI*.

Thayer, J. T., and Ruml, W. 2009. Using distance estimates in heuristic search. In *ICAPS*. AAAI Press.

Thayer, J., and Ruml, W. 2010. Finding acceptable solutions faster using inadmissible information. In *Proceedings of the International Symposium on Combinatorial Search*.

Zilberstein, S. 1998. Satisficing and bounded optimality. In *AAAI Spring Symposium on Satisficing Models*.

A Deeper Analysis of the Results in Travel Domain

In this section we analyze the reported behavior of LAMA and SapaReplan in greater depth. We begin with a general analysis of the domain itself and the behavior of (simplistic) systematic state-space search upon it, concluding that cost-based methods suffer an enormous disadvantage. The empirical results are not nearly so dramatic as the dire predictions of the theory, or at least do not appear so. We consider to what extent the various additional techniques of the planners (violating the assumptions of the theory) in fact mitigate the pitfalls of ε-cost, and to what extent these only serve to mask the difficulty.

A.1 Analysis of Travel Domain

We argue that search under f_c pays a steep price in time and memory relative to search under $\hat{f}_s$. The crux of the matter is that the domain is reversible, so relaxation-based heuristics cannot penalize fruitless or even counter-productive passenger movements by more than the edge-weight of that movement. Then plateaus in g are plateaus in f, and the plateaus in g_c are enormous.

First note that the domain has a convenient structure: The global state space is the product of the state space of shuffling planes around between cities/airports via the fly action (expensive), and the state space of shuffling people around between (stationary) planes and cities/airports via the board/debark actions (cheap). For example, in the rendezvous problems, there are $5^4 = 625$ possible assignments of planes to cities, and $(5 + 4)^{2k}$ possible assignments of passengers to locations (planes + cities), so that the global state space has exactly $5^4 \cdot 9^{2k}$ reachable states (with k the number of passengers at one of the origins).[8]

Boarding and debarking passengers is extremely cheap, say on the order of cents, while flying planes between cities

[8]Fuel and zoom are distracting aspects of ZenoTravel-STRIPS, so we remove them. Clever domain analysis could do the same.

is quite a bit more expensive, say on the order of hundreds of dollars (from the perspective of passengers). So $\frac{1}{\varepsilon} \approx 10000$ for this domain — a constant, but much too large to ignore.

To analyze state-space approaches in greater depth let us make all of the following additional assumptions: The heuristic is relaxation-based, imperfect, and in particular heuristic error is due to the omission of actions from relaxed solutions relative to real solutions. Heuristic error is not biased in favor of less error in estimation of needed fly actions — in this problem planes are *mobiles* and *containers* whereas people are only *mobiles*. Finally, there are significantly but not overwhelmingly more passengers than planes.

Then consider a child node, in plane-space, that is in fact the correct continuation of its parent, but the heuristic fails to realize it. So its f is higher by the cost or size of one plane movement: 1 under normalized costs. Moreover assume that moving passengers is not heuristically good (in this particular subspace). (Indeed, moving passengers is usually a bad idea.) Then moving a passenger increases f_c by at most 2ε (and at least ε), once for g_c and once for h_c. As $\frac{1}{2\varepsilon} \approx 5000$ we have that search under f_c explores the passenger-shuffling space of the parent to, at least, *depth* 5000. Should the total heuristic error in fact exceed one fly action, then each such omission will induce backtracking to a further 5000 levels: for any search node n reached by a fly action set $e_c(n) = f_c(x) - f_c(n)$ with x some solution of interest (set e_s similarly). Then if search node n ever appears on the open list it will have its passenger-shuffling subspace explored, under f_c, to at least depth $e_c \cdot 5000$ before x is found (and at most depth $e_c \cdot \frac{1}{\varepsilon}$). Under $\hat{f}_s$, we have instead exploration up to at least depth $e_s \cdot \frac{1}{2}$ and at most depth $e_s \cdot \frac{1}{1}$.

As 5000 objects is already far above the capabilities of any current domain-independent planners, we can say that at most plane-shuffling states considered, cost-based search *exhausts* the entire associated passenger-shuffling space during backtracking. That is, it stops exploring the space due to exhausting finite possibilities, rather than by adding up sufficiently many instances of 2ε increases in f — the result is the same as if the cost of passenger movement was 0. Worse, such exhaustion commences immediately upon backtracking for the first time (with admissible heuristics). Unless *very* inadmissible (large heuristic weights), then even with inadmissible heuristics, still systematic search should easily get trapped on cost plateaus — before finding a solution.

In contrast, size-based search will be exhausting only those passenger assignments differing in at most e_s values; in the *worst* case this is equivalent to the cost-based method, but for good heuristics is a notable improvement. (In addition the size-based search will be exploring the plane-shuffling space deeper, but that space is [assumed to be] much smaller than any single passenger-shuffling space.) Then it is likely the case that cost-based search dies before reporting a solution while size-based search manages to find one or more.

A.2 Analyzing LAMA's Performance

While LAMA-size out-performs LAMA, it is hardly as dramatic a difference as predicted above. Here we analyze the results in greater depth, in an attempt to understand how LAMA avoids being immediately trapped by the passenger-shuffling spaces. Our best, but not intuitive, explanation is its pessimistic delayed evaluation leads to a temporary sort of depth-first bias, allowing it to skip exhaustion of many of the passenger-shuffling spaces until *after* finding a solution. So, (quite) roughly, LAMA is able to find one solution, but not two.

Landmarks. The passenger-shuffling subspaces are search plateaus, so, the most immediate hypothesis is that LAMA's use of landmarks helps it realize the futility of large portions of such plateaus (*i.e.*, by pruning them). However, LAMA uses landmarks only as a heuristic, and in particular uses them to order an additional (also cost-based) open list (taking every other expansion from that list), and the end result is actually greater breadth of exploration, not greater pruning.

Multiple Open Lists. Then an alternative hypothesis is that LAMA avoids immediate death by virtue of this additional exploration, i.e., one open list may be stuck on an enormous search plateau, but if the other still has guidance then potentially LAMA can find solutions due to the secondary list. In fact, the lists interact in a complex way so that conceivably the multiple-list approach even allows LAMA to 'tunnel' out of search plateaus (in either list, so long as the search plateaus do not coincide). Indeed the secondary list improves performance, but turning it off still did not cripple LAMA in our tests (unreported), let alone outright kill it.

Small Instances. It is illuminating to consider the behavior of LAMA and LAMA-size with only 4 passengers total; here the problem is small enough that optimality can be proved. LAMA-size terminates in about 12 minutes. LAMA terminates in about 14.5 minutes. Of course the vast majority of time is spent in the last iteration (with heuristic weight 1 and all actions considered) — and both are unrolling the exact same portion of state space (which is partially verifiable by noting that it reports the same number of unique states in both modes). There is only one way that such a result is at all possible: the cost-based search is re-expanding many more states. That is difficult to believe; if anything it is the size-based approach that should be finding a greater number of suboptimal paths before hitting upon the cheapest. The explanation is two-fold. First of all pessimistic delayed evaluation leads to a curious sort of depth-first behavior. Secondly, cost-based search pays far more dearly for failing to find the cheapest path first.

Delayed Evaluation. LAMA's delayed evaluation is not equivalent to just pushing the original search evaluation function down one level. This is because it is the *heuristic* which is delayed, not the full evaluation function. LAMA's evaluation function is the sum of the *parent's* heuristic on cost-to-go and the *child's* cost-to-reach: $f_L(n) = g(n) + h(n.p.v)$. One can view this technique, then, as a transformation of the original heuristic. Crucially, the technique increases the inconsistency of the heuristic. Consider an optimal path and the perfect heuristic. Under delayed evaluation of the perfect heuristic, each sub-path has an f_L-value in excess of f^* by exactly the cost of the last edge. So a high cost edge followed by a low cost edge demonstrates the non-monotonicity of f_L induced by the inconsistency wrought by delayed evaluation. The problem with non-monotonic evaluation functions is not the decreases *per se*, but the increases that precede them. In this case, a low cost edge followed by a high cost edge along an optimal path *induces backtracking*

despite the perfection of the heuristic prior to being delayed.
Depth-first Bias. Consider some parent n and two children x and y ($x.p = n$, $y.p = n$) with x reached by some cheap action and y reached by some expensive action. Observe that siblings are always expanded in order of their cost-to-reach (as they share the same heuristic value), so x is expanded before y. Now, delaying evaluation of the heuristic was pessimistic: $h(x.v)$ was taken to be $h(n.v)$, so that it appears that x makes no progress relative to n. Suppose the pessimism was unwarranted, for argument's sake, say entirely unwarranted: $h(x.v) = h(n.v) - c(x.a)$. Then consider a cheap child of x, say w. We have:

$$\begin{aligned} f_L(w) &= g(w) + h(x.v), \\ &= g(x) + c(w.a) + h(n.v) - c(x.a), \\ &= f_L(x) - c(x.a) + c(w.a), \\ &= f(n) + c(w.a), \end{aligned}$$

so in particular, $f_L(w) < f_L(y)$ because $f(n) + c(w.a) < f(n) + c(y.a)$. Again suppose that w makes full progress towards the goal (the pessimism was entirely unwarranted), so $h(w.v) = h(x.v) - c(w.a)$. So any of its cheap children, say z, satisfies:

$$\begin{aligned} f_L(z) &= g(w) + c(z.a) + h(x.v) - c(w.a), \\ &= f_L(w) - c(w.a) + c(z.a), \\ &= f_L(x) - c(x.a) + c(w.a) - c(w.a) + c(z.a), \\ &= f_L(x) - c(x.a) + c(z.a), \\ &= f(n) + c(z.a). \end{aligned}$$

Inductively, any low-cost-reachable descendant, say x', that makes full heuristic progress, has an f_L value of the form $f(n) + c(x'.a)$, and in particular, $f_L(x') < f_L(y)$, that is, all such descendants are expanded prior to y.

Generalizing, any low-cost-reachable and not heuristically bad descendant of any cheaply reachable child (x) is expanded prior to any expensive sibling (y).[9] Call that the good low-cost subspace.

Once such an expensive sibling is finally expanded (and the cost is found to be justified by the heuristic), then its descendants can start to compete on even footing once more. Except for the good low-cost subspaces: the good low-cost subspace of x is entirely expanded prior to the good low-cost subspace of y. In practice this means that LAMA is quite consistent about taking all promising low cost actions immediately, globally, like boarding all passengers in a problem, rather than starting some fly action halfway through a boarding sequence.

Then LAMA exhibits a curious, temporary, depth-first behavior initially, but in the large exhibits the normal breadth-first bias of systematic search. Depth-first behavior certainly results in finding an increasingly good sequence of plans to the same state. In this case, at every point in the best plan to some state where a less-expensive sibling leads to a slightly worse plan to the same state is a point at which LAMA finds a worse plan first. The travel domain is very strongly connected, so there are many such opportunities, and so we have a reasonable explanation for how LAMA could possibly be re-expanding more states than LAMA-size in the smallest instances of the travel domain.

Overhead. Moreover, the impact of failing to find the right plan first is quite distinct in the two planners. Consider two paths to the same plane-shuffling state, the second one actually (but not heuristically) better. Then LAMA has already expanded the vast majority, if not the entirety, of the associated passenger-shuffling subspace before finding the second plan. That entire set is then re-expanded. The size-based approach is not compelled to exhaust the passenger-shuffling subspaces in the first place (indeed, it is compelled to backtrack to other possibilities), and so in the same situation ends up performing less re-expansion work within each passenger-shuffling subspace. Then even if the size-based approach is overall making more mistakes in its use of planes (finding worse plans first), which is to be expected, the price per such mistake is notably smaller.

Summary. LAMA is out-performed by LAMA-size, due to the former spending far too much time expanding and re-expanding states in the ε-cost plateaus. It fails in "depth-first" mode: finding not-cheapest almost-solutions, exhausting the associated cheap subspace, backtracking, finding a better path to the same state, re-exhausting that subspace, ..., in particular exhausting memory extremely slowly (it spends all of its time re-exhausting the *same* subspaces).

A.3 Analyzing the Performance of SapaReplan

The contrasting failure mode, "breadth-first", is characterized by exhausting each such subspace as soon as it is encountered, thereby rapidly exhausting memory, without ever finding solutions. This is largely the behavior of SapaReplan (which does eager evaluation), with cost-based methods running out of memory (much sooner than the deadline, 30 minutes) and size-based methods running out of time. So for SapaReplan it is the size-based methods that are performing many more re-expansions, as in a much greater amount of time they are failing to run out of memory. From the results, these re-expansions must be in a useful area of the search space.

In particular it seems that the cost-based methods must indeed be exhausting the passenger-shuffling spaces more or less as soon as they are encountered — as otherwise it would be impossible to both consume all of memory yet fail to find better solutions. (Even with fuel there are simply too few distinct states modulo passenger-shuffling.) However, they do find solutions before getting trapped, in contradiction with theory.

The explanation is just that the cost-based methods are run with large (5) heuristic weight, thereby introducing significant depth-first bias (but not nearly so significant as with pessimistic delayed evaluation), so that it is possible for them to find a solution before attempting to exhaust such subspaces. It follows that they find solutions within seconds, and then spend minutes exhausting memory (and indeed that is what occurs). The size-based methods are run with small heuristic weight (2) as they tend to perform better in the long run that way. It would be more natural to use the same heuristic weight for both types, but, the cost-based approaches do conform to theory with small heuristic weights — producing no solutions, hardly an interesting comparison.

[9] The bound on heuristic badness is $c(y.a)$.

Living on the Edge: Safe Search with Unsafe Heuristics

Erez Karpas and **Carmel Domshlak**
Faculty of Industrial Engineering & Management,
Technion, Israel

Abstract

Considering heuristic search for satisficing and cost-optimal planning, we introduce the notion of *global safeness* of a heuristic, a property weaker than the standard safeness, which we claim is better for heuristic search. We describe one concrete approach for creating globally safe and optimality preserving heuristics by exploiting information in the history of the search via a new form of inference related to existential landmarks. We evaluate our approach on some state-of-the-art heuristic search tools for cost-optimal planning, and discuss the outcomes of this evaluation.

Introduction

These days, state-space heuristic search is the most prominent approach to classical planning, though the way it is exploited in planning differ in some crucial respects from the way it is approached in the search community. In most heuristic search literature, the assumption is that the search problem and the heuristic are both given as a "black box"—the structure of the search space and the way the heuristic works are hidden from the search algorithm. This assumption suffices for analyzing various properties of the search algorithms, but limits the palette of generic enhancements that can possibly be applied to heuristic search. In contrast, in domain independent planning, the structure of the search space is made explicit by means of a description of the problem in terms of state variables, action preconditions and effects specified in terms of these state variables, and so on. This allows for exploiting more information than would be available in "classical" heuristic search. This is not a new observation — in fact, helpful actions and preferred operators (Hoffmann and Nebel 2001; Helmert 2006) are just one example of using more information than would be available for a "classical" heuristic-search algorithm.

Although preferred operators still only look at one search state, recent work has shown that it's possible use more information than is available in a single state. One example is the *LM-A** search algorithm (Karpas and Domshlak 2009), which uses information from multiple paths in the search space to enhance landmark-based heuristics. Another example of a "non-classical" heuristic is selective-max, an online learning approach which uses information from previously evaluated states, in the form of examples for a classifier (Domshlak, Karpas, and Markovitch 2010).

In this paper, we continue this line of investigation by taking a closer look at the *safeness* property of heuristics. A heuristic function h is called *safe* if for all states $s \in S$, if h declares s to be a dead-end (that is, $h(s) = \infty$), then s really is a dead-end. We claim that this definition is too restrictive, and instead formulate the more desired property of *global-safeness*. We also demonstrate how one can derive a path-dependent globally-safe heuristic, and draw connections to a new type of information which can be inferred, which we call existential landmark.

Preliminaries

We consider planning tasks formulated in STRIPS (Fikes and Nilsson 1971). A planning task $\Pi = \langle P, A, s_0, G\rangle$, where P is a set of propositions, A is a set of actions, each of which is a triple $a = \langle \mathsf{pre}(a), \mathsf{add}(a), \mathsf{del}(a)\rangle$, $s_0 \subseteq P$ is the inital state, and $G \subseteq P$ is the goal. For ease of notation and without loss of generality, in what follows we assume that there is a single goal proposition ($G = \{p_g\}$), which can only be achieved by one action, END.

An action a is applicable in a state s if $\mathsf{pre}(a) \subseteq s$. Applying action a in state s results in the new state $s' = (s \setminus \mathsf{del}(a)) \cup \mathsf{add}(a)$. A sequnce of actions $\langle a_0, a_1, \ldots, a_n\rangle$ is applicable in state s_0 if a_0 is applicable in s_0 and results in state s_1, a_1 is applicable in s_1 and results in s_2, and so on. An action sequence $\langle a_0, a_1, \ldots, a_n\rangle$ is a valid plan for Π if it is applicable in s_0, and $a_n = END$. An optimal plan is (one of) the shortest such valid plans. If π_1 and π_2 are two action sequences, by $\pi_1 \cdot \pi_2$ we denote the concatenation of π_1 and π_2.

Globally Safe Heuristics

A heuristic h is *safe*, if for every state s from which there is a path to the goal, $h(s) < \infty$. In other words, h does not declare any false dead ends. Most of the heuristics used in domain-independent planning (including those based on delete-relaxation, abstractions, and critical paths) are safe.

While safeness can be a useful property for pruning search sub-spaces, it is relevant only in problems exhibiting dead-ends in the first place. In general, we claim that safeness is too strong a requirement. Consider the hypothetical ideal case, where we have a heuristic assigning a value of h^* (or, actually, any other finite value) to the states along some optimal plan, and ∞ to all other states. Using any reasonable search algorithm with this heuristic would lead straight to finding an optimal solution, without getting sidetracked at all. However, this heuristic is clearly not safe, since it declares many false dead-ends. This idealized scenario suggests examining substantial relaxations of the safeness property, defined below.

Definition 1 (G-safe & GO-safe heuristics) *A heuristic h is called* globally safe (G-safe, for short) *if, for any planning task* Π*, if* Π *is solvable, then there exists a valid plan* ρ *for* Π *such that for any state s along ρ, $h(s) < \infty$. In particular, if there exists such an* optimal *plan ρ, then h is called* globally safe with respect to optimality (or GO-safe, for short).

The following proposition follows directly from the definition.

