28th International Conference on
Automated Planning and Scheduling

June 24–29, 2018, Delft, the Netherlands

HSDIP 2018

Proceedings of the 10th Workshop on

Heuristics and Search for Domain-independent Planning (HSDIP)

Edited by:

Guillem Francès, Daniel Gnad, Michael Katz, Nir Lipovetzky,
Christian Muise, Miquel Ramirez, and Silvan Sievers

Organization

Guillem Francès
University of Basel, Switzerland

Daniel Gnad
Saarland University, Germany

Michael Katz
IBM Research AI, NY, USA

Nir Lipovetzky
University of Melbourne, Australia

Christian Muise
IBM Research AI, Cambridge, MA, USA

Miquel Ramirez
University of Melbourne, Australia

Silvan Sievers
University of Basel, Switzerland

Foreword

Planning as heuristic search remains among the dominating approaches to many variations of domain-independent planning, including classical planning, temporal planning, planning under uncertainty and adversarial planning, for nearly two decades. The research on both heuristics and search techniqes is thriving, now more than ever, as evidenced by both the quality and the quantity of submissions on the topic to major AI conferences and workshops.

This workshop seeks to understand the underlying principles of current heuristics and search methods, their limitations, ways for overcoming those limitations, as well as the synergy between heuristics and search. To this end, this workshop intends to offer a discussion forum and a unique opportunity to showcase new and emerging ideas to leading researchers in the area. Past workshops have featured novel methods that have grown and formed indispensable lines of research.

This year marks an important landmark, being the tenth workshop since the first workshop on Heuristics for Domain-independent Planning (HDIP), which was held in 2007. HDIP was subsequently held in 2009 and 2011. With the fourth workshop in 2012, the organizers sought to recognize the role of search algorithms by acknowledging search in the name of the workshop, renaming it to the workshop on Heuristics and Search for Domain-independent Planning (HSDIP). The workshop continued flourishing under the new name and has become an annual event at ICAPS.

Guillem Francès, Daniel Gnad, Michael Katz, Nir Lipovetzky, Christian Muise, Miquel Ramirez, and Silvan Sievers
June 2018

Contents

Relaxed Modification Heuristics for Equi-Reward Utility Maximizing Design

Sarah Keren[†], **Luis Pineda**[‡], **Avigdor Gal**[†], **Erez Karpas**[†], **Shlomo Zilberstein**[‡]
[†]Technion–Israel Institute of Technology
[‡]College of Information and Computer Sciences, University of Massachusetts Amherst
sarahn@campus.technion.ac.il, lpineda@cs.umass.edu*

Abstract

Grounded in a stochastic setting, the objective of equi-reward utility maximizing design (ER-UMD) is to find a valid modification sequence, from a given set of possible environment modifications, which yields maximal agent utility. To efficiently traverse the typically large space of possible modification options, we use heuristic search and propose new heuristics, which relax the design process so instead of computing the value achieved by a single modification, we use a *dominating* modification guaranteed to be at least as beneficial. The proposed technique enables heuristic caching for similar nodes thereby saving computational overhead. We specify sufficient conditions under which this approach is guaranteed to produce admissible estimates and describe a range of models that comply with these requirements. In addition, we provide simple methods to automatically generate dominating modifications. We evaluate our approach on a range of settings for which our heuristic is admissible and compare its efficiency with that of a previously suggested heuristic that employs a relaxation of the environment and a compilation from ER-UMD to planning.

Introduction

Equi-reward utility maximizing design (ER-UMD) (Keren *et al.* 2017) involves redesigning stochastic environments to maximize agent performance. The input of a ER-UMD problem consists of a description of a stochastic environment, a utility measure of the agents acting within it and the possible ways to modify the environment. The objective is to find a modification sequence that maximizes agent utility. The design process is viewed as a search in the often exponential space of possible modification sequences, which motivates the use of heuristic estimations to guide the search.

In this work we present the *simplified-design* heuristic, which relaxes the modification process by mapping each modification that is expanded during the search to a modification that *dominates* it, *i.e.*, a modification guaranteed to yield a value at least as high and use its value as estimation of the value of the original modification.

To generate dominating modifications we propose two approaches, namely *modification relaxation* and *padding*. Modification relaxation consists of applying a hypothetical modification whose effect is potentially easier to compute than the original modification. Padding appends to the examined modification additional modifications. The computed values of padded sequences are cached. When a modification is mapped to a previously encountered relaxed modification, the cached value is reused. Of course, both approaches can be combined with the potential benefit lying in the ability to avoid redundant computations of irrelevant sets of modifications, those that do not affect the agent's expected utility.

For models with lifted modification representations we provide a simple way to automatically generate dominating modifications. We then specify sufficient conditions under which this approach is guaranteed to produce admissible heuristics, *i.e.*, heuristics that over-estimate the value of the original modification. In addition, we formulate and implement a family of models that comply with these requirements and compare the efficiency of our proposed approach with that of a previously suggested heuristic that employs an environment relaxation and with a compilation from ER-UMD to planning.