Proposition 1 *Safeness implies GO-safeness implies G-safeness.*

The next propositions capture the main attractiveness of employing G-safe and GO-safe heuristics.

Proposition 2 *Any complete systematic search algorithm, which detects dead-ends using a G-safe heuristic h, will find a solution for any solvable planning task* Π.

Proof sketch: h is G-safe, and thus there is a plan ρ such that states along ρ are not declared by h to be dead-ends. Let s be the shallowest state along ρ which has not been expanded yet. Since we are using a complete search algorithm, search will not terminate before s is expanded, unless another solution is found before that. Thus, either the solution ρ will be found, or another solution will be found before ρ is fully expanded. ■

Proposition 3 *Let h be a GO-safe heuristic, admissible for all states which it does not declare as dead-ends. Then, for any solvable planning task* Π*, A^* search using h will find an optimal solution for* Π.

Proof sketch: Consider the transition system induced by h, which is the transition system of Π, with edges outgoing from states that are declared by h to be dead-ends removed. This is the transition system that is searched when using h. Because h is GO-safe, every optimal path to the goal in the induced transition system corresponds to an optimal solution of Π. And since h is admissible in the induced transition system, A^* will find an optimal path in that system. ■

While the notion of globally safe heuristics is appealing in theory, we still need to show that a concrete non-trivial instance of this notion actually exists. Indeed, at the moment of writing this paper, we could not provide the reader with an interesting example of *state-dependent* G-safe heuristics. However, the situation is different with the more general family of *path-dependent heuristics*, and thus we proceed with formalizing our relaxed versions of safeness with respect to such heuristics.

Definition 2 (G-safe path-dependent heuristic) *A path-dependent heuristic h is called* G-safe*, if, for any planning task* Π*, if* Π *is solvable, then there exists a valid plan ρ for* Π*, such that for every prefix ρ' of ρ, $h(\rho) < \infty$. That is, when h evaluates any prefix of ρ, it is not declared as a dead-end. In particular, if there exists such an* optimal *plan ρ, then h is called* GO-safe.

In fact, since any state-dependent heuristic can be seen as a path-dependent heuristic which only looks at the last state reached by the path, Definition 1 can be seen as a special case of Definition 2. In the rest of the paper, we describe a procedure for creating GO-safe path-dependent heuristics, and share our findings on the empirical effectiveness of adopting GO-safeness.

Unjustified Actions and Global Safeness

Our G-safe path-dependent heuristics are based upon the notion *unjustified actions*. Informally, an action along a plan is unjustified if removing it from that plan does not invalidate the latter. In order to formally define unjustified actions, we use the notion of plan's *causal links*.

Let $\pi = \langle a_0, a_1, \dots a_n \rangle$ be an action sequence applicable in state s_0. The triple $\langle a_i, p, a_j \rangle$ forms a *causal link* if $i < j$, $p \in \mathsf{add}(a_i)$, $p \in \mathsf{pre}(a_j)$, $p \notin s_i$ (s_i is the state before applying action a_i), and for $i < k < j$, $p \notin \mathsf{del}(a_k) \cup \mathsf{add}(a_k)$. In other words, p is a precondition of a_j, and is achieved by a_i, and is not deleted or added by some other action until a_j occurs. In such a causal link, a_i is called its *supporter*, and a_j is called its *consumer*.

Definition 3 (Unjustified Action) *Given a plan $\rho = \langle a_0, a_1, \dots a_n \rangle$, the action instance $a_i \neq END$ is* unjustified *if there is no causal link in ρ such that a_i is the supporter in that causal link.*

An immediate from the definition, yet important for what comes next, property is that *optimal plans never contain unjustified actions*. This brings us to define "hopeless paths" in the (forward) search.

Definition 4 (Hopeless paths) *For any planning task* Π*, path π from the initial state s_0 to a state s is* hopeless *if there is no path π' from s to the goal such that $\pi \cdot \pi'$ contains no unjustified actions.*

In other words, if π is hopeless, then any plan that has π as a prefix will contain unjustified occurrences of actions. Using this definition, we can finally formulate the connection between unjustified actions and global safeness.

Theorem 1 (Global path-dependent safeness) *Let h be a safe path-dependent heuristic. Then the path-dependent heuristic*

$$h'(\pi) := \begin{cases} \infty & \text{if } \pi \text{ is hopeless} \\ h(\pi) & \text{otherwise} \end{cases}$$

is a GO-safe path-dependent heuristic.

Proof sketch: Again, the claim trivially holds for unsolvable tasks. Let Π be a solvable planning task, and let $\rho = \langle a_0, a_1, \ldots a_n \rangle$ be an optimal plan for Π. By the virtue of being optimal, π does not contain any unjustified actions. Let $\pi = \langle a_0, a_1, \ldots a_i \rangle$ be a prefix of ρ, leading to state s. Then $\pi' = \langle a_{i+1}, a_{i+2}, \ldots a_n \rangle$ is a path from s to the goal, such that $\pi \cdot \pi'$ contains no unjustified actions. Therefore π is not hopeless, and there thus is at least one optimal plan which is never declared as a dead-end by h'. ■

A word of caution is in place here. Note that according to Theorem 1 it might be safe to declare a path π as a dead-end, but *not* the state s to which π leads. This happens because of the path-dependent nature of unjustified actions. To illustrate that, consider the following simple planning task $\Pi = \langle P, A, s_0, G \rangle$, where $P = \{p_1, p_2, p_g\}$, $s_0 = \{\}$, $G = \{p_g\}$, and the following actions:

- $a_1 = \langle \emptyset, \{p_1\}, \emptyset \rangle$
- $a_2 = \langle \{p_1\}, \{p_2\}, \emptyset \rangle$
- $a_{12} = \langle \emptyset, \{p_1, p_2\}, \emptyset \rangle$
- $END = \langle \{p_1, p_2\}, \{p_g\}, \emptyset \rangle$

Following path $\pi = \langle a_1, a_{12} \rangle$ leads to state $s = \{p_1, p_2\}$. Action a_1 along π can not be justified, since a_{12} achieves p_1, the proposition that a_1 achieves, and it is not a consumer of a_1. Therefore, π is hopeless (since there is no path π' from s to the goal that can justify a_1), and it is indeed globally safe to declare path π as a dead-end. However, it is *not* safe to declare state s as a dead-end, since Π is solvable, and it is easy to see that s lies on any solution path. The following theorem addresses this issue.

Theorem 2 (Global safeness along optimal paths)
If h be a safe heuristic, then the heuristic

$$h'(s) := \begin{cases} \infty & \textit{if some optimal path from } s_0 \textit{ to } s \textit{ is hopeless} \\ h(s) & \textit{otherwise} \end{cases}$$

is a GO-safe heuristic.

Proof sketch: Let Π be a (solvable) planning task, and s be a state of Π. If there is no optimal plan for Π which goes through state s, then by definition, it is GO-safe to declare s as a dead end. Assume now that there is an optimal plan $\rho = \rho_1 \cdot \rho_2$ for Π such that applying ρ_1 in s_0 leads to s. Let π be an optimal path from s_0 to s. Assume for contradiction that π is hopeless, and there is no path π' from s to the goal such that $\pi \cdot \pi'$ contains no unjustified actions. Then, specifically, $\pi \cdot \rho_2$ contains some unjustified action. However, $\pi \cdot \rho_2$ is an optimal plan (since π is an optimal path to s, ρ_2 is an optimal path from s to the goal, and s is along an optimal plan), and that contradicts the fact that optimal plans never contain unjustified actions. Hence, if an optimal path π from s_0 to state s is hopeless, then s is not on an optimal path from s_0 to the goal. ■

The problem with applying Theorem 2 is that we do not always know when the current path to some state s is indeed optimal. However, by modifying the A^* search algorithm to reevaluate the heuristic *every time* a state s is reopened (because a shorter path to s has been found), we can prune states according to Theorem 2, and still guarantee that the search will return an optimal solution. For the sake of brevity, we omit a formal proof of this, but refer the reader to the proof that A^* with an admissible heuristic returns an optimal solution, and specifically to the following lemma (Pearl 1984, p. 78, Lemma 2): *Let n' be the shallowest open node on an optimal path π from s_0 to some arbitrary node n''. Then $g(n') = g^\star(n')$.*

Exploiting Unjustified Actions in Search

While Theorem 1 brings us closer to globally safe heuristics, we still need a way of identifying whether a given path from the initial state is hopeless. In what follows, we close the gap by presenting two such methods.

Existential Landmarks

Suppose state s was reached via path $\pi = \langle a_0, a_1, \ldots, a_n \rangle$. Using standard causal link analysis, we can identify the causal links present in π. We make one slight enhancment to the standard analysis by not allowing an action a to justify its inverse action a', where inverse actions are identified according to the criteria in Hoffmann (2002). Denote by U the set of actions in π which are not supporters in any causal link in π. The pseudo code for extracting U is depicted in Figure 1.

For π to have a continuation π' such that $\pi \cdot \pi'$ has no useless occurrences of actions (and thus not be hopeless), every action in U must be the supporter of some action in π'. Using the same causal link analysis, we can also identify which propositions can possibly appear in such causal links for each action. Denote by $pp(a)$ the set of propositions which $a \in U$ is potentially a supporter of. Note that in some cases, it is possible to detect dead-ends, just by noticing that $pp(a) = \emptyset$, which means that all of the effects of a have either been reachieved by some other action or deleted, and so there is no way to justify a.

For $\pi \cdot \pi'$ to contain no useless actions, for all $a \in U$, a must support some proposition in $pp(a)$. Let $ia(a) = \{a' \mid \mathsf{pre}(a') \cap pp(a) \neq \emptyset\}$ denote the set of all actions which have a precondition in $pp(a)$. Then any continuation π' of π, such that $\pi \cdot \pi'$ contains no useless occurrences of actions, must use an action from $ia(a)$ for each $a \in U$. Although this seems similar to the notion of disjunctive action landmark, the difference is that $ia(a)$ does not constrain *all* plans, but rather only *optimal plans having π as a prefix.* Since $ia(a)$ means that there *exists* some plan which achieves it, we call it an existential disjunctive action landmark.

Once we have these existential disjunctive action landmarks, it is possible to reason about them in combination with "regular" landmarks. For example, it is possible to find a cost partitioning between these existential disjunctive action landmarks, and any other set of landmarks, such as those used by h_{LA} (Karpas and Domshlak 2009). The "achievers" of such an existential disjunctive action landmark are simply the actions that compose the landmark.

```
causal-link-analysis($\pi$)
$support := \emptyset$
// $support$ holds pairs of propositions
// and the actions that supported them
for $a_i \in \pi$ (in order)
    update($a_i$)

update($a$)
for $p \in (\mathsf{pre}(a) \cap supported\_props)$
    // We have used the supported proposition,
    // it is no longer unjustified
    $a' := supporter(p)$
    // Make sure an action doesn't justify its inverse
    if $a'$ is not the inverse of $a$
        // remove $a'$ and all its supported propositions
        $support := support \setminus \{\langle p', a' \rangle | \exists p' : \langle p', a' \rangle \in support\}$
for $p \in (\mathsf{add}(a) \cup \mathsf{del}(a))$
    // We need to justify the effects of $a$ later
    if $p \in supported\_props$
        // This effect was already achived by another action
        // we need to make sure this is not its last effect
        $a' := supporter(p)$
        $support := support \setminus \{\langle p, a' \rangle\}$
        if $supported\_by(a') = \emptyset$ then
            return dead-end
    $support := support \cup \{\langle p, a \rangle\}$

Macros used:
$supported\_props \equiv \{p | \exists a : \langle p, a \rangle \in support\}$
$supported\_by(a) \equiv \{p | \langle p, a \rangle \in support\}$
$supporter(prop) \equiv a$ s.t. $\langle p, a \rangle \in support$
```

Figure 1: Procedure for the causal link analysis.

A Compilation-Based Approach

A different approach for identifying hopeless paths is based on compiling into the planning task Π (part of) the constraints imposed on it by unjustified actions. Let state s be a state reached via path π. We will define a new planning task $\Pi'_{s,\pi} = \langle P', A', s', G' \rangle$ as follows:

- $P' = P \cup \{justified(a) \mid a \in A\}$
- $A' = \{\langle \mathsf{pre}(a) \cup \{justified(a)\},$ $\mathsf{add}(a) \cup \{justified(a') \mid a' \in A, \mathsf{add}(a') \cap \mathsf{pre}(a) \neq \emptyset\},$ $\mathsf{del}(a) \cup \{justified(a)\}\rangle \mid a \in A\}$
- $s' = s \cup \{justified(a) \mid a$ is not unjustified in π or a does not occur in $\pi\}$
- $G' = G \cup \{justified(a) \mid a \in A\}$

In words, we add a proposition $justified(a)$ for each action a, with the meaning that $justified(a)$ holds when action a has been justified (or does not need to be justified). Applying action a deletes $justified(a)$, thus forcing some later action to use one of a's effects. Applying action a', which has one of a's effects as a precondition, adds $justified(a)$, since a' used an effect of a.

Note that this compilation is weaker than Definition 3, since we do not require that no action deletes or adds the supported proposition along the way, and we do not account for which action achieved which proposition. However, we can still show the following.

Theorem 3 (Soundness of Π') *Let Π be a solvable planning task, and let state s be reached via path π. If π is not hopeless, then $\Pi'_{s,\pi}$ is solvable.*

Proof sketch: Path π is not hopeless, therefore there exists some path π' such that $\pi \cdot \pi'$ contains no unjustified actions. We will show that π' is also a solution for $\Pi'_{s,\pi}$. Since $\pi \cdot \pi'$ contains no unjustified actions, for every unjustified action a in π, there is an action a' in π' that uses an effect of a. Therefore π' will achieve all the $justified(a)$ propositions which were false in s'. Furthermore, no action in π' is unjustified, and so for every action a along π', there is another action in π' which achieves $justified(a)$. Since π' achieves all of the goals of Π, and all of the $justified$ goals of $\Pi'_{s,\pi}$, π' is a solution for $\Pi'_{s,\pi}$. ■

It is also possible to come up with a different compilation, which has a proposition $achieved(a, p)$ for each pair of action a and proposition p (from the original task), and denotes that p was achieved by action a. Such a compilation must use conditional effects, but has the potential to be more informative than the simple compilation described here, since it does not ignore the information about which action achieved which proposition.

Empirical Evaluation

We have presented the concept of unjustified actions, and suggested two principled ways in which they can be explited in heuristic-search planning. Since the compilation-based approach requires introducing many new state variables (one for each grounded action), which severly limits the applicabilty of this approach in practice, we did not perform an empirical evaluation of it in this paper.

We have implemented the existential landmark approach on top of the Fast Downward planning system (Helmert 2006), and combined the existential disjunctive action landmarks with the optimal cost partitioning of landmarks (Karpas and Domshlak 2009). In our evaluation we used three baseline heuristic: $h_{\text{LM-CUT}}$, h_{LA}, and h_{GC}.

- h_{GC} is the most basic admissible landmark-based heuristic: it only accounts for the goal landmarks, and combines their information via the optimal action cost partitioning. In other words, h_{GC} is simply an admissible goal-count heuristic. We add the existential action landmarks discovered using our approach to the cost partitioning.
- h_{LA} (Karpas and Domshlak 2009) is state-of-the-art admissible landmarks heuristic, with efficient optimal cost partitioning (Keyder, Richter, and Helmert 2010). The landmarks used here are those discovered using the RHW method (Richter and Westphal 2010). Here as well, we add the existential action landmarks discovered using our approach to the cost-partitioning.

Domain	h_{GC}	h_{GC}+	h_{LA}	h_{LA}+	$h_{\text{LM-CUT}}$	$h_{\text{LM-CUT}}$+
BLOCKS	17	17	21	21	28	28
DEPOT	3	**4**	7	7	7	7
DRIVERLOG	7	**9**	12	**13**	13	13
LOGISTICS00	10	10	20	20	20	20
TRUCKS-STRIPS	3	3	**6**	5	**10**	9
ZENOTRAVEL	8	8	9	9	13	13
TOTAL	48	51	75	75	91	90

Table 1: Total number of solved tasks for each method. Results where there was a change between a baseline and its enhancement are in bold.

- $h_{\text{LM-CUT}}$ (Helmert and Domshlak 2009) is a state-of-the-art admissible landmarks heuristic, but unfortunately, it does not support adding the existential landmark information to its cost partitioning. Therefore, the only enhancement we used with $h_{\text{LM-CUT}}$ was path pruning: upon reaching a state s via path π, we look at the causal link analysis of π, and if there is an action in π such that $pp(a) = \emptyset$ (and therefore a can not be justified), then we prune path π.

We compare the baseline heuristics to the same heuristics, enhanced by our existential landmarks (as explained above). To ensure admissibility, in the runs enhanced with unjustified action information, we modified A^* to re-evaluate the heuristic once a shorter path to a known state has been found. All of the experiments were conducted with a 3GB memory limit and a 30 minute time limit, on a single core of a 2.33GHz Intel Q8200 CPU.

Table 1 lists the number of tasks solved using each baseline configuration, and the number of tasks solved when using unjustified actions. Overall, there is not much change in the number of solved tasks, and the overhead involved in keeping track of unjustified actions even leads to some losses. However, looking into the results in more details reveals a more colorful picture.

Tables 2 lists the average ratios of expanded states, evaluations and total solution time, when using unjustified actions, relative to to the baseline of the same method. For each heuristic, this is averaged on the tasks solved by both the baseline and the version enhanced with unjustified actions.

Looking at Table 2a, we can see that the number of expanded states is reduced drastically. We remark that only in one ZENOTRAVEL task, using unjustified actions led to more expanded states than with the baseline, when using $h_{\text{LM-CUT}}$. Especially remarkable is the finding that in LOGISTICS00, we can reduce the number of expanded states by more than half, simply by pruning hopeless paths, without modifying the $h_{\text{LM-CUT}}$ heuristic at all. On the other hand, using unjustified actions with $h_{\text{LM-CUT}}$ in BLOCKS did not make any difference. This is because the *hand-empty* predicate is achieved by any action which puts a block down, and is a precondition of any action which picks a block up, thus any action justifies the next action.

From Table 2b, we can see that even considering that the same state might be evaluated more than once with unjustified actions, the number of evaluations is still reduced overall. However, there is more than one task where the number of evaluations is increased over the baseline. Even taking the overhead of keeping track of unjustified actions into account, Table 2c still shows us that overall, using unjustified actions with $h_{\text{LM-CUT}}$ and h_{GC} speeds up search.

Domain	h_{GC} ratio	h_{LA} ratio	$h_{\text{LM-CUT}}$ ratio
BLOCKS	0.93	0.99	1.00
DEPOT	0.56	0.84	0.98
DRIVERLOG	0.58	0.68	0.82
LOGISTICS00	0.57	0.97	0.43
TRUCKS-STRIPS	0.5	0.57	0.9
ZENOTRAVEL	0.53	0.83	0.92
AVG.	0.69	0.87	0.82
NORMALIZED AVG.	0.61	0.81	0.84

(a) Expanded States

Domain	h_{GC} ratio	h_{LA} ratio	$h_{\text{LM-CUT}}$ ratio
BLOCKS	0.93	1.00	1.00
DEPOT	0.64	0.92	0.99
DRIVERLOG	0.64	0.76	0.86
LOGISTICS00	0.61	0.99	0.52
TRUCKS-STRIPS	0.64	0.73	0.90
ZENOTRAVEL	0.58	0.89	0.91
AVG.	0.73	0.92	0.85
NORMALIZED AVG.	0.67	0.88	0.86

(b) Evaluations

Domain	h_{GC} ratio	h_{LA} ratio	$h_{\text{LM-CUT}}$ ratio
BLOCKS	1.12	1.11	1.06
DEPOT	1.03	1.22	1.02
DRIVERLOG	0.84	0.96	0.98
LOGISTICS00	0.86	1.30	0.64
TRUCKS-STRIPS	1.03	1.29	1.01
ZENOTRAVEL	0.81	1.16	0.93
AVG.	0.96	1.17	0.93
NORMALIZED AVG.	0.95	1.17	0.94

(c) Total Time

Table 2: Average ratios of expanded states / evaluations / total time with unjustified actions relative to the baseline. For each heuristic, the average is over tasks solved by both the baseline and the version enhanced with unjustified actions. AVG. is the average over all tasks, and NORMALIZED AVG. is the average over domain averages.

Related

While the use of information from other states presented here might seem somewhat similar to the heuristic value propagation of pathmax (Mero 1984) or BPMX (Felner et al. 2005), this is not the case. Our approach uses knowledge about the explicit structure of the problem, in the form of causal links, and is therefore not applicable within the "classic" heuristic search, black-box view of the problem. On the other hand, pathmax and BPMX only need information about heuristic values and successor relations, and so they are applicable (and have been applied) without explicit structure of the problem.

Another field where more information than present in a single search state is used to make decisions is learning for planning. Typically, a learning/planning system attempts to learn some domain-specific knowledge offline from training examples, in order to create a more efficient planner on unseen problem instances from that domain. However, there is very little work on *online* learning for planning, which would also constitute a form of non-classical heuristic search.

A similar notion to unjustified actions, termed useless actions, was introduced by Wehrle, Kupferschmid, and Podelski (2008). The authors there defined a useless action to be an action which is not the start of an optimal path. However, it is PSPACE-hard to determine if an action is useless, and so Wehrle et al. use an approximation of useless actions, and can no longer guarantee optimal solutions. Furthermore, the definition of useless action suffers from the same problem as the definition of a safe heuristic — any action which is the start of an optimal path is not useless. It would be very interesting to identify a condition which is sufficient to ensure that a given action is the start of an optimal path, and then just prune the rest of the search space.

Conclusion

We have defined a novel notion of global safeness, and have demonstrated how unjustified actions can be used to derive globally safe heuristics. We also performed an empirical evaluation, demonstrating that exploiting global safeness via unjustified actions can lead to significant improvements to the performance of sequentially-optimal planning.

Note that using unjustified actions in satisficing search is even easier since there is no need to worry about maintaining admissibility, and we intend to explore these possibilities in our future work. Finding an efficient way to apply the compilation-based approach presented here is yet another interesting direction to explore.

References

Domshlak, C.; Karpas, E.; and Markovitch, S. 2010. To max or not to max: Online learning for speeding up optimal planning. In *AAAI*.

Felner, A.; Zahavi, U.; Schaeffer, J.; and Holte, R. C. 2005. Dual lookups in pattern databases. In *IJCAI*, 103–108.

Fikes, R. E., and Nilsson, N. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *AIJ* 2:189–208.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In *ICAPS*. In press.

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. 2002. Local search topology in planning benchmarks: A theoretical analysis. In *AIPS*. 379-387.

Karpas, E., and Domshlak, C. 2009. Cost-optimal planning with landmarks. In *IJCAI*.

Keyder, E.; Richter, S.; and Helmert, M. 2010. Sound and complete landmarks for and/or graphs. In *ECAI*, 335–340.

Mero, L. 1984. A heuristic search algorithm with modifiable estimate. *AIJ* 23:13–27.

Pearl, J. 1984. *Heuristics: intelligent search strategies for computer problem solving*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Richter, S., and Westphal, M. 2010. The lama planner: Guiding cost-based anytime planning with landmarks. *JAIR* 39:127–177.

Wehrle, M.; Kupferschmid, S.; and Podelski, A. 2008. Useless actions are useful. In *ICAPS*, 388–395. AAAI Press.

Enhancing Search for Satisficing Temporal Planning with Objective-driven Decisions

J. Benton[†] and **Patrick Eyerich**[‡] and **Subbarao Kambhampati**[†]

[†] Dept. of Computer Science and Eng.
Arizona State University
Tempe, AZ 85287 USA
{j.benton,rao}@asu.edu

[‡] Department of Computer Science
University of Freiburg
Freiburg, Germany
eyerich@informatik.uni-freiburg.de

Abstract

Heuristic best-first search techniques have recently enjoyed ever-increasing scalability in finding satisficing solutions to a variety of automated planning problems, and temporal planning is no different. Unfortunately, achieving efficient computational performance often comes at the price of clear guidance toward solution of high quality. This fact is sharp in the case of many best-first search temporal planners, who often use a node evaluation function that is mismatched with the objective function, reducing the likelihood that plans returned will have a short makespan but increasing search performance. To help mitigate matters, we introduce a method that works to progress search on actions declared "useful" according to makespan, even when the original search may ignore the makespan value of search nodes. We study this method and show that it increases over all plan quality in most of the benchmark domains from the temporal track of the 2008 International Planning Competition.

Introduction

Heuristic best-first search planning methods have been the de facto standard for generating scalable satisficing planning algorithms for over a decade. This success has lead to using these techniques in a broad range of problems, including temporal planning. In temporal planning problems, actions have duration and the objective function (f_{OF}) is typically to minimize plan makespan (i.e., f_{OF} is plan makespan). However, as recently pointed out by Benton et al. (2010), using makespan to evaluate search nodes (i.e., in typical A^*-like fashion with $f = g + h$) can significantly reduce the scalability of search. Instead, most temporal planners tend to avoid directly searching on makespan and use other measures for node evaluation to achieve scalable performance (c.f., Sapa (Do and Kambhampati 2003) and TFD (Eyerich, Mattmüller, and Röger 2009)). Unfortunately, taking this approach can come at the cost of plan quality—one cannot necessarily expect a low makespan when the node evaluation of search is mismatched with the objective function.

Although the strategy of optimizing for f_{OF} using a search not explicitly guided by f_{OF} might seem quixotic, we have argued elsewhere (Cushing, Benton, and Kambhampati 2011) that this is, in fact, the only reasonable thing to do when there is a high cost-variance. Indeed, in makespan planning we have g-value plateaus over f_{OF} (i.e., where g_{OF} does not change from the parent to child node and $f_{OF} = g_{OF} + h_{OF}$) and hence an implied infinite cost variance. Nevertheless, while strategies that use a node evaluation function other than f_{OF} can improve scalability, and may find reasonably low makespan plans using branch-and-bound search, the quality improvement within a given time limit may not be sufficiently high.

To try to improve matters, we adopt a general way of pushing the base-level search toward higher quality plans according to f_{OF}. We seek to augment search approaches where $f \neq f_{OF}$ and f is the node evaluation function for an A^*-like best-first search. We adapt the concept of "uselessness" introduced by Wehrle, Kupferschmid, and Podelski (2008), who prune away certain states from consideration that appear to offer no benefit for finding a solution toward the goal. In specific, our approach evaluates search nodes according to their degree of "usefulness" on f_{OF} itself.

The idea is to provide a window into the objective function while still maintaining scalability. This is done, in particular, on areas of the search space that appear to have *zero-cost* according to f_{OF}. For makespan, this lets us consider parallelizing actions as much as possible (i.e., adding an action that runs in parallel with others often has no effect on makespan, so it is highly desirable for minimization). Among nodes generated from a *zero-cost* (i.e., *zero-makespan*) action, we choose the node with high usefulness for immediate expansion, a local lookahead that allows the main search algorithm to then evaluate its children on the evaluation function f. In the end, we get a final heuristic search that proceeds in two steps: (1) a best-first search step that uses the planner's usual *open list* structure and search (i.e., including its usual node evaluation function f used by the search), and (2) a local decision step where the most useful search states (according to f_{OF}) are expanded.

The rest of this paper proceeds as follows. After an overview of background and notation, we discuss the idea of operator usefulness. We then show how to embed local decisions on usefulness into a best-first search framework. Finally, we present empirical analysis showing the quality benefits that can be gained by using this approach in the planner Temporal Fast Downward (TFD) (Eyerich, Mattmüller, and Röger 2009), a state-of-the-art best-first search heuristic planner for temporal planning.

Background and Notation

Before going into our technique, we must first cover some background information and notation on best-first search, temporal planning, g-value plateaus and useless search operators.

Best-First Search

Best-first search is a well-known algorithm for searching over a (possibly infinite) state space from an initial state I to a state that entails the goal G (Pearl 1984). We provide a brief overview of it while introducing notation. The algorithm, shown in Algorithm 1, provides an *open list*, where the best-valued state may be removed, and a *closed list* which allows for duplicate detection.

It begins with the initial state I in the open list. It then repeats the following process: The best-valued state (according to an evaluation function f) s is taken. If s is not in the closed list it is added, otherwise we skip this state (as it has already been explored). If $s \models G$, then s is returned as a solution, otherwise s is *expanded*. This means all possible successor states (also known as *child* states) are generated. To generate a child state we use an operator $o = \langle p, e, c \rangle$, where p is a set of conditions for the applicability of o, e is a transformation function on states (i.e., effects), and c is a constant-valued cost. o is considered applicable on s (the *parent* state) if $s \models p$. Provided the conditions of o are met, we define $o(s) = s'$ to be the child state of s, such that s' is the state resulting from the transformation function e on s. Each child state is evaluated according to f. In the best-first search algorithm A^*, $f(s') = g(s') + h(s')$, where g is a function returning the current (known) cost of s' and h is a function returning a heuristic distance from s' to the least-cost goal node. After evaluation, s' is added into the open list.

Importantly, the function f and its components g and h, are often defined in terms of the objective function of the search. For instance, in cost-based classical planning, where we seek to minimize the cost of a plan, we can define f, g, and h in terms of the summed costs of actions in a plan. Provided h always returns a lower bound on the cost to a goal, this guarantees optimal solutions. Even when this is not the case, the conventional wisdom and empirical evidence points to this yielding better-quality solutions than metrics like search distance (c.f., Richter and Westphal (2010)).

Algorithm 1: Best-first search.

```
1  open.add($I$)
2  closed ← $\emptyset$
3  while open is not empty do
4      $s$ ← open.remove_best_f()
5      if $s \notin$ closed then
6          closed.add($s$)
7          if $s \models G$ then
8              return $s$ as solution
9          forall the child states $s'$ of $s$ do
10             if $s'$ is no dead-end then
11                 open.add($s'$)
12 return no solution found
```

Temporal Planning

In temporal planning, actions can run concurrently with one another and have duration, and the (typical) objective is to minimize plan makespan. For simplicity, we assume a temporal semantics similar to Temporal GraphPlan (TGP) (Smith and Weld 1999). That is, action conditions must hold through their entire duration and effects can only change at the end of an action (i.e., they cannot be changed by some other action running in parallel).

For our discussion, we assume planners with a forward state-space search space like that of Sapa (Do and Kambhampati 2003) and Temporal Fast Downward (TFD) (Eyerich, Mattmüller, and Röger 2009), where each state has a time stamp t that defines what point in time actions are added and an event queue (or agenda), that defines specific *decision epochs* where actions end. A special search operation called *advance time* increases t from its current value to the next (i.e., smallest) decision epoch and applies the effects of the action at that time point.[1]

Best-first search satisficing temporal planners use a range of techniques to speed up search. We cannot go over all of those here. It is important, however, to understand how these planners might go about minimizing makespan. We are concerned with two related questions (1) how to define the cost of search operations and (2) how to define g-values and h-values. To answer (1), we consider the objective function (i.e., makespan minimization). Any search operation that adds to makespan has a cost equal to the amount of makespan added, all others do not. From this decision, it follows that we should define g as the makespan of the partial state, g_m, and h as the remaining "makespan-to-go", h_m. This makes $f_m = f_{OF}$ (where we introduced f_{OF} as the objective function in the introduction).

Some planners consider g to be the time stamp of a state, which we call g_t, and the remaining makespan from that point to be estimated by a heuristic h_t (Haslum and Geffner 2001). When summed, these turn out to be equal values, i.e., $g_t + h_t = g_m + h_m$. Other planners (such as Sapa (Do and Kambhampati 2003)) have an evaluation function that does not return makespan values.[2] Since no known comprehensive studies have been conducted on various definitions of f for temporal planning, it is difficult to say what effect this has on final plan makespan. However, as we pointed out earlier, some evidence in cost-based planning points to a negative effect (Richter and Westphal 2010).

g-value Plateaus

A g-value plateau is a region of the search space where the evaluation of states with respect to g does not change from parent to child. In state-space temporal planners, as highlighted by Benton et al. (2010), g-values can occur in abun-

[1]Note that some forward state-space temporal planners do not use this exact method for search (c.f., Coles et al. (2009)), but we frame our discussion in this way since the planner we use for our experiments (Temporal Fast Downward) does.

[2]Also note that the Temporal Fast Downward planner uses g_t as its g-value, and a heuristic on the sum of action durations.

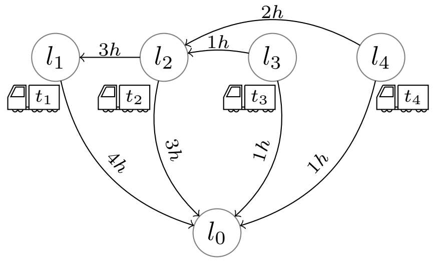


Figure 1: Layout for the example problem.

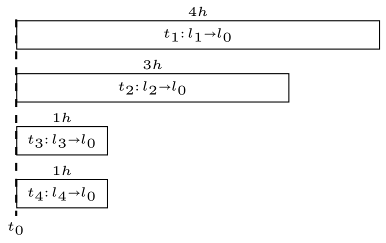


Figure 2: A makespan-optimal plan for the example.

dance and can cause problems for best-first search. To see this, let us look at an example.

Example: Consider the simple problem shown in Figure 1 of moving trucks in a logistics domain. We have 5 locations $\{l_0, \ldots, l_4\}$ all connected in the following way: Driving from location l_1 to l_0 takes 4 hours, from l_2 to l_0 takes 3 hours, and from l_3 or l_4 to l_0 takes 1 hour. We can also drive from l_4 to l_2 in 2 hours, from l_3 to l_2 in 1 hour, and from l_2 to l_1 in 3 hours. Trucks $t_1, \ldots, t_4$ start respectively at $l_1, \ldots, l_4$. The goal is to move all the trucks to l_0. Each *drive* action has the condition that the truck must be at the appropriate location and the effect that the truck is at its destination. An optimal solution to this planning problem is shown in Figure 2.

Given that every optimal solution requires that, at time t_0, we begin driving t_1 from l_1 to l_0, let us assume a planner is expanding the search state with this action. In this case, every child that drives one of the other trucks (and the advance time operator) has an equal g-value. In fact, we would end up with a fairly expansive g-value (and f-value) plateau where we must try every possible way of adding the various *drive* actions across all decision epochs until we reach a plan with (an optimal) makespan of 4 hours.

Useless Search Operations

Useless operators are defined by Wehrle, Kupferschmid, and Podelski (2008) as those operators that are unnecessary for finding optimal solutions from a state s. In sequential classical planning with unit costs, you can further formalize the notion of useless operators by taking $d(s)$ as the optimal distance to a goal state from s and $d^{\bar{o}}(s)$ as the optimal distance to a goal state without an operator o (i.e., $d(s)$ is a perfect heuristic and $d^{\bar{o}}(s)$ is a perfect heuristic without the operator o). If $d^{\bar{o}}(s) \leq d(o(s))$ then o is a useless operator. To understand why consider that if $d^{\bar{o}}(s) \leq d(o(s))$, then it must also be the case that $d(s) \leq d(o(s))$. Since this is true if and only if an there are no optimal plans that start from s, o must be useless.

Unfortunately, this definition is not easily transferable to planning for makespan minimization (at least not in planners like TFD) where we have zero-cost search operations. Using the original definition, search operations would be considered useless when they were not (e.g., when adding a useful operator that runs parallel to an optimal plan we would get no increase in makespan). Instead, we can give a weaker criterion for uselessness.

Definition 1. *An operator o is* guaranteed *makespan-useless when $d_m^{\bar{o}}(s) < d_m(o(s))$, where $d_m(s)$ returns the remaining makespan of a plan.*

This definition lets some useless search operations to fall through the cracks, but it catches operators that will strictly increase makespan when added to the plan.

This idea can be extended to any heuristic, such that $h^{\bar{o}}(s)$ represents running the heuristic without the operator o from state s. Wehrle, Kupferschmid, and Podelski (2008) call operators declared useless by $h^{\bar{o}}(s) \leq h(o(s))$ *relatively useless operators* in the sequential classical planning setting. We use $h_m^{\bar{o}}(s) < h_m(o(s))$ to define possibly *relatively makespan-useless* operators.