Example 1 *To illustrate our* simplified-design *heuristic consider Figure 1(left) where an adaptation of the Vacuum cleaning robot domain suggested by Keren* et al. *(2017) is portrayed. The setting includes a robot (depicted by a black circle) that needs to collect, as quickly as possible, pieces of dirt (depicted by stars) scattered in the room. The robot needs to navigate around the furniture in the room, depicted by shaded cells. Accounting for uncertainty, the robot may slip when moving, ending up in a different location than intended. To facilitate the robot's task, the environment can be modified by removing furniture or by placing high friction tiles to reduce the probability of slipping. The number of allowed modifications is constrained by a design budget.*

The simplified-design *heuristic is implemented by partitioning the environment into* zones *(Figure 1(center)). To heuristically evaluate the impact of removing the piece of furniture indicated by the arrow in Figure 1(right), we remove all furniture from the entire zone and use this value as an (over) estimation of the single modification. When considering the removal of another piece of furniture in the same zone, the already computed value is reused.*

*Last three authors email addresses: avigal@ie.technion.ac.il, karpase@technion.ac.il, shlomo@cs.umass.edu

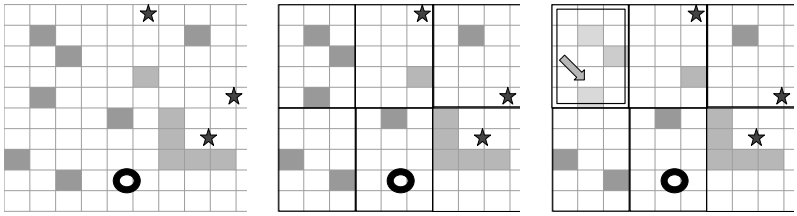


Figure 1: An example ER-UMD problem

The main contributions of this work are threefold. First, we propose a new class of heuristics for ER-UMD, called *simplified-design*. Second, we identify conditions under which this class of heuristics is admissible. Finally, we describe a concrete procedure to automatically generate such heuristics. Our empirical evaluation demonstrates the benefit of the proposed heuristic.

In the remaining of the paper we first overview the ER-UMD framework and then describe our novel techniques for solving the ER-UMD problem. Our empirical evaluation is followed by a description of related work and concluding remarks.

Background: Equi-Reward Utility Maximizing Design as Heuristic Search

The *equi-reward utility maximizing design* (ER-UMD) problem, recently suggested by Keren *et al.* (2017), takes as input an environment with stochastic action outcomes, a utility measure of the agents that act in it, a set of allowed modifications, and a set of constraints. The aim is to find an optimal sequence of modifications to apply to the environment for maximizing the agent utility[1] under the given constraints.

The ER-UMD framework considers stochastic environments defined by the quadruple $\epsilon = \langle S, A, f, s_0 \rangle$ with a set of states S, a set of actions A, a stochastic transition function $f : S \times A \times S \to [0, 1]$ specifying the probability $f(s, a, s')$ of reaching state s' after applying action a in $s \in S$, and an initial state $s_0 \in S$. We let $\mathcal{E}$, $\mathcal{S}_\mathcal{E}$ and $\mathcal{A}_\mathcal{E}$ denote the set of all environments, states and actions, respectively.

An ER-UMD model is a tuple $\omega = \langle \epsilon^0, \mathcal{R}, \gamma, \mathbf{\Delta}, \mathcal{F}, \Phi \rangle$ where, $\epsilon^0 \in \mathcal{E}$ is an initial environment, $\mathcal{R} : \mathcal{S}_\mathcal{E} \times \mathcal{A}_\mathcal{E} \times \mathcal{S}_\mathcal{E} \to \mathbb{R}$ is a Markovian and stationary reward function specifying the reward $r(s, a, s')$ an agent gains from transitioning from state s to s' by the execution of a, and γ is a discount factor in $(0, 1]$, representing the deprecation of agent rewards over time. The set $\mathbf{\Delta}$ contains the atomic modifications a system can apply. A modification sequence is an ordered set of modifications $\vec{\delta} = \langle \delta_1, \ldots, \delta_n \rangle$ s.t. $\delta_i \in \mathbf{\Delta}$ and $\vec{\mathbf{\Delta}}$ is the set of all such sequences. $\mathcal{F} : \mathbf{\Delta} \times \mathcal{E} \to \mathcal{E}$ is a deterministic modification transition function, specifying the result of applying a modification to an environment. Finally, $\Phi : \vec{\mathbf{\Delta}} \times \mathcal{E} \to \{0, 1\}$ specifies allowed modification sequences in an environment.

[1]Whenever agent utility is expressed as cost, the objective is to minimize expected cost.