Useful Search Operations

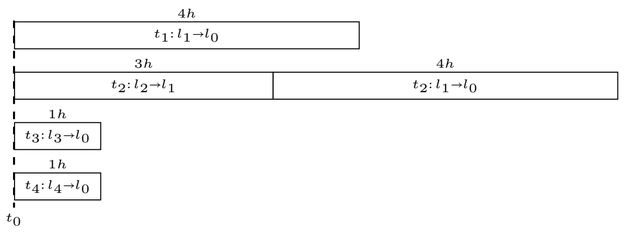
In contrast with a useless search operator, which should never be explored, a *useful* search operator is one whose absence would lead to worse plan quality. We can find a useful operator by using the same method for finding useless operators. In other words, operator o is *useful* in a state s if $d^{\bar{o}}(s) > d(o(s))$. We say the state $o(s) = s'$ is useful if the operator generating it is useful. In this section we discuss how we can use this notion to enhance satisficing planners using best-first search.

Integrating Usefulness with Satisficing Planning

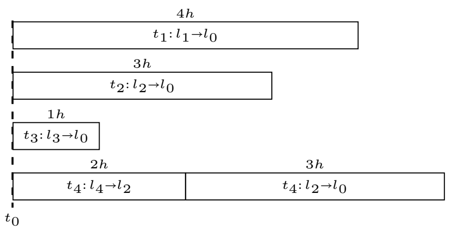
To see how we might use the concept of operator usefulness, let us revisit the example from Figure 1. Every optimal plan will need to include the operator that drives t_1 from l_1 to l_0, starting from time 0. Therefore, as before, we optimistically assume that the search algorithm finds (and chooses to expand) a state s_e that already includes this operator. Otherwise we make no assumptions on the definition of g- or h-values or what order states would be expanded in during a planner's regular (perhaps modified) best-first search.

Consider the possibility of using operator usefulness here. Let us assume for a moment that we can find d_m and $d_m^{\bar{o}}$ (i.e., the perfect heuristic on makespan). We want to find *guaranteed makespan-useful* operators, or operators o where $d_m^{\bar{o}}(s) > d_m(o(s))$. In our example on the state s_e, we would find the operators depicted in Figure 3 as guaranteed makespan-useful. Again, we cannot say whether an operator whose absence does not change makespan is useful or useless.

Since the guaranteed makespan-useful search operations are within a makespan g-value plateau, they are locally without cost according to the makespan objective. We would



(a) Without t_2: l_2→l_0



(b) Without t_4: l_4→l_0

Figure 3: Plans resulting from removing useful operators.

like to greedily seize the opportunity to expand the best possible search operator, keeping in mind that, since we have no guarantees on search behavior, the possibility of expanding a certain node may never present itself again. The idea is to generate a short "tunnel" in areas of original best-first search that provide zero-cost search operations according to the objective function (which is not necessarily the evaluation function used for nodes).

Notice that some operators are more useful than others. For instance, if we are disallowed from driving t_2 from l_2 to l_0, the makespan of the plan increases by 3 hours. In contrast, if we cannot drive t_4 from l_4 to l_0 the makespan increases by just 1 hour. Looking at Figure 3 again, indeed driving t_2 from l_2 to l_0 is the most useful operator, as we lose the most time when we do not include it. Therefore, we would want to choose this operator before all others to include in the plan.

More formally, at a state s, we can define the makespan *heuristic degree of usefulness* of an operator o as $\upsilon^o(s) = h_m^{\bar{o}}(s) - h_m(o(s))$. With this information, we can create a new search algorithm that interleaves local search decisions using degree of usefulness on makespan g-value plateaus (i.e., plateaus on g_m) and any best-first search strategy. We show how the interleaving would work when combined with a best-first search strategy in Algorithm 2

Up to line 12, the algorithm follows regular best-first search. At that point, generated child states are split from the regular search and placed into a υ*list* if they are on a g_m-value plateau and are possibly *makespan-useful*. Otherwise, they are placed in the regular *open* list. On line 18, the algorithm begins the local decision stage. If there are any nodes in the υ*list* (and there exists at least one node that has a value unequal to the others), the most useful node is removed and expanded, its child states placed directly into the *open* list. Finally, the rest of the υ*list* contents are placed into the *open* list. On the next iteration on line 4, the υ*list* is cleared.

Algorithm 2: Local decisions on degree of usefulness interleaved with best-first search for temporal planning.

```
1  open.add($I$)
2  closed ← $\emptyset$
3  while open is not empty do
4      $\upsilon$list← $\emptyset$
5      $s$ ← open.remove_best_f()
6      if $s \notin closed$ then
7          closed.add($s$)
8          if $s \models G$ then
9              return $s$ as solution
10         forall the child states $s'$ of $s$ do
11             if $s'$ is not dead-end then
12                 if $g_m(s) = g_m(s')$ and $\upsilon^o(s) \leq 0$ then
13                     $\upsilon$list.add($s'$)
14                 else
15                     open.add($s'$)
16         if $\upsilon$list is not empty &
17           $\exists s_1, s_2 \in \upsilon list$: $o_1(s) = s_1$,
18            $o_2(s) = s_2$ & $\upsilon^{o_1}(s) < \upsilon^{o_2}(s)$ then
19             $s''$ ← $\upsilon$list.remove_best_$\upsilon^o$
20             closed.add($s''$)
21             if $s \models G$ then
22                 return $s$ as solution
23             forall the child states $s'''$ of $s''$ do
24                 if $s'''$ is not dead-end then
25                     open.add($s'''$)
26         open ← open ∪ $\upsilon$list
27 return no solution found
```

Evaluation

We have shown how to integrate our strategy for making local decisions into best-first search. In this section, we evaluate this approach in the planner Temporal Fast Downward (TFD) (Eyerich, Mattmüller, and Röger 2009). The best-first search in TFD uses a modified version of the context-enhanced additive heuristic (Helmert and Geffner 2008) that sums the durations as operator costs, meaning the heuristic captures a sequential view of actions. With this heuristic, h_{dur}^{cea}, the planner uses a state's timestamp as its g-value (i.e., g_t) for a final $f = g_t + h_{dur}^{cea}$. Its search strategy uses preferred operators as defined in its classical planning counterpart, Fast Downward (Richter and Helmert 2009). TFD is an *anytime* planner, searching until it exhausts the search space or it reaches a timeout limit. It also reschedules solutions after it finds them in an attempt to minimize makespan (so a sequential plan may be rescheduled to a highly parallel one).

To detect possibly makespan-useful operators, we used the heuristic produced by Benton et al. (2010) for TFD in their original study of g-value plateaus. It uses a modified version of h_{dur}^{cea} to capture causal and temporal constraints between actions, detected during the extraction of a relaxed

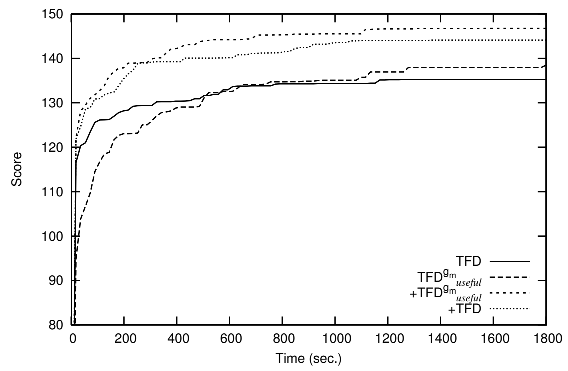


Figure 4: Anytime behavior showing the change in IPC score as time progresses.

plan. These constraints are encoded in a Simple Temporal Network (STN). The makespan of the schedule produced by the solution to the STN is returned as a heuristic estimate.

We implemented Algorithm 2 into TFD for a planner we call TFD$^{g_m}_{useful}$. Additionally, we generated a two-phase version of the planner. This variant employs *lazy evaluation* in the original TFD planner in a first phase. This means each state's parent heuristic is used (rather than its own value). After a solution is found (or 8 minutes has passed, whichever comes first) the planner restarts into a second phase using TFD$^{g_m}_{useful}$. It turns out that, using lazy evaluation, TFD tends to find (lower quality) solutions fast. The search uses the plan found in the first phase (if any) to prune the search space of the second phase (which focuses on finding higher quality plans using the useful actions technique). This is similar to the strategy employed by LAMA, though that planner restarts with different weights in a weighted A* search (Richter and Westphal 2010). We call this version +TFD$^{g_m}_{useful}$. Finally, to help discover the benefits granted by the two-phased search, we implemented +TFD, which uses TFD with lazy evaluation until a solution is found (or 8 minutes has passed) then switches to TFD without lazy evaluation.

Along with the original TFD, we tested all these variants on the temporal domains from the 2008 International Planning Competition (IPC-2008) (except *modeltrain*, as in our tests TFD is unable to solve more than 2 problems in this domain, depending on its settings). This benchmark set is known to contain plateaus on g_m (Benton et al. 2010).[3] The experiments were run on a 2.7 GHz AMD Opteron processor, with a timeout of 30 minutes and a memory limit of 2 GB.

Recall that, to find whether a search operator is useful requires a heuristic to be run on both the parent (without considering the operator) and the child (using the normal heuristic). Let us analyze about how many times we are running heuristics in TFD$^{g_m}_{useful}$ and the second phase of +TFD$^{g_m}_{useful}$ as compared to the original TFD. Assume n nodes are generated by the search with e nodes expanded and a branching factor of b. We will always run h^{cea}_{dur} on every child node, so TFD would run this heuristic n times. Assuming a percentage of states in g-value plateaus, r, we must run the modified (and slower) makespan heuristic in TFD$^{g_m}_{useful}$ about $b \times e \times r + n \times r$ times. Therefore, we may expect a slowdown in finding solutions from TFD to TFD$^{g_m}_{useful}$.

To help see whether this greatly effects the search, we measured the quality and coverage across all benchmarks over time using the scoring system from the IPC-2008, which captures a notion of coverage and quality by summing the values of all $Q*/Q$, where $Q*$ is the makespan of the best known solution for a problem and Q is the makespan found by the planner for the problem (unsolved instances get a score of 0). The results of this appear in Figure 4. As expected, the scores for TFD$^{g_m}_{useful}$ start lower than TFD, but at about 10 minutes dominate it. On the other hand, +TFD$^{g_m}_{useful}$ dominates +TFD very quickly. Given that +TFD and +TFD$^{g_m}_{useful}$ use the same initial search (i.e., TFD with lazy evaluation), this provides an indication that the *useful actions* lookahead technique used in combination with the two-phase search is more effective than applying the useful actions technique alone. Hence, with a computational time limit we have a greater likelihood of finding a better solution if we stop +TFD$^{g_m}_{useful}$ than +TFD (and after around 10 minutes, the same holds true for TFD$^{g_m}_{useful}$ vs. TFD).

The final IPC scores for all four variations are found in Table 1. Notice that +TFD$^{g_m}_{useful}$ is better than the other approaches across problems. Looking carefully across the do-

Domain	TFD	+TFD	TFD$^{g_m}_{useful}$	+TFD$^{g_m}_{useful}$
Crewplanning	22.44	29.64	22.57	29.66
Elevators	13.88	14.16	15.40	15.79
Openstacks	25.79	27.22	27.81	28.78
Parcprinter	8.73	8.74	8.47	8.50
Pegsol	28.73	28.73	28.81	28.81
Sokoban	10.93	10.88	10.79	10.79
Transport	5.06	5.06	4.68	4.68
Woodworking	19.72	19.71	19.93	19.73
Overall	**135.28**	**144.13**	**138.48**	**146.75**

Table 1: Results using the measure from IPC 2008.

Domain	TFD	+TFD	TFD$^{g_m}_{useful}$
Crewplanning	(24) 8%	(30) 0%	(24) 7%
Elevators	(19) 7%	(19) 6%	(20) 3%
Openstacks	(30) 13%	(30) 6%	(30) 4%
Parcprinter	(13) -4%	(13) -4%	(13) 1%
Pegsol	(30) 0%	(30) 0%	(30) 0%
Sokoban	(11) -1%	(11) -1%	(11) 0%
Transport	(6) 9%	(6) 9%	(6) 0%
Woodworking	(26) 7%	(26) 7%	(25) -1%
Total	(159) **6%**	(165) **3%**	(159) **3%**

Table 2: Percentage improvement made by +TFD$^{g_m}_{useful}$, comparing plans solved by both approaches (number of plans solved by both approaches shown in parenthesis).

[3]We used the *elevators-numeric* and *openstacks-adl* variants for our results on the respective domains following the IPC-2008 rules by using the domain variant that all our planner versions did best in.

mains, we can see the individual increases in IPC score are fairly minor between TFD$^{g_m}_{useful}$ and +TFD$^{g_m}_{useful}$ in all but the *Crewplanning* domain, for which the lazy heuristic evaluation scheme tends to work quite well. +TFD shows that this improvement appears to come from using lazy evaluation for finding a first solution. As far as the IPC metric is concerned, we get better values when using useful actions (i.e., between TFD vs. TFD$^{g_m}_{useful}$ and +TFD vs. +TFD$^{g_m}_{useful}$) in *Crewplanning*, *Elevators*, *Openstacks*, *Pegsol*, and *Woodworking*.

To get a better view of the plan-for-plan quality improvements, we compare direct makespan decreases (i.e., quality increases) on the problems solved by TFD, +TFD, TFD$^{g_m}_{useful}$ and +TFD$^{g_m}_{useful}$. Table 2 shows the percentage improvement on each domain as compared with +TFD$^{g_m}_{useful}$ for problems solved by both planners. It is evident that +TFD$^{g_m}_{useful}$ gives better quality plans over the usual TFD search in most of the domains (all but *Parcprinter* and *Sokoban*, for which there is a small decrease in plan quality). We get the largest increase in *Openstacks*, a domain where sequential plans are often easy to find (at least with TFD) but less easy to reschedule into parallel ones.

Related Work

For this work, we sought to give some degree of remedy for maintaining quality while employing scalability techniques for searching over zero-cost search operators. As mentioned earlier, others have noticed this problem. Richter and Westphal (2010) provide an investigation into the issue of variable-cost operators in the best-first search classical planner LAMA, where they notice better scalability (but lower quality) when ignoring search operator costs completely versus taking costs into account. The work most related to ours was done by Benton et al. (2010), where they study the problem of g-value plateaus that occur in temporal planning in-depth. Cushing, Benton, and Kambhampati (2011) also study the broader problem of searching using variable-cost operators, while Thayer and Ruml (2009) propose a search method for bounded-suboptimal search with variable operator costs.

Our approach for tackling the problem is related to two common enhancements to best-first search for satisficing planning: (1) methods that permanently or temporarily prune search spaces using heuristics and (2) methods that perform lookaheads. The prime examples of pruning search using heuristics are *helpful actions* (Hoffmann and Nebel 2001) and *preferred operators* (Richter and Helmert 2009). These techniques find a relaxed solution to a planning problem and exclude all operators that fail to achieve values which are part of it, marking included operators as helpful or preferred.

Interestingly, the idea of *helpful actions* was extended upon to generate a lookahead technique in the planner YAHSP (Vidal 2004). This planner attempts to use the entire relaxed solution to perform a lookahead. That is, it tries to use each action in the relaxed plan, applying them in order to reach deeper into the search space. It turns out that this approach works quite well to increase solution coverage across benchmarks, but does not do as well in terms of the quality of the solutions found. Others have expanded on this idea by attempting to find relaxed plans that would perform better for lookahead (Baier and Botea 2009), and still others have used a more of a local search approach for probing the search space based on solutions to relaxed plans (Lipovetzky and Geffner 2011).

Finding whether an operator is useful or useless also has connections to landmarks (Hoffmann, Porteous, and Sebastia 2004). It requires that we run the heuristic without an operator o at state s. For most reasonable heuristics, if we get a value of *infinity* when excluding o from its computation, then o is a landmark to finding a solution from s.

Summary and Future Work

We have shown a novel method for enhancing heuristic best-first search for satisficing planning. Our approach focuses on finding plans of higher quality by exploiting the notion of *usefulness* on the objective function f_{OF}. The idea is to turn the search toward solutions of higher quality, even when the node evaluation function differs from f_{OF}. In particular, our technique performs a lookahead on useful operators over g-value plateaus in temporal planning, where the objective is to minimize makespan. We applied this method to the best-first search of the planner Temporal Fast Downward (TFD). Our experiments on the variants TFD$^{g_m}_{useful}$ and +TFD$^{g_m}_{useful}$ show that using this method, we can achieve better quality solutions.

For future work, we are investigating how to apply similar approaches to cost-based classical planning, where often the biggest problem is not g-value plateaus, but variable cost operators. Further, we are considering ways of exploiting the connection to *local* landmarks to enhance the search further.

Acknowledgements We thank Robert Mattmüller for discussions on this work. This research is supported in part by ONR grants N00014-09-1-0017 and N00014-07-1-1049, the NSF grant IIS-0905672, and by DARPA and the U.S. Army Research Laboratory under contract W911NF-11-C-0037.

References

Baier, J. A., and Botea, A. 2009. Improving planning performance using low-conflict relaxed plans. In *Proc. ICAPS 2009*.

Benton, J.; Talamadupula, K.; Eyerich, P.; Mattmüller, R.; and Kambhampati, S. 2010. G-value plateaus: A challenge for planning. In *Proc. ICAPS 2010*.

Coles, A. J.; Coles, A. I.; Fox, M.; and Long, D. 2009. Extending the use of inference in temporal planning as forwards search. In *ICAPS*.

Cushing, W.; Benton, J.; and Kambhampati, S. 2011. Cost-based satisficing search considered harmful. In *Proceedings of the Workshop on Heuristics in Domain-independent Planning*.

Do, M. B., and Kambhampati, S. 2003. Sapa: A multi-objective metric temporal planner. *JAIR* 20:155–194.

Eyerich, P.; Mattmüller, R.; and Röger, G. 2009. Using the context-enhanced additive heuristic for temporal and numeric planning. In *Proc. ICAPS 2009*, 130–137.

Haslum, P., and Geffner, H. 2001. Heuristic planning with time and resources. In *Proc. ECP 2001*, 121–132.

Helmert, M., and Geffner, H. 2008. Unifying the causal graph and additive heuristics. In *Proc. ICAPS 2008*, 140–147.

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.

Lipovetzky, N., and Geffner, H. 2011. Searching for plans with carefully designed probes. In *ICAPS*.

Pearl, J. 1984. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley.

Richter, S., and Helmert, M. 2009. Preferred operators and deferred evaluation in satisficing planning. In *Proc. ICAPS 2009*, 273–280.

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

Smith, D., and Weld, D. 1999. Temporal planning with mutual exclusion reasoning. In *Proc. IJCAI 1999*, 326–337.

Thayer, J. T., and Ruml, W. 2009. Using distance estimates in heuristic search. In *ICAPS*.

Vidal, V. 2004. A lookahead strategy for heuristic search planning. In *Proc. ICAPS 2004*, 150–159.

Wehrle, M.; Kupferschmid, S.; and Podelski, A. 2008. Useless actions are useful. In *Proc. ICAPS 2008*, 388–395.

Exploiting Problem Symmetries in State-Based Planners

Nir Pochter
School of Eng. and Computer Science
The Hebrew University of Jerusalem
Jerusalem, Israel
nirp@cs.huji.ac.il

Aviv Zohar
Microsoft Research
Silicon Valley
Mountain View, CA
avivz@microsoft.com

Jeffrey S. Rosenschein
School of Eng. and Computer Science
The Hebrew University of Jerusalem
Jerusalem, Israel
jeff@cs.huji.ac.il

Abstract

Previous research in Artificial Intelligence has identified the possibility of simplifying planning problems via the identification and exploitation of symmetries. We advance the state of the art in algorithms that exploit symmetry in planning problems by generalizing previous approaches, and applying symmetry reductions to state-based planners. We suggest several algorithms for symmetry exploitation in state-based search, but also provide a comprehensive view through which additional algorithms can be developed and fine-tuned. We evaluate our approach to symmetry exploitation on instances from previous planning competitions, and demonstrate that our algorithms significantly improve the solution time of instances with symmetries.

1 Introduction

Classical planning problems are notorious for the exponentially large number of states that need to be navigated in order for a solution to be found. Such problems, however, often contain symmetries. In that case, many of the different states in the search space are in fact symmetric to one another, with respect to the task at hand, and the result of searching through one state will inevitably be equivalent to searches through all of its symmetric counterparts. Though detecting and exploiting such symmetries would shrink the search space, the actual exploitation of symmetries is made difficult by the fact that symmetry detection is potentially complex. All currently known algorithms for the highly related problem of finding graph automorphisms work in exponential time, in the worst case. The situation is worsened by the fact that the graph on which we wish to detect automorphisms is the huge search space that the planning problem defines; it does not typically fit into memory.

The idea of exploiting symmetries has appeared in model checking, e.g., in work by Emerson and Sistla (1996), and in the context of constraint satisfaction problems, beginning with a seminal paper by Puget (1993). The most common approach uses additional constraints to break existing symmetries (e.g., see (Walsh 2007)) and reduce the search space to a more manageable size. Planning problems can be reduced to various constraint satisfaction problems and to propositional logic formulas. Some have tried to use this approach to apply symmetry reduction in search (Miguel 2001; Rintanen 2003). Rintanen adds symmetry-breaking constraints to the propositional logic representation of transition sequences derived from planning problems to remove symmetries from different points in the sequence. However, state-of-the-art planners usually employ a state-based approach combined with heuristic search. It is unclear how to directly translate knowledge in symmetry exploitation in earlier planning systems and CSPs to improvements in today's most efficient planners.

A previous line of work by Fox and Long (1999; 2002) has shown how to prune symmetries in planning via modifications of their GraphPlan-based algorithm (which is not state-based). Another approach is to use symmetries to improve heuristics instead of pruning. Porteous, Long and Fox (2004) do so for "almost-symmetries" in the FF planner.

Our own work can be seen as a continuation of this line of research, and its extension to the more ubiquitous state-based planners. Among our contributions, we present a framework for the understanding of symmetry exploitation from the state-based point of view.

The essence of our approach to symmetry exploitation is made up of two main steps. First, we explain how knowledge of symmetries in the search space can be useful to prune some of the search done by state-based algorithms such as A*; for this, we employ techniques similar to the canonicalization used by Emerson (1996) as well as pruning of shallower symmetries similar in spirit to Rintanen (2003). We then explain how to deduce the existence of relevant symmetries in the search space from the smaller problem description (in a way similar to Cohen *et. al.* (2006) which have done so for CSPs). Since our approach works at the core of the search algorithm by pruning branches of the search, it is completely independent of any heuristic that may be applied to guide that search, and can assist whenever the planning problem contains some amount of symmetries.

1.1 Example

We begin by presenting an example that has appeared in the International Planning Competition (IPC), and has been examined in the context of symmetries by Fox and Long: the gripper domain. The gripper domain consists of two rooms, n numbered balls, and a robot with two arms, or grippers. The robot can pick up a single ball in each of its grippers, move between rooms, and put balls down. A problem instance is usually set up in the following manner: all balls and the robot begin in one room, and the goal is to move the balls to the other room.

The problem seems trivial to solve (at least for humans), but is surprisingly difficult for planners. The difficulty stems from the fact that the balls in the problem are effectively interchangeable, but are not considered as such by conventional planning algorithms. The balls may be picked up in any of $n!$ possible orders, each of which will lead to a different plan, and planners may potentially try all such orders. The reason for this was well-explained by Helmert and Röger (2008), where it is shown that any A* search with an imperfect heuristic is bound to explore states on multiple shortest paths to the goal. Our objective is to identify during search that some states (and therefore the plans going through them) are symmetric to one another, and can thus be pruned; e.g., the state in which the robot holds one ball in its right gripper (and the left is free) is symmetric to the state in which the ball is held in the left gripper (with the right one free). Only one of the two states needs to be explored in order to find an optimal plan. This means that on domains in which symmetries occur, the heuristic will no longer have to be perfect in order to avoid opening most of the states.

2 Preliminaries

We consider planning in the SAS$^+$ formalism (Bäckström and Nebel 1995). Tasks that are represented in STRIPS or PDDL can be automatically converted to SAS$^+$(Helmert 2006), which is sufficiently general, and is used in Fast-Downward—the planner used in our implementation. The algorithms we present do not take into account action costs and conditional operators, but these, and other additional structures in the planning language, can be accounted for with only minor modifications.

2.1 Definitions for Planning Problems

Definition 1 *A planning task is a tuple $(V, O, s_0, S_\star)$:*

- *$V = \{v_1, ..., v_n\}$ is a finite set of state variables, each with a finite domain D_v.*[1] *A fact is a pair $\langle v, d\rangle$ where $v \in V$ and $d \in D_v$. A partial variable assignment is a set of facts, each with a different variable. A state is a partial variable assignment defined over all variables V.*
- *O is a set of operators o specified via $\langle pre(o), eff(o)\rangle$, both being partial variable assignments to V.*
- *s_0 is the initial state.*
- *$S_\star$ is a partial variable assignment which is the goal.* [2]

An operator $o \in O$ is applicable in a state s iff $pre(o) \subseteq s$.

The states of a problem naturally define a search space in the form of a state transition graph—a directed multigraph (S, E), where S is the set of all states, and E, the set of edges, contains a directed edge from state s to state s' exactly for every operator that is enabled at s and leads to s'. [3]

An instance of a planning problem is then fully specified by designating an initial state s_0 and goals $S_\star$ with the objective of finding the shortest possible path (i.e., a plan) between s_0 and some $s_\star \in S_\star$.[4]

2.2 Graph Automorphisms, Permutation Groups

The symmetries in the problem are manifested as automorphisms in the state transition graph. We briefly present some relevant definitions from Group Theory.

An automorphism of a graph (S, E) is a permutation $\sigma : S \to S$ of the vertices of the graph that maintains the structure; i.e., for every two vertices $s_1, s_2 \in S$, we have that $(s_1, s_2) \in E$ iff $(\sigma(s_1), \sigma(s_2)) \in E$. The automorphisms of (S, E), with the composition action, constitute a *group* denoted $Aut(S, E)$. Unless otherwise specified, we will denote by G some subgroup of $Aut(S, E)$ (and refrain from using it to denote graphs).

Definition 2 *$\Sigma = \{\sigma_1, \ldots, \sigma_k\}$ is said to* generate *a finite group G, if G is exactly all the permutations that are obtained by repeatedly composing elements of Σ.*

Finding a generating set for the automorphisms of a graph is harder than solving the graph isomorphism problem (for which no polynomial-time algorithm is currently known). See the paper by Luks (1993) for an overview of complexity results relating to permutation groups. Generating sets are usually found using brute force backtracking search with some specialized pruning techniques that allow the set to be found surprisingly quickly in practice.

Definition 3 *The* orbit *of a vertex s with respect to some subgroup G is denoted by $G(s)$ and is simply the set of vertices to which elements in G map s. $G(s) = \{\sigma(s) \mid \sigma \in G\}$.*

Given a generating set for G, the orbit of vertices can be computed in polynomial time. We will be especially interested in subgroups of $Aut(S, E)$ that fix a certain vertex or a set of vertices. These are defined below:

Definition 4 *The* point-stabilizer *of a vertex s with respect to G, denoted G_s is a subgroup of G that contains all the permutations that fix s. $G_s = \{\sigma \mid \sigma \in G, \sigma(s) = s\}$.*

Definition 5 *The set-stabilizer of a set $X \subset S$ with respect to G is the subgroup of G that maps elements of X only to X. $G_X = \{\sigma \in G \mid \forall x \in X \;\; \sigma(x) \in X\}$.*

Finding generators for the stabilizer of a single vertex can be done in polynomial time. In contrast to that, finding generators for the set-stabilizer group is a more difficult problem, that is known to be harder than graph isomorphism.

[1]To simplify definitions, we assume unique names for the values, so given a value it is known to which variable it corresponds.

[2]We will slightly abuse notation and use $S_\star$ to denote the set of goal states.

[3]Equivalent operators may exist, and so there can be more than one edge between two states.

[4]We often refer to states in the planning problem as vertices in the transition graph, and vice versa.

3 Symmetries in the Search Space

Our goal in this section is to show how the symmetries in the transition graph can be used to prune the search. The first observation we make is that stabilizing vertices can help when automorphisms are applied to paths.

Observation 1 *Let (S, E) be a graph with a group G of automorphisms, and let $(s_0, s_1, \ldots, s_\star)$ be a sequence of vertices that forms a path from s_0 to $s_\star$. If we simply apply some $\sigma \in G$ to the path, we will have another path of equal length: $(\sigma(s_0), \sigma(s_1), \ldots, \sigma(s_\star))$. This path, however, may not begin at s_0, and thus may be of little value. Instead, we may choose to fix a given vertex s_i along the path via the point-stabilizer subgroup G_{s_i}, and apply the transformation only to some part of the path. From the application of any $\sigma \in G_{s_i}$ to the suffix $s_i, \ldots, s_\star$, we get a path $(s_0, s_1, \ldots, s_{i-1}, \sigma(s_i), \sigma(s_{i+1}), \ldots, \sigma(s_\star))$ which is connected because $s_i = \sigma(s_i)$. This path does begin at s_0.*

This rather basic observation will form the basis of our modification of state-based planning algorithms, and specifically ones that are based on A*. To use it, we will need to be able to perform three basic tasks: to find $G \subset Aut(S, E)$, to compute the stabilizer G_s of a state s, and to determine for two states s_1, s_2 if $s_1 \in G(s_2)$.

We discuss how to perform these three tasks in the next section, and for now present how they may be used in a planning problem. As we are constrained for space, we assume the reader is intimate with the workings of the A* algorithm, and do not specify it. In addition, we do not present full proofs of the correctness of our algorithms, which follow along similar lines to the proofs of A*'s correctness.

3.1 Searching in Orbit Space

Observation 1 suggests that in order to ensure that symmetric paths always connect to s_0, we may stabilize any vertex along a path that starts there. One vertex will surely be along any path from s_0, and that is s_0 itself. A simple approach would then be to restrict ourselves to the group G_{s_0}.

Our goal is to find a shortest path from s_0 to any of the nodes in $G_{s_0}(S_\star)$. Once such a path is found to some node $\sigma(s_\star)$, we can translate it to a path from start to goal by simply applying σ^{-1} on the entire path. To gain the most benefit from our search procedure, we do not conduct the procedure on the original search space, but rather consider orbits of the search space as nodes in our search, in a manner similar to that of Emerson and Sistla (1996). For this purpose we define a graph with vertex set S_{orb}, the set of orbits of vertices in the transition graph with respect to the group G_{s_0}. The edge set that we will use is E_{orb} which will contain an edge from one orbit to another iff there exists an edge between nodes from these orbits in the original graph. $S_{orb} = \{G_{s_0}(s) | s \in S\}$; $E_{orb} = \{(o_1, o_2) \in S_{orb} \times S_{orb} \, | s_1 \in o_1, \ s_2 \in o_2, \ (s_1, s_2) \in E\}$

We can conduct a search in (S_{orb}, E_{orb}) and the resulting path will translate to a plan in the original transition graph. While we wish to run a search on the graph of orbits, we really only know how to search in the original search space. Our heuristic function may also only apply to states and not to orbits. Therefore, the practical search algorithm simulates a search on the orbits through a search in the original space, by using a single state from each orbit as its representative:

1. Before the search, find generators for the group G_{s_0}.
2. Whenever generating a node s_2, search for a previously generated node s_1 such that $s_2 \in G_{s_0}(s_1)$. If such a node was found, treat s_2 exactly as if it was s_1, otherwise, s_2 represents a newly found orbit.
3. Stop the search when expanding a node from $G_{s_0}(s_\star)$.

Alas, the stopping condition of the algorithm we present is not guaranteed to work with all heuristic functions. In fact, since we stop the search at some node from $G_{s_0}(s_\star)$ that is not necessarily a goal state, our algorithm must assure us that when this node is expanded, its f-value is minimal. For general admissible heuristics it is not guaranteed that $h(\sigma(s_\star)) = 0$, but an admissible heuristic that will give all nodes in $G_{s_0}(s_\star)$ the same h-value of 0, will work.

Notice that the above algorithm will never expand the children of a node that is not from a new orbit. It will be matched to a previously generated node, and the A* algorithm will treat it as that node (it may update its f-value if it has a lower one than the previously-seen representative for the orbit, but it does not need to be expanded).

While many heuristics that will give all goal-symmetric states the same value do exist, we would like an algorithm that would function with admissible heuristics that are not necessarily symmetric. In addition, as we will later see, the need to check the halting condition of the algorithm for each node (trying to see if it is in the orbit of the goal) requires a great deal of computation. Our next modification (which is also the one used in our experiments) will allow us to simplify this check, and to use heuristics that are just admissible, by using a somewhat restricted set of symmetries.

3.2 Stabilizing the Goal

Our previous algorithm was not guaranteed to reach a goal state, which caused it to require complicated checks for the terminating condition and to use only goal-symmetric heuristics. To solve this issue, we work with the symmetry group $G_{s_0,S_\star}$ that stabilizes both the start, and the partial assignment $S_\star$. The modified algorithm is as follows:

1. Before the search, find generators for the group $G_{s_0,S_\star}$.
2. Whenever generating a state s_2, search for a previously generated state s_1 such that $s_2 \in G_{s_0,S_\star}(s_1)$. If such a state was found, treat s_2 exactly as if it was s_1, otherwise, s_2 represents a newly found orbit.
3. Stop the search when a goal state is expanded.

This algorithm is also effectively a search in the orbit-space of a smaller group, but the orbit of goal states contains only goal states, and so our halting condition is simplified. An additional benefit is that we can relax the search for matching states in step 2. If we fail to match a state to another one in its orbit, it will simply be added as a node in the search but will not change the minimal length of the path we end up finding. We can therefore do this matching in a heuristic manner (but we must make sure that no false positive matches are made). Our previous approach required exact matching in order to identify the goal state correctly.

3.3 Shallow Pruning of Symmetric States

Both of the previous algorithms have stabilized the start state and have thus limited the symmetries that were used. To show that other approaches may be possible, we will briefly exhibit another modification that can be implemented alone or combined with search in orbit-space that will work through somewhat different symmetry groups.

When expanding a certain state s during search in highly symmetric problems, we often find that we are generating a large number of child states, many of which are symmetric to one another. We then waste effort matching each of them to the orbits of previously generated states, and pruning them then. It would be ideal if we could use a shallow pruning technique to avoid generating some of these immediate children.

1. For each state s that is being expanded, compute the pointwise stabilizer of the goal and s itself: $G_{s,S_\star}$.
2. Choose one operator from each orbit in $G_{s,S_\star}$ to apply in s (given that it is enabled), and avoid applying the others.[5]
3. Stop the search once a goal state has been expanded.

The intuition behind the modification is as follows. If s is not itself on the shortest path to the goal, then failing to generate some of its children will not affect the path found in the search (if one of its children is on the shortest path, it will be a child of another node as well). If it is on the shortest path, then two symmetric actions will only lead to symmetric states (with respect to $G_{s,S_\star}$) and a path through one such state to the goal can be turned into a path through the other, as is demonstrated in Observation 1.

This form of symmetry pruning is somewhat costly as it requires many computations of the state stabilizer. It is very similar in spirit to the approach of Fox and Long (2002) which they term dynamic symmetries. In fact, from the state-based perspective, it is not the symmetry group that changes, but rather the subgroup of it that we exploit at any given point. The shallow pruning algorithm fails to remove many symmetric states from the graph when applied alone (symmetries between states that do not have a common parent will not be detected), but it can be combined with our second version of search in orbit space.

4 Symmetries in the Problem Description

Above, we have shown how to exploit symmetries to speed up search. We now explain how to detect such symmetries, and how to perform the basic computations we needed.

Our approach to symmetry detection is similar to the ones used in previous research. We detect symmetries in the description of the problem and infer through these about symmetries in the transition space, i.e., we identify places where variables, values and operators can be renamed (permuted somehow) so as to leave us with exactly the same problem as before. These syntactic symmetries in the problem description, which is much smaller than the search space,

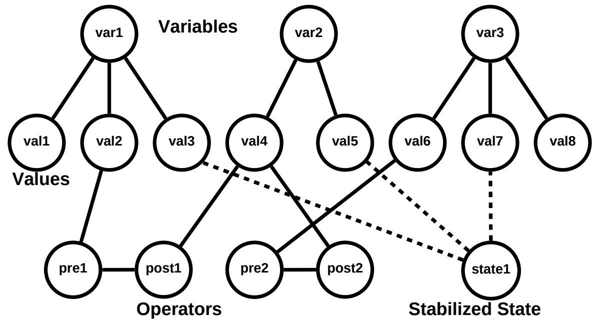


Figure 1: A problem description graph (PDG) with a stabilized state.

imply the existence of symmetries in the transition graph.[6] To find syntactic symmetries, we construct a graph that encodes the problem's structure, and use conventional group-theoretic algorithms to find generators for its automorphism group. While the algorithms to find the generators for the automorphism group run in worst-case exponential time, they are quite manageable for graphs of hundreds of thousands of vertices. These suffice to describe typical planning problems that challenge current planners.