The reward function $\mathcal{R}$, discount factor γ and environment $\epsilon \in \mathcal{E}$ represent an infinite horizon discounted reward Markov decision process (MDP) (Bertsekas 1995) $\langle S, A, f, s_0, \mathcal{R}, \gamma \rangle$. We assume agents are optimal and let $\mathcal{V}^*(\omega)$ represent the discounted expected agent reward of following an optimal policy from the initial state s_0 in an initial environment ϵ^0. The objective is to find a legal modification sequence $\vec{\delta} \in \vec{\mathbf{\Delta}}^*$ to apply to ϵ^0 that maximizes $\mathcal{V}^*(\omega^{\vec{\delta}})$ under the constraints, where $\omega^{\vec{\delta}}$ is the ER-UMD that results from applying $\vec{\delta}$ to ϵ^0.

Keren *et al.* (2017) propose to view the design process as a search in the space of modification sequences and suggest two methods for solving the ER-UMD problem. The first, referred to as *DesignComp*, embeds the offline design stage into the definition of the agent's planning problem (i.e. MDP description) which can be solved by any off-the-shelf MDP solver. The second approach, namely the Best First Design (BFD) algorithm, applies a heuristic search in the space of modifications. To evaluate the value of a modification sequence efficiently, the *simplified-environment* heuristic was proposed, relaxing the environment using relaxation approaches from the literature (*e.g.*, delete relaxation that ignores the negative outcomes of an action (Bonet *et al.* 1997)), before evaluating a modification on the relaxed environment.

The *simplified-design* Heuristic

To estimate the value of a modification we *relax* the design process by mapping the modification to a modification that *dominates* it, meaning it achieves a utility at least as high as the original modification's utility. This approach can be exploited in two ways. First, if the value of the dominating modification is easier to compute, it can be used to estimate the value of the original modification. In addition, we can cache the computed values and reuse them for each encountered node (and corresponding modification) that is dominated by the same relaxed modification.

After formally defining the *simplified-design* heuristic, we characterize ER-UMD settings where relaxing modifications is easy to implement and in which our approach is guaranteed to yield admissible heuristic, *i.e.*, overestimations of the expected value of the applied modifications.

We let ϵ^δ represent the environment that results from applying δ to ϵ, and let $\vec{\mathbf{\Delta}}$ and $\mathcal{E}_\omega$ represent all modification sequences and environments in a ER-UMD model, respectively.

Definition 1 (dominating modification) *Given an ER-UMD model* $\omega = \langle \epsilon^0, \mathcal{R}, \gamma, \mathbf{\Delta}, \mathcal{F}, \Phi \rangle$*, a modification sequence* δ' dominates *modification* δ *in* ω *if for every* $\epsilon \in \mathcal{E}_\omega$*,*

$$\mathcal{V}^*(\omega^\delta) \leq \mathcal{V}^*(\omega^{\delta'})$$

where ω^δ *and* $\omega^{\delta'}$ *are the ER-UMD models that have* ϵ^δ *and* $\epsilon^{\delta'}$ *as their initial environments, respectively.*

The *simplified-design* heuristic, denoted by h^{simdes} estimates the value of applying a modification δ to ω by the value of the dominating modification δ'.

$$h^{simdes}(\omega^\delta) \stackrel{def}{=} \mathcal{V}^{max}(\omega^{\delta'}) \quad (1)$$

Lemma 1 h^{simdes} *is admissible in any ER-UMD model* ω*.*

Proof: Immediate from the definition of dominance. ∎

Admissibility of dominating modifications

The *simplified-design* heuristic creates dominating modifications using two main methods, namely *relaxation* and *padding*.

Modification relaxation uses the dominance relation between modifications (Definition 1) to generate modifications guaranteed to be at least as beneficial as the original ones. Applying a relaxed modification is guaranteed to produce admissible estimates since, by definition, the relaxed modification is guaranteed to return a value that is no lower than the original modification. It is worth noting that the relaxed modification is not necessarily applicable in reality, yet may result in a model for which utility is calculated more efficiently.

In Example 1, we can estimate the value of applying a high friction tile that reduces the probability of slipping from 50% to 10%, by using the value of applying a relaxed hypothetical modification that reduces the probability of slipping to 0. Ignoring the probabilistic nature of the modified environment potentially reduces the computational overhead of the actual setting.

Another type of a dominating modification is created via *modification padding*, which involves appending to the explored modification a sequence of modifications.

Definition 2 (padded modification) *Given an ER-UMD model* $\omega = \langle \epsilon^0, \mathcal{R}, \gamma, \mathbf{\Delta}, \mathcal{F}, \Phi \rangle$*,* $\vec{\delta} = \langle \delta_1, \ldots, \delta_n \rangle$ *is a* padded modification *of* $\delta \in \mathbf{\Delta}$ *in* ω *if* $\exists i \ 1 \leq i \leq