Formally, we define an undirected graph for a planning problem, that we call the *Problem Description Graph* (PDG). The PDG has four types of vertices: there is a vertex for every variable, every value, and two vertices for each operator—one that represents the preconditions and one that represents the effects. The vertices are connected by edges as follows. We connect every variable vertex to the values that belong to it, every precondition and effect belonging to the same operator, every precondition vertex to the values in its precondition list, and finally, each effect vertex to the values in its effects list. Figure 1 illustrates such a graph.

We are specifically interested in the group G^{PDG} of automorphisms of the PDG that are restricted to map vertices of each type (variables, values, preconditions, and effects) only to vertices of the same type. Several tools solve this problem of colored-graph automorphism quickly. Our own implementation uses Bliss (Junttila and Kaski 2007).

Observation 2 *Every permutation $\pi \in G^{PDG}$ can be interpreted as a permutation operator on states. We denote this operation by $\hat{\pi} : S \rightarrow S$. It is defined in the following manner: $\hat{\pi}(s) = \{\langle \pi(v), \pi(d) \rangle \mid \langle v, d \rangle \in s\}$*

That is, if the value d is assigned to variable v in state s then the value $\pi(d)$ holds in variable $\pi(v)$ in state $\hat{\pi}(s)$. The edges between variables and their values in the PDG assure us that $\hat{\pi}(s)$ will be a legal state (all variables will be assigned exactly one value), as π is thus restricted to preserve the link between a variable and its values.

Proposition 1 *The operator $\hat{\pi}$ (as defined above in Observation 2 for any corresponding $\pi \in G^{PDG}$) is a proper permutation on states. Furthermore, $\hat{\pi} \in Aut(S, E)$.*

[5]The orbits of an edge in the transition graph is the set of edges to which it is mapped by the group action.

[6]Fox and Long (1999) advocate using "object symmetries" which use more restricted subgroups than those we detect, but may be found faster in practice. This alternative approach is compatible with our methods of exploitation.

The proposition holds because the edges in the PDG encode the entire structure of the planning problem and any (colored) automorphism of the PDG is equivalent to renaming the operators, variables and values in a way that preserves the semantics. A similar observation about the relation between syntactic symmetry and semantic symmetry has been made in the context of CSPs (Cohen et al. 2006). We do not provide the proof due to lack of space.

4.1 Stabilizing a Given State

We now show how to look for subgroups that stabilize a given state, using the PDG representation. Given a state s, we are interested in all $\pi \in G^{PDG}$ for which $\hat{\pi}(s) = s$.

Observation 3 *Using the definition of $\hat{\pi}$'s operation on states, we can see that in order for a state to remain unchanged under $\hat{\pi}$, it must be that π maps all values that hold in s onto themselves: $G_s = \{\hat{\pi} | \langle v, d\rangle \in s \rightarrow \langle \pi(d), \pi(v)\rangle \in s\}$. This means that in order to find G_s we just need to compute the set stabilizer (within G^{PDG}) of all values that hold in s.*[7]

We ensure that assigned values are mapped only into each other by adding a vertex to the PDG, connecting it only to assigned values (as depicted in Figure 1), and then computing generators for the automorphisms of this modified graph.

4.2 Locating States from the Same Orbit

Given a group G, and a state s, we wish to know if our search has already encountered a state in $G(s)$. One naive (and time-consuming) approach would be to iterate over all previously encountered states, and to attempt to compute $\sigma \in G$ that will perform the match. Each such comparison is in itself a computational problem (that is harder than Graph-Iso). We shall instead use a method that is commonly employed in computational group theory: the canonical form. We map each state s into another state $C(s) \in G(S)$ that acts as a representative for the orbit. $C_G()$ is a mapping such that: $C_G(s) = C_G(s')$ iff $\exists \sigma \in G : s = \sigma(s')$. One possible way to pick a representative state is to look for the one that has a lexicographically minimal representation. Using the canonical form, we are able to rapidly check if a new state is symmetric to one that has been previously encountered by keeping the canonical form of all states we encounter in some form of a hash-set (or any other data structure that lets us test membership in O(1) time). Since A* keeps such data sets of generated states anyway, we simply use the canonical form as the key instead of the state itself. Unfortunately, is is NP-hard to find the lex-minimal state (Luks 1993).

Approximate canonical forms If we wish to speed up the canonical-form computation for states, we can use a heuristic approach. Instead of finding the lexicographically smallest state in $G(s)$, we greedily search for one that is relatively small (by applying various permutations) and keep the best state we encounter. We may often find that instead of mapping all members of $G(s)$ to a single state, we map them to several ones that are local minima in our greedy search. In this case we do not reduce the search space as effectively, but we do it much faster.

[7]Partial assignments are stabilized in the same manner.

5 Experimental Evaluation

Symmetry exploitation has the potential to slow search if the savings it provides in the number of explored states are negligible in comparison to the additional running time it requires per state. We therefore conducted our experiments with the aim of checking if the pruning algorithm fulfills its purpose, i.e., if it makes the search faster on problems that contain enough symmetries, while not severely hurting performance in problems that do not have any.

Since shallow pruning requires calculating automorphisms that stabilize each state, it is too slow to use if the canonical state evaluations that it saves are very efficient. Our search in orbit space that was based on greedy canonicalization attempts proved so fast that we did not gain from additional shallow pruning (but slower approaches, e.g., exact matching, benefit a great deal).

We implemented our algorithm on the Fast Downward system (Helmert 2006), and compared between the running times of the planner with and without our modifications. Our tests were conducted using various heuristics and were done on domains from recent planning competitions (IPCs). For symmetry detection, we used Bliss (Junttila and Kaski 2007), a tool that finds generators for automorphism groups. All experiments were conducted on Linux computers with 3.06GHz Intel Xeon CPUs, using 30-minute timeouts. For all experiments we set a 2GB memory limit.

We used three heuristics for testing: first, LM-Cut (Helmert and Domshlak 2009), which is one of the best heuristic function known today; second, Merge-and-Shrink (Helmert, Haslum, and Hoffmann 2007), which is less informative, but has a different behavior as it spends more time in pre-processing and less during the heuristic calculation for each state; finally, we also tested using h^0 (blind search).

Table 1 shows the number of tasks solved by each heuristic, with and without the symmetry-pruning algorithm.[8] The biggest advantage was, as expected, in the gripper domain, where all heuristics failed to solve more than 7 tasks without our modification, and all of them solved all 20 tasks with it. Even though other domains contained relatively few symmetries, using symmetries for pruning still improved the results in some of those domains.

Table 2 shows the number of node expansions and search time for each heuristic in selected tasks, both with and without symmetry detection. Empty cells correspond to runs that exceeded the time and memory constraints. We chose to display a sample of instances from the gripper domain (where symmetries were abundant). Other domains had fewer symmetries, and we picked the mprime domain, where we did not solve additional instances, but did improve the running time of some tasks, as a typical example (we display the

[8]Because of space limitations, we did not present these numbers for domains in which the symmetry detection algorithm did not change the number of tasks solved, which are: airport, blocks, depot, drivelog, freecell, grid, logistics00, mystery, mprime, openstacks, pathway-noneg, rover and trucks.

	h^0		$h^{m\&s}$		h^{LM-cut}	
domain	reg	sym	reg	sym	reg	sym
gripper(20)	7	**20**	7	**20**	7	**20**
logistics98(35)	2	2	4	5	**6**	**6**
miconic-STRIPS(150)	50	50	53	56	**141**	**141**
pipesworld-notankage(50)	14	16	21	**22**	17	18
pipesworld-tankage(50)	9	13	14	**16**	10	13
psr-small(50)	48	49	**50**	**50**	49	49
satellite(36)	4	5	6	6	7	**9**
tpp(30)	5	6	6	6	6	**7**
zenotravel(20)	7	8	11	11	12	**13**
Total	273	296	316	336	434	**455**

Table 1: The number of solved tasks per heuristic

	h^0				$h^{m\&s}$				h^{LM-cut}			
	expanded		**time(sec)**		**expanded**		**time(sec)**		**expanded**		**time(sec)**	
inst	reg	sym	reg	sym	reg	sym	reg	sym	reg	sym	reg	sym
gripper												
7	10.1M	1.72K	90.0	0.16	10.1M	937	126.5	2.83	10.1M	561	1320	0.14
8	—	2.01K	—	0.25	—	1.35K	—	3.62	—	740	—	0.24
20	—	60.5K	—	66.32	—	16.0K	—	41.07	—	3.14K	—	10.5
mprime												
5	—	—	—	—	1.70M	565K	161.2	147.6	46.2K	22.6K	525	224.1
12	108K	77K	5.92	6.03	35.0K	29.8K	8.78	9.18	122	87	1.57	0.97
19	—	—	—	—	150K	150K	208.3	212.4	—	—	—	—
21	1.5M	438K	186	360	—	—	—	—	900	504	229	97
logistics-9-0-extra-trucks												
2	—	—	—	—	1.07M	325K	43.3	22.3	74.5K	22.9K	130	30.7
3	—	—	—	—	4.12M	657K	188	57.6	183K	30.7K	425	54.2
4	—	—	—	—	—	—	—	—	286K	31.4K	946	71.3

Table 2: Expanded nodes and search time in select instances

four solved instances with the maximal number of expanded nodes). While some cases show improvements in both node expansions and running time (instances with enough symmetries), others exhibit improvements only in node expansion or none at all (few or no symmetries). Still, the loss in running time is never severe. The improvement in the number of node expansions did not lead to the same effect in search time in all heuristics. $h^{m\&s}$ and h^0 spend very little time per state evaluation (and so symmetry pruning is less effective), while h^{LM-cut}, which is currently the best heuristic known, takes more time to calculate per state.

As the IPC problems are often constructed with few symmetries (perhaps in an attempt to focus on other difficulties), we sought to demonstrate that symmetries *can* occur naturally. The third set of instances we exhibit in Table 2 was created by taking a problem from the logistics domain and adding more trucks at the same locations as those of existing trucks.[9] The instance numbers described match the number of trucks that were added to the instance, and our algorithm does exploit emerging symmetries.

6 Conclusions

We presented a set of algorithms that we applied to state-based planners in order to help them deal with highly symmetric problems. Implementing our techniques shows that significant improvements in symmetric instances can be achieved without serious harm to performance elsewhere.

[9]The domain deals with delivery of packages using trucks.

Future work includes improving the approximation of the canonical state, and finding fast variants of shallow pruning that will make it more useful in practice. Also, we have not fully utilized all symmetries in the transition graph, and believe that other symmetry exploitation algorithms can be developed within the general framework we have outlined.

Acknowledgments This work was partially supported by Israel Science Foundation grant #898/05, and Israel Ministry of Science and Technology grant #3-6797.

References

Bäckström, C., and Nebel, B. 1995. Complexity results for SAS+ planning. *Computational Intelligence* 11:625–656.

Cohen, D.; Jeavons, P.; Jefferson, C.; Petrie, K.; and Smith, B. 2006. Symmetry definitions for constraint satisfaction problems. *Constraints* 11:115–137.

Emerson, E. A., and Sistla, A. P. 1996. Symmetry and model checking. *Formal Methods in System Design* 9(1/2):105–131.

Fox, M., and Long, D. 1999. The detection and exploitation of symmetry in planning problems. In *Proceedings of IJCAI'99*, 956–961.

Fox, M., and Long, D. 2002. Extending the exploitation of symmetries in planning. In *Proceedings of AIPS'02*, 83–91.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In *Proceedings of ICAPS'09*, 162–169.

Helmert, M., and Röger, G. 2008. How good is almost perfect? In *Proceedings of AAAI'08*, 944–949. AAAI Press.

Helmert, M.; Haslum, P.; and Hoffmann, J. 2007. Flexible abstraction heuristics for optimal sequential planning. In *Proceedings of ICAPS'07*, 176–183.

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

Junttila, T., and Kaski, P. 2007. Engineering an efficient canonical labeling tool for large and sparse graphs. In *Proceedings of ALENEX'07*, 135–149. SIAM.

Luks, E. M. 1993. Permutation groups and polynomial-time computation. *Groups and Computation, DIMACS series in Discrete Math and Theoretical Comp. Sci.* 11:139–175.

Miguel, I. 2001. Symmetry-breaking in planning: Schematic constraints. In *Proceedings of the CP'01 Workshop on Symmetry in Constraints*, 17–24.

Porteous, J.; Long, D.; and Fox, M. 2004. The identification and exploitation of almost symmetry in planning problems. In Brown, K., ed., *Proceedings of the UK PlanSIG'04*.

Puget, J.-F. 1993. On the satisfiability of symmetrical constrained satisfaction problems. In *Methodologies for Intelligent Systems*, volume 689 of *LNCS*. Springer. 350–361.

Rintanen, J. 2003. Symmetry reduction for SAT representations of transition systems. In *Proceedings of ICAPS'03*, 32–41.

Walsh, T. 2007. Breaking value symmetry. In *Principles and Practice of Constraint Programming (CP'07)*, volume 4741 of *LNCS*. Springer Berlin / Heidelberg. 880–887.

Landmark-Aware Strategies for Hierarchical Planning

Mohamed Elkawkagy and **Pascal Bercher** and **Bernd Schattenberg** and **Susanne Biundo**

Institute of Artificial Intelligence,
Ulm University, D-89069 Ulm, Germany,
email: *forename.surname*@uni-ulm.de

Abstract

In hierarchical planning, landmarks are abstract tasks the decomposition of which are mandatory when trying to find a solution to a given problem. In this paper, we present novel domain-independent strategies that exploit landmark information to speed up the planning process. The empirical evaluation shows that the landmark-aware strategies outperform established search strategies for hierarchical planning.

1 Introduction

While landmarks are widely used to improve the performance of classical planners, a different notion of landmarks has recently been developed for HTN-based approaches (Elkawkagy, Schattenberg, and Biundo 2010). Unlike the classical case where landmarks are facts that must hold in some intermediate state of any solution plan, hierarchical landmarks are mandatory tasks – tasks that have to be decomposed on any search path leading from the initial plan to a solution of the planning problem.

Hierarchical task network (HTN) planning relies on the concepts of tasks and methods (Erol, Hendler, and Nau 1994). While primitive tasks correspond to classical planning operators, abstract tasks are a means to represent complex activities. For each abstract task, a number of methods are available each of which provides a task network, i.e., a plan that specifies a predefined (abstract) solution of the task. Planning problems are (initial) task networks. They are solved by incrementally decomposing the abstract tasks until the network contains only primitive ones in executable order.

Strategies of HTN-based planners differ in the ways they select appropriate methods and interleave the decomposition of tasks with measures to resolve causal interactions between tasks. Systems of the SHOP family, like SHOP2, expand tasks in the order in which they are to be executed and consider causality only on primitive levels (Nau et al. 2003). Other strategies alternate task decomposition and causal conflict resolution (McCluskey 2000) or comply with the current state of the task network (Schattenberg, Bidot, and Biundo 2007).

In this paper, we describe how the exploitation of landmark information leads to novel domain-independent search strategies for HTN-based planning. A so-called landmark table is extracted from the current planning problem in a pre-processing step. It lists the landmark tasks and reveals the various options at hand. Options are tasks that are not mandatory, but may have to be decomposed depending on the method that is selected to implement the respective landmark. This information is used to compute the expansion effort of the problem – a heuristic to guide the selection of methods and with that reduce the effective branching factor of the search space. We implemented the landmark-aware planning strategies in our experimental setting and evaluated their performance on the well-established *Satellite* and *UM-Translog* benchmarks. It turned out that the novel strategies outperform their conventional counterparts on practically all problems, if the decomposition hierarchy of the underlying domain is of non-trivial depth.

The use of landmarks in hierarchical planning is quite novel. In classical state-based planning the concept of landmarks (Porteous, Sebastia, and Hoffmann 2001) enabled the development of strong heuristics (Helmert and Domshlak 2009; Bonet and Helmert 2010). LAMA, the currently best performing classical planner uses such a landmark heuristic (Richter and Westphal 2010). The work of Zhu and Givan (2004) generalized landmarks to so-called action landmarks. As for HTN-based planning, Marthi, Russell, and Wolfe (2008) introduce abstract landmark facts that are gained from effects of basic actions via incremental abstraction.

In the remainder of the paper, we give a brief introduction into the underlying planning framework and the concept of hierarchical landmarks. We then define the landmark-aware strategies and describe the experimental setting as well as the evaluation results.

2 Planning Framework

The planning framework is based on a hybrid formalization (Biundo and Schattenberg 2001) which fuses HTN planning with partial-order causal-link (POCL) planning. For the purpose of this paper, only the HTN shares of the framework are considered, however. A *task schema* $t(\bar{\tau}) = \langle \text{prec}, \text{eff} \rangle$ specifies the preconditions and effects of a task via conjunctions of literals over the task parameters $\bar{\tau} = \tau_1 \ldots \tau_n$. *States* are sets of literals. Applicability of tasks and the state transformations caused by their execution are defined as usual. A *plan* $P = \langle S, \prec, V, C \rangle$ consists of a set S of *plan steps*, i.e., uniquely labeled, (partially) instantiated tasks, a set $\prec$ of *ordering constraints* that impose a partial order on S, a set V of *variable constraints*, and a set C of *causal links*. V consists of (in)equations that associate variables with other variables or constants; it also reflects the (partial) instantiation of the plan steps in P. We denote by $Ground(S, V)$ the set of ground tasks obtained by equating all parameters of all tasks in P with constants, in a way compatible with V. The causal links are adopted from POCL planning: a causal link $l{:}t(\overline{\tau}) \rightarrow_{\varphi} l'{:}t'(\overline{\tau}')$ indicates that φ is implied by the precondition of plan step $l'{:}t'(\overline{\tau}')$ and at the same time is a consequence of the effects of plan step $l{:}t(\overline{\tau})$. Hence, φ is said to be *supported* this way. Methods $m = \langle t(\overline{\tau}), P\rangle$ relate an abstract task $t(\overline{\tau})$ to a plan P, which is called an *implementation* of $t(\overline{\tau})$. Multiple methods may be provided for each abstract task.

An HTN planning problem $\Pi = \langle D, s_{\text{init}}, P_{\text{init}}\rangle$ is composed of a domain model $D = \langle T, M\rangle$, where T and M denote sets of task schemata and decomposition methods, an initial state s_{init}, and an initial plan P_{init}. A plan $P = \langle S, \prec, V, C\rangle$ is a solution to Π if and only if:

1. P is a refinement of P_{init}, i.e., a successor of the initial plan in the induced search space (see Def. 1 below);
2. each precondition of a plan step in S is supported by a causal link in C and no such link is threatened, i.e., for each causal link $l{:}t(\overline{\tau}) \rightarrow_{\varphi} l'{:}t'(\overline{\tau}')$ the ordering constraints in $\prec$ ensure that no plan step $l''{:}t''(\overline{\tau}'')$ with an effect that implies $\neg\varphi$ can be placed between plan steps $l{:}t(\overline{\tau})$ and $l'{:}t'(\overline{\tau}')$;
3. the ordering and variable constraints are consistent, i.e., $\prec$ does not induce cycles on S and the (in)equations in V are not contradictory; and
4. all plan steps in S are primitive ground tasks.

$\mathcal{S}ol_{\Pi}$ denotes the set of all solutions of Π.

Please note that we encode the initial state via the effects of an artificial primitive "start" task, as it is usually done in POCL planning. In doing so, the second criterion guarantees that the solution is executable in the initial state.

In order to refine the initial plan into a solution, there are various *refinement steps* (or *plan modifications*) available; in HTN planning, these are: (1) the decomposition of abstract tasks using methods, (2) the insertion of causal links to support open preconditions of plan steps, (3) the insertion of ordering constraints, and (4) the insertion of variable constraints. Given an HTN planning problem we can define the induced search space as follows.

Definition 1 (Induced Search Space) *The directed graph $\mathcal{P}_{\Pi} = \langle \mathcal{V}_{\Pi}, \mathcal{E}_{\Pi}\rangle$ with vertices $\mathcal{V}_{\Pi}$ and edges $\mathcal{E}_{\Pi}$ is called the* induced search space of planning problem Π *if and only if (1) $P_{\text{init}} \in \mathcal{V}_{\Pi}$, (2) if there is a plan modification refining $P \in \mathcal{V}_{\Pi}$ into a plan P', then $P' \in \mathcal{V}_{\Pi}$ and $(P, P') \in \mathcal{E}_{\Pi}$, and (3) $\mathcal{P}_{\Pi}$ is minimal such that (1) and (2) hold.*

For $\mathcal{P}_{\Pi} = \langle \mathcal{V}_{\Pi}, \mathcal{E}_{\Pi}\rangle$, we write $P \in \mathcal{P}_{\Pi}$ instead of $P \in \mathcal{V}_{\Pi}$.

Note that $\mathcal{P}_{\Pi}$ is in general neither acyclic nor finite. For the former, consider a planning problem in which there are the abstract tasks $t(\overline{\tau})$, $t'(\overline{\tau}')$ as well as two methods, each of which transforms one task into the other. For the latter, consider a planning problem containing an abstract task $t(\overline{\tau})$ and a primitive task $t'(\overline{\tau}')$ as well as two methods for $t(\overline{\tau})$: one maps $t(\overline{\tau})$ to a plan containing only $t'(\overline{\tau}')$, the other maps $t(\overline{\tau})$ to a plan containing $t'(\overline{\tau}')$ and $t(\overline{\tau})$ thus enabling the construction of arbitrary long plans.

In order to search for solutions the induced search space is explored in a heuristically guided manner by the following standard refinement planning algorithm:

Algorithm 1: Refinement Planning Algorithm

```
Input  : The sequence Fringe = $\langle P_{\text{init}}\rangle$.
Output : A solution or fail.
1 while Fringe = $\langle P_1 \dots P_n\rangle \neq \varepsilon$ do
2     $F \leftarrow f^{\text{FlawDet}}(P_1)$
3     if $F = \emptyset$ then return $P_1$
4     $\langle m_1 \dots m_k\rangle \leftarrow f^{\text{ModOrd}}(\bigcup_{f \in F} f^{\text{ModGen}}(f))$
5     succ $\leftarrow \langle app(m_1, P_1) \dots app(m_k, P_1)\rangle$
6     Fringe $\leftarrow f^{\text{PlanOrd}}(succ \circ \langle P_2 \dots P_n\rangle)$
7 return fail
```

The fringe $\langle P_1 \dots P_n\rangle$ is a sequence containing all unexplored plans that are direct successors of visited non-solution plans in $\mathcal{P}_{\Pi}$. It is ordered in a way such that a plan P_i is estimated to lead more quickly to a solution than plans P_j for $j > i$. The current plan is always the first plan of the fringe. The planning algorithm iterates on the fringe as long as no solution is found and there are still plans to refine (line 1). Hence, the flaw detection function f^{FlawDet} in line 2 calculates all flaws of the current plan. A flaw is a set of plan components that are involved in the violation of a solution criterion. The presence of an abstract task raises a flaw that consists of that task, a causal threat consists of a causal link and the threatening plan step, for example. If no flaws can be found, the plan is a solution and returned (line 3). In line 4, the modification generating function f^{ModGen} calculates all plan modifications that address the flaws of the current plan. Afterwards, the modification ordering function f^{ModOrd} orders these modifications according to a given strategy. The fringe is finally updated in two steps: first, the plans resulting from applying the modifications are computed (line 5) and put at the beginning of the fringe (line 6). Second, the plan ordering function f^{PlanOrd} orders the updated fringe. This step can also be used to discard plans, i.e., to delete plans permanently from the fringe. This is useful for plans that contain unresolvable flaws like an inconsistent ordering of tasks. If the fringe becomes empty, no solution exists and fail is returned.

In this setting, the search strategy appears as a combination of the plan modification and plan ordering functions. In order to perform a depth first search, for example, the plan ordering is the identity function ($f^{\text{PlanOrd}}(\overline{P}) = \overline{P}$ for any sequence $\overline{P}$), whereas the modification ordering f^{ModOrd} determines, which branch of the search space to visit first.

3 Landmarks

The landmark-aware planning strategies rely on hierarchical and local landmarks – ground tasks that occur in the plan sequences leading from a problem's initial plan to its solution.

Definition 2 (Solution Sequences) *Let $\langle \mathcal{V}_\Pi, \mathcal{E}_\Pi \rangle$ be the induced search space of planning problem Π. Then, for any plan $P \in \mathcal{V}_\Pi$, $\mathcal{S}ol\mathcal{S}eq_\Pi(P) := \{\langle P_1 \ldots P_n \rangle \mid P_1 = P, (P_i, P_{i+1}) \in \mathcal{E}_\Pi$ for all $1 \leq i < n$, and $P_n \in \mathcal{S}ol_\Pi$ for $n \geq 1\}$.*

Definition 3 (Landmark) *A ground task $t(\overline{\tau})$ is called a* landmark *of planning problem Π, if and only if for each $\langle P_1 \ldots P_n \rangle \in \mathcal{S}ol\mathcal{S}eq_\Pi(P_{\text{init}})$ there is an $1 \leq i \leq n$, such that $t(\overline{\tau}) \in Ground(S_i, V_n)$ for $P_i = \langle S_i, \prec_i, V_i, C_i \rangle$ and $P_n = \langle S_n, \prec_n, V_n, C_n \rangle$.*

While a landmark occurs in every plan sequence that is rooted in the initial plan and leads towards a solution, a *local* landmark occurs merely in each such sequence rooted in a plan containing a specific abstract ground task $t(\overline{\tau})$.

Definition 4 (Local Landmark of an Abstract Task) *For an abstract ground task $t(\overline{\tau})$ let $\mathcal{P}_\Pi(t(\overline{\tau})) := \{P \in \mathcal{P}_\Pi \mid P = \langle S, \prec, V, C \rangle$ and $t(\overline{\tau}) \in Ground(S, V)\}$.*
A ground task $t'(\overline{\tau}')$ is a local landmark *of $t(\overline{\tau})$, if and only if for all $P \in \mathcal{P}_\Pi(t(\overline{\tau}))$ and each $\langle P_1 \ldots P_n \rangle \in \mathcal{S}ol\mathcal{S}eq_\Pi(P)$ there is an $1 \leq i \leq n$, such that $t'(\overline{\tau}') \in Ground(S_i, V_n)$ for $P_i = \langle S_i, \prec_i, V_i, C_i \rangle$ and $P_n = \langle S_n, \prec_n, V_n, C_n \rangle$.*

Since there are only finitely many task schemata and we assume only finitely many constants, there is only a finite number of (local) landmarks.

Given a planning problem Π, the relevant landmark information can be extracted in a pre-processing step. We use the extraction procedure introduced in previous work of the authors (Elkawkagy, Schattenberg, and Biundo 2010) and assume that the landmark information is already stored in a so-called *landmark table*. Its definition relies on a task decomposition graph, which is a relaxed representation of how the initial plan of a planning problem can be decomposed.

Definition 5 (Task Decomposition Graph) *The directed bipartite graph $\langle V_T, V_M, E \rangle$ with task vertices V_T, method vertices V_M, and edges E is called the task decomposition graph (TDG) of planning problem Π if and only if*

1. *$t(\overline{\tau}) \in V_T$ for all $t(\overline{\tau}) \in Ground(S, V)$, for $P_{init} = \langle S, \prec, V, C \rangle$,*
2. *if $t(\overline{\tau}) \in V_T$ and if there is a method $\langle t(\overline{\tau}'), \langle S, \prec, V, C \rangle \rangle \in M$, then*
   (a) $\langle t(\overline{\tau}), \langle S, \prec, V', C \rangle \rangle \in V_M$ such that $V' \supseteq V$ binds all variables in S to a constant and
   (b) $(t(\overline{\tau}), \langle t(\overline{\tau}), \langle S, \prec, V', C \rangle \rangle) \in E$,
3. *if $\langle t(\overline{\tau}), \langle S, \prec, V, C \rangle \rangle \in V_M$, then*
   (a) $t'(\overline{\tau}') \in V_T$ for all $t'(\overline{\tau}') \in Ground(S, V)$ and
   (b) $(\langle t(\overline{\tau}), \langle S, \prec, V, C \rangle \rangle, t'(\overline{\tau}')) \in E$, and
4. *$\langle V_T, V_M, E \rangle$ is minimal such that (1), (2), and (3) hold.*

Note that the TDG of a planning problem is always finite as there are only finitely many ground tasks.

Please also note that, due to the uninformed instantiation of unbound variables in a decomposition step in criterion 2.(a), the TDG of a planning problem becomes in general intractably large. We hence prune parts of the TDG which can provably be ignored due to a relaxed reachability analysis of primitive tasks. This pruning technique is described in our earlier work (Elkawkagy, Schattenberg, and Biundo 2010).

The *landmark table* is a data structure that represents a (possibly pruned) TDG plus additional information about local landmarks.

Definition 6 (Landmark Table) *Let $\langle V_T, V_M, E \rangle$ be a (possibly pruned) TDG of the planning problem Π. The* landmark table *of Π is the set $LT = \{\langle t(\overline{\tau}), M(t(\overline{\tau})), O(t(\overline{\tau})) \rangle \mid t(\overline{\tau}) \in V_T$ abstract ground task$\}$, where $M(t(\overline{\tau}))$ and $O(t(\overline{\tau}))$ are defined as follows:*

$$M(t(\overline{\tau})) := \{t'(\overline{\tau}') \in V_T \mid t'(\overline{\tau}') \in Ground(S, V) \text{ for all } \langle t(\overline{\tau}), \langle S, \prec, V, C \rangle \rangle \in V_M\}$$

$$O(t(\overline{\tau})) := \{Ground(S, V) \setminus M(t(\overline{\tau})) \mid \langle t(\overline{\tau}), \langle S, \prec, V, C \rangle \rangle \in V_M\}$$

Each landmark table entry partitions the tasks introduced by decompositions into two sets: mandatory tasks $M(t(\overline{\tau}))$ are those ground tasks that are contained in all plans introduced by some method which decomposes $t(\overline{\tau})$; hence, they are local landmarks of $t(\overline{\tau})$. The optional task set $O(t(\overline{\tau}))$ contains for each method decomposing $t(\overline{\tau})$ the set of ground tasks which are not in the mandatory set; it is hence a set of sets of tasks.

Please note that the landmark table encodes a possibly pruned TDG and is thus not unique. In fact, various local landmarks might only be detected after pruning. For instance, suppose an abstract task has three available methods, two of which have some tasks in their referenced plans in common. However, the plan referenced by the third method is disjunctive to the other two. Hence, the mandatory sets are empty. If the third method can be proven to be infeasible and is hence pruned from the TDG, the mandatory set will contain those tasks the plans referenced by the first two methods have in common.

Example

The following example will demonstrate how the TDG and a landmark table of a planning problem looks like.

Thus, let $\Pi = \langle D, s_{\text{init}}, P_{\text{init}} \rangle$ an HTN planning problem with $P_{\text{init}} = \langle \{l_1{:}t_1(\tau_1)\}, \{\tau_1 = c_1\} \rangle$[1], $D = \langle T, M \rangle$, $T = \{t_1(\tau_1), \ldots, t_5(\tau_5)\}$, and $M = \{m_a, m'_a, m_b, m'_b\}$ with:

$$m_a := \langle t_1(\tau_1), \langle \{l_1{:}t_3(\tau_1), l_2{:}t_3(\tau_2), l_3{:}t_2(\tau_1)\}, \{\tau_1 \neq \tau_2\} \rangle \rangle$$
$$m'_a := \langle t_1(\tau_1), \langle \{l_1{:}t_2(\tau_1), l_2{:}t_1(\tau_1)\}, \emptyset \rangle \rangle$$
$$m_b := \langle t_3(\tau_1), \langle \{l_1{:}t_4(\tau_1), l_2{:}t_5(\tau_1)\}, \emptyset \rangle \rangle$$
$$m'_b := \langle t_3(\tau_1), \langle \{l_1{:}t_4(\tau_1)\}, \emptyset \rangle \rangle$$

The TDG for Π is given in Figure 1; the according landmark table is depicted in Table 1.

[1] As our example comes without ordering constraints and causal links, we give plans as 2-tuples $P = \langle S, V \rangle$

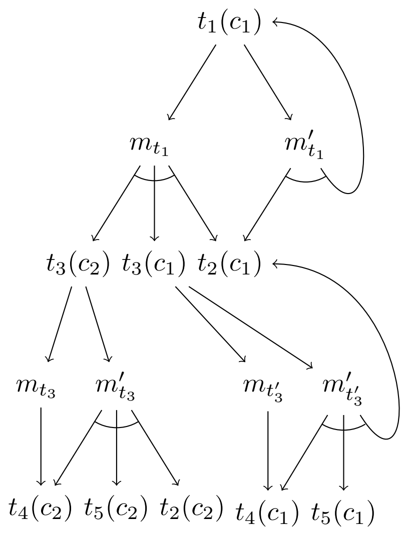


Figure 1: The TDG for the planning problem Π. The method vertices are given as follows:
$m_{t_1} = \langle t_1(c_1), m_a|_{\tau_1=c_1,\tau_2=c_2}\rangle$, $m'_{t_1} = \langle t_1(c_1), m'_a|_{\tau_1=c_1}\rangle$,
$m_{t_3} = \langle t_3(c_2), m_b|_{\tau_1=c_2}\rangle$, $m'_{t_3} = \langle t_3(c_2), m'_b|_{\tau_1=c_2}\rangle$,
$m_{t'_3} = \langle t_3(c_1), m_b|_{\tau_1=c_1}\rangle$, $m'_{t'_3} = \langle t_3(c_1), m'_b|_{\tau_1=c_2}\rangle$

Table 1: The landmark table for the TDG of Figure 1.

Abs. Task	**Mandatory**	**Optional**
$t_1(c_1)$	$\{t_2(c_1)\}$	$\{\{t_3(c_2), t_3(c_1)\}, \{t_1(c_1)\}\}$
$t_3(c_2)$	$\{t_4(c_2)\}$	$\{\emptyset, \{t_5(c_2), t_2(c_2)\}\}$
$t_3(c_1)$	$\{t_4(c_1)\}$	$\{\emptyset, \{t_5(c_1), t_2(c_1)\}\}$

4 Landmark-Aware Strategies

Exploiting landmarks during planning is based on the idea that identifying such landmarks along the refinement paths perfectly guides the search process because they are "inevitable" elements on the way to any solution. The mandatory sets in the landmark table do not contribute directly to the identification of a solution path. They do, however, allow to estimate upper and lower bounds for the number of expansions an abstract task requires before a solution is found. A landmark table entry $\langle t(\overline{\tau}), M(t(\overline{\tau})), O(t(\overline{\tau}))\rangle$ carries the following information: if the planning system decomposes the task $t(\overline{\tau})$, all tasks in the mandatory set $M(t(\overline{\tau}))$ are introduced into the refinement plan, no matter which method is used. With the optional tasks at hand we can now infer that in the most optimistic case a solution can be developed straight from the implementation of the method with the "smallest" remains according to $O(t(\overline{\tau}))$. Following a similar argument, the upper bound for the "expansion effort" can be obtained by adding the efforts for all implementations that are stored in the optional set.

From the above considerations, two essential properties of our landmark-aware strategies emerge: first, since the landmark exploitation will be defined in terms of measuring expansion alternatives, the resulting strategy component has to be a modification ordering function. Second, if we base the modification preference on the optional sets in the landmark table entries, we implement an abstract view on the method definition that realizes the least-commitment principle.

Concerning the first two strategies below, we interpret the term "expansion effort" literally and therefore define "smallest" method to be the one with the fewest abstract tasks in the implementing plan. To this end, we define the cardinality of a set of tasks in terms of the number of corresponding entries that a given landmark table does contain.

Definition 7 (Landmark Cardinality) *Given a landmark table LT, we define the* landmark cardinality *of a set of tasks* $o = \{t_1(\overline{\tau}_1), \dots, t_n(\overline{\tau}_n)\}$ *to be*

$$|o|_{LT} := |\{t(\overline{\tau}) \in o \mid \langle t(\overline{\tau}), M(t(\overline{\tau})), O(t(\overline{\tau}))\rangle \in LT\}|$$

A heuristic based on this information is obviously a rough over-estimation of the search effort because the landmark table typically contains a number of tasks that turn out to be unachievable in the given problem. The strategy also does not take into account the refinement effort it takes to make an implementation operational on the primitive level by establishing causal links, resolving causal threats, and grounding tasks. For the time being, we assume that all methods deviate from a perfect heuristic estimate more or less to the same amount. We will see that this simplification actually yields a heuristic with good performance.

Definition 8 (Landmark-aware strategy lm_1**)** *Given a plan* $P = \langle S, \prec, V, C\rangle$*, let* $t_i(\overline{\tau}_i)$ *and* $t_j(\overline{\tau}_j)$ *be ground instances of two abstract tasks in S that are compatible with the (in)equations in V and that are referenced by two abstract task flaws* $\mathtt{f}_i$ *and* $\mathtt{f}_j$*, respectively, that are found in P. Let a given landmark table LT contain the corresponding entries* $\langle t_i(\overline{\tau}_i), M(t_i(\overline{\tau}_i)), O(t_i(\overline{\tau}_i))\rangle$ *and* $\langle t_j(\overline{\tau}_j), M(t_j(\overline{\tau}_j)), O(t_j(\overline{\tau}_j))\rangle$.

The modification ordering function lm_1 *orders a plan modification* $\mathtt{m}_i$ *before* $\mathtt{m}_j$ *if and only if* $\mathtt{m}_i$ *addresses* $\mathtt{f}_i$*,* $\mathtt{m}_j$ *addresses* $\mathtt{f}_j$*, and*

$$\sum_{o \in O(t_i(\overline{\tau}_i))} |o|_{LT} < \sum_{o \in O(t_j(\overline{\tau}_j))} |o|_{LT}$$

This strategy implements the least commitment principle, as it favors those decomposition plan refinements that impose less successor plans. It reduces the effective branching factor of the search space (cf. *fewest alternatives first* heuristic in HTN planning (Tsuneto, Nau, and Hendler 1997)). The proper choice of the ground task instances $t_i(\overline{\tau}_i)$ and $t_j(\overline{\tau}_j)$ in the above definition is crucial for the actual performance, however, because the plan modifications typically operate on the lifted abstract tasks and method definitions.

While the above heuristic focuses on the very next level of refinement, a strategy should also take estimates for subsequent refinement levels into account, thus minimizing the number of refinement choices until no more decompositions are necessary. To this end, for a given landmark table LT, let $O^*(t(\overline{\tau}))$ be the transitive closure of the optional sets on a recursive traversal of the table entries, beginning in $t(\overline{\tau})$.

Definition 9 (Closure of the Optional Set) *The* closure of the optional set *for a given ground task $t(\overline{\tau})$ and a landmark table LT is the smallest set $O^*(t(\overline{\tau}))$, such that $O^*(t(\overline{\tau})) = \emptyset$ for primitive $t(\overline{\tau})$, and otherwise:*

$$O^*(t(\overline{\tau})) = O(t(\overline{\tau})) \cup \bigcup_{o \in O(t(\overline{\tau}))} \Big(\bigcup_{t'(\overline{\tau}') \in o} O^*(t'(\overline{\tau}')) \Big)$$

$$\textit{with } \langle t(\overline{\tau}), M(t(\overline{\tau})), O(t(\overline{\tau})) \rangle \in LT$$

Note that $O^*(t(\overline{\tau}))$ is always finite due to the finiteness of the landmark table, even for cyclic method definitions.

Considering the the previous example (cf. Figure 1 and Table 1), the closures for the three abstract tasks of the planning problem Π are as follows: $O^*(t_1(c_1)) = O(t_1(c_1)) \cup O(t_3(c_2)) \cup O(t_3(c_1))$, $O^*(t_3(c_2)) = O(t_3(c_2))$, and $O^*(t_3(c_1)) = O(t_3(c_1))$.

Definition 10 (Landmark-aware strategy lm_1^*) *Given the prerequisites from Def. 8, the modification ordering function lm_1^* orders a plan modification $\mathtt{m}_i$ before $\mathtt{m}_j$ if and only if $\mathtt{m}_i$ addresses $\mathtt{f}_i$, $\mathtt{m}_j$ addresses $\mathtt{f}_j$, and*

$$\sum_{o \in O^*(t_i(\overline{\tau}_i))} |o|_{LT} < \sum_{o \in O^*(t_j(\overline{\tau}_j))} |o|_{LT}$$

So far, the "expansion effort" has been measured in terms of decompositions that have to be applied until a solution is obtained. The following strategies take into account that also primitive tasks in a decomposition contribute to the costs for developing the current plan into a solution. The cost measure is thereby a uniform one: solving the flaws affecting a primitive task is regarded as expensive as the expansion of an abstract task.

Definition 11 (Landmark-aware strategy lm_2) *Given the prerequisites from Def. 8, the modification ordering function lm_2 orders a plan modification $\mathtt{m}_i$ before $\mathtt{m}_j$ if and only if $\mathtt{m}_i$ addresses $\mathtt{f}_i$, $\mathtt{m}_j$ addresses $\mathtt{f}_j$, and*

$$\sum_{o \in O(t_i(\overline{\tau}_i))} |o| < \sum_{o \in O(t_j(\overline{\tau}_j))} |o|$$

Like we did for the landmark-aware strategy lm_1, we define a variant for strategy lm_2 that examines the transitive closure of the optional sets.

Definition 12 (Landmark-aware strategy lm_2^*) *Given the prerequisites from Def. 8, the modification ordering function lm_2^* orders a plan modification $\mathtt{m}_i$ before $\mathtt{m}_j$ if and only if $\mathtt{m}_i$ addresses $\mathtt{f}_i$, $\mathtt{m}_j$ addresses $\mathtt{f}_j$, and*

$$\sum_{o \in O^*(t_i(\overline{\tau}_i))} |o| < \sum_{o \in O^*(t_j(\overline{\tau}_j))} |o|$$

Since the landmark information can be extracted from any domain model and problem in an automated pre-processing step, the above strategies are conceptually domain- and problem-independent heuristics. In addition, they are independent from the actual plan generation procedure, hence their principles can be incorporated into any refinement-based hierarchical planning system.

5 Evaluation

We evaluated the performance of the landmark-aware strategies in a series of experiments in comparison to conventional hierarchical search strategies.

We base our evaluation on the same benchmark problems as in our previous work (Elkawkagy, Schattenberg, and Biundo 2010) including the domain reduction technique. In the new experiments, we compare our novel landmark-aware strategies with conventional ones and thereby show that even on the reduced domain models the landmark information can be used to improve the search efficiency.

Conventional Hierarchical Search Strategies

For the strategies *SHOP* and *UMCP*, we used plan and modification ordering functions that induce the search strategies of these planning systems: in the UMCP system (Erol, Hendler, and Nau 1994), plans are primarily developed into completely primitive plans in which causal interactions are dealt with afterwards. The SHOP strategy (Nau et al. 2003) prefers task expansion for the abstract tasks in the order in which they are to be executed.

In all other strategies the plan ordering function *Fewer Modifications First (fmf)* was used. It prefers plans for which a smaller number of refinement options is found, thereby implementing the least commitment principle on the plan ordering level. For the comparison to our landmark-aware modification ordering functions, we also conducted experiments with the following modification ordering functions:

The *Expand-Then-Make-Sound (ems)* procedure (McCluskey 2000) alternates task expansion with other modifications, which results in a "level-wise" concretion of all plan steps. We also included the well-established *Least Committing First (lcf)* paradigm, a generalization of POCL strategies, which prefers those modifications that address flaws for which the smallest number of alternative solutions is available. From more recent work (Schattenberg, Bidot, and Biundo 2007), two HotSpot-based strategies were deployed. HotSpots denote plan components that are affected by multiple flaws, thereby quantifying to which extent solving one deficiency may interfere with the solution options for coupled components. The *Direct Uniform HotSpot (du-HotSpot)* strategy strictly avoids to address flaws that refer to HotSpot plan components. While the du-HotSpot heuristic treats all flaws uniformly when calculating their interference potential, the *Direct Adaptive HotSpot (da)* strategy puts problem-specific weights on binary combinations of flaw types that occur in the plan. It adapts to a repeated occurrence of flaw type combinations by increasing their weights: if abstract task flaws happen to coincide with causal threats, their combined occurrence becomes more important for the current plan generation episode. As a generalization of singular HotSpots to commonly affected areas of plan components, the *HotZone* modification ordering function takes connections between HotSpots into account and tries to evade modifications that deal with these clusters.

Experimental Results

We conducted our experiments on two well-established planning domains (cf. Table 2). *Satellite* is a benchmark for non-hierarchical planning. The hierarchical encoding of this domain regards the original primitive operators as implementations of abstract observation tasks. The domain model consists of 3 abstract and 5 primitive tasks, and includes 8 methods. *UM-Translog* is a hierarchical planning domain that supports transportation and logistics. It shows 21 abstract and 48 primitive tasks as well as 51 methods.

Please note that we performed our experiments on the reduced domain models. For the satellite domain, the domain model reduction did not have any effect on the number of tasks and/or methods; for the UM-Translog domain, the size of the reduced domain models depends on the given problem instance and is on average 63% as large as the unreduced domain model (Elkawkagy, Schattenberg, and Biundo 2010).

The strategies lm_1, lm_1^*, lm_2, and lm_2^* do outperform the other strategies on practically all problems in the UM-Translog domain (cf. Table 2a) in terms of both size of the explored search space and computation time. This is quite surprising because the landmark table does not reveal any information about causal dependencies on the primitive task level and the strategies hence cannot provide a focused guidance. An adequate selection of the decomposition refinements obviously pays off well enough to compensate for random choice on the causality issues. Another interesting facet is that the strategies $\mathrm{lm}_1^*/\mathrm{lm}_2^*$ being the better informed heuristic while repeatedly performing worse than $\mathrm{lm}_1/\mathrm{lm}_2$. Furthermore, the same anomaly occurs when comparing $\mathrm{lm}_2/\mathrm{lm}_2^*$ with the more abstract but also more successful $\mathrm{lm}_1/\mathrm{lm}_1^*$. We suppose these phenomena result from two sources: First, the random choice of ground candidates for the lifted task instances is relatively unreliable and this effect gets amplified by traversing along the landmark closures and into the primitive task level. Second, the most important choice points are on the early decomposition levels, i.e., once a method has been chosen for implementing the transport, this refinement puts more constraints on the remaining decisions than the strategy can infer from the feasibility analysis underlying the landmark table.

On the Satellite domain our landmark-aware strategies do not clearly dominate any other strategy (cf. Table 2b). This meets our expectations as there is hardly any landmark information available due to the shallow decomposition hierarchy of this domain and any landmark-centered strategy is bound to loose its strength given limited landmark information. However, none of the other strategies in this domain dominated any landmark-aware strategy; thus, all evaluated strategies can be regarded as equally good.

6 Conclusion

In this paper, we introduced four novel landmark-aware search strategies to improve hierarchical planning. In a number of experiments these strategies competed with a set of representative search procedures from the literature. The results showed that the new strategies outperformed the established ones on all relevant problems, i.e., problems with a deep task hierarchy. Further work will be devoted to the construction and evaluation of other types of landmark-aware strategies and to the investigation of those domain model and problem features that suggest their deployment.

ACKNOWLEDGEMENTS

This work is done within the Transregional Collaborative Research Centre SFB/TRR 62 “Companion-Technology for Cognitive Technical Systems” funded by the German Research Foundation (DFG).

References

Biundo, S., and Schattenberg, B. 2001. From abstract crisis to concrete relief – a preliminary report on combining state abstraction and HTN planning. In *Proc. of the 6th European Conference on Planning (ECP 2001)*, 157–168. Springer.

Bonet, B., and Helmert, M. 2010. Strengthening landmark heuristics via hitting sets. In *Proc. of ECAI 2010*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, 329–334. IOS Press.

Elkawkagy, M.; Schattenberg, B.; and Biundo, S. 2010. Landmarks in hierarchical planning. In *Proc. of ECAI 2010*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, 229–234. IOS Press.

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

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What’s the difference anyway? In *Proc. of ICAPS 2009*, 162–169.

Marthi, B.; Russell, S. J.; and Wolfe, J. 2008. Angelic hierarchical planning: Optimal and online algorithms. In *Proc. of ICAPS 2008*, 222–231.

McCluskey, T. L. 2000. Object transition sequences: A new form of abstraction for HTN planners. In *Proc. of the 5th Int. Conf. on Artificial Intelligence Planning Systems (AIPS 2000)*, 216–225. AAAI Press.

Nau, D. S.; Au, T.; Ilghami, O.; Kuter, U.; Murdock, W.; Wu, D.; and Yaman, F. 2003. SHOP2: An HTN planning system. *JAIR* 20:379–404.

Porteous, J.; Sebastia, L.; and Hoffmann, J. 2001. On the extraction, ordering, and usage of landmarks in planning. In *Proc. of the 6th European Conf. on Planning (ECP 2001)*, 37–48.

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

Schattenberg, B.; Bidot, J.; and Biundo, S. 2007. On the construction and evaluation of flexible plan-refinement strategies. In *Proc. of the 30th German Conf. on Artificial Intelligence (KI 2007)*, LNAI 4667, 367–381. Springer.

Tsuneto, R.; Nau, D. S.; and Hendler, J. A. 1997. Plan-refinement strategies and search-space size. In *Proc. of the 4th European Conf. on Planning (ECP 1997)*, volume 1348 of *LNCS*, 414–426. Springer.

Zhu, L., and Givan, R. 2004. Heuristic planning via roadmap deduction. In *IPC-4*, 64–66.

Table 2: This table shows the impact of the deployed modification ordering functions on the planning process. While SHOP and UMCP denote strategy function combinations that simulate the respective search procedures, all other strategy implementations use fmf as the plan ordering function. The tests were run on a machine with a 3 GHz CPU and 256 MB Heap memory for the Java VM. *Space* refers to the number of created plans and *Time* refers to the used time in seconds including pre-processing, which takes only a few seconds even for large problem specifications. Values are the arithmetic means over three runs. Dashes indicate that no solution was found within a limit of 5,000 created nodes and a run-time of 150 minutes. The best result for a given problem is emphasized bold, the second best bold and italic.

(a) Results for the UM-Translog domain. Problems with different versions differ in the number and kind of available locations and/or the number of parcels to transport.

Mod. ordering	Hopper Truck		Auto Truck		Reg. Truck (a)		Reg. Truck (b)		Reg. Truck (c)		Reg. Truck (d)		Flatbed Truck	
function f^{ModOrd}	**Space**	**Time**	**Space**	**Time**	**Space**	**Time**	**Space**	**Time**	**Space**	**Time**	**Space**	**Time**	**Space**	**Time**
da-HotSpot	144	352	644	2077	239	562	114	257	148	352	723	2560	99	237
du-HotSpot	101	224	459	1304	1508	4097	160	460	117	258	–	–	1047	2601
HotZone	55	121	197	527	191	473	55	117	55	137	–	–	159	399
lm$_1$	***52***	***111***	**133**	**329**	145	**374**	62	135	***53***	**122**	***291***	1172	63	155
lm$_1^*$	**51**	**109**	***135***	***462***	154	430	**52**	**112**	65	142	**266**	***1162***	**61**	**144**
lm$_2$	62	162	***135***	464	***141***	469	***53***	123	55	151	339	**1128**	109	315
lm$_2^*$	124	340	146	489	**137**	413	57	148	**51**	**122**	305	1318	110	308
lcf	55	118	155	470	162	463	78	173	127	222	327	1278	***62***	179
ems	147	295	405	976	211	507	127	262	114	235	–	–	1571	3797
SHOP	89	212	164	433	146	***406***	106	241	83	190	926	4005	98	257
UMCP	58	122	156	474	177	506	55	***113***	57	***127***	308	1263	63	***149***

Mod. ordering	Armored R-Truck		Auto Traincar (a)		Auto Traincar (b)		Mail Traincar		Refr. Reg. Traincar		Airplane	
function f^{ModOrd}	**Space**	**Time**	**Space**	**Time**	**Space**	**Time**	**Space**	**Time**	**Space**	**Time**	**Space**	**Time**
da-HotSpot	120	359	–	–	184	705	641	2031	588	1958	172	620
du–HotSpot	75	201	–	–	1390	4018	424	1090	307	775	643	2134
HotZone	122	355	–	–	701	1616	81	224	***76***	***196***	345	1323
lm$_1$	***71***	***177***	**158**	**596**	183	608	**75**	**184**	**72**	**180**	142	441
lm$_1^*$	**61**	**155**	304	1473	***158***	***543***	***78***	***205***	89	212	189	676
lm$_2$	73	199	420	1519	211	888	84	248	91	256	***104***	***320***
lm$_2^*$	81	228	367	1446	**142**	**511**	87	238	84	226	114	436
lcf	86	198	–	–	227	926	79	209	90	225	247	798
ems	113	269	–	–	2558	6447	879	1806	500	1048	784	2517
SHOP	95	227	–	–	247	963	121	274	173	353	150	450
UMCP	75	172	***220***	***739***	161	546	92	229	90	244	**70**	**215**

(b) Results for the Satellite domain. The description "x — y — z" stands for a Satellite problem with x observations, y satellites, and z modi.

Mod. ordering	1 — 1 — 1		1 — 2 — 1		2 — 1 — 1		2 — 1 — 2		2 — 2 — 1		2 — 2 — 2	
function f^{ModOrd}	**Space**	**Time**	**Space**	**Time**	**Space**	**Time**	**Space**	**Time**	**Space**	**Time**	**Space**	**Time**
da-HotSpot	**56**	**60**	68	78	782	1131	832	1301	2186	6841	**142**	***175***
du-HotSpot	100	107	139	150	–	–	–	–	–	–	–	–
HotZone	***61***	**60**	57	62	1281	4764	–	–	1094	1338	871	1114
lm$_1$	73	80	194	208	***560***	***652***	***352***	***400***	693	***785***	295	362
lm$_1^*$	78	85	**34**	**37**	847	969	1803	2569	739	813	619	1228
lm$_2$	78	86	128	140	4890	5804	**200**	**251**	–	–	483	965
lm$_2^*$	73	80	91	99	–	–	1905	2553	–	–	***146***	**161**
lcf	86	93	71	77	1120	1338	3022	4069	***407***	**701**	–	–
ems	65	64	47	53	1586	2608	–	–	1219	1579	–	–
SHOP	62	66	105	111	**138**	**155**	–	–	1406	1780	–	–
UMCP	83	91	***36***	***41***	883	1035	1558	1894	**278**	1097	1062	1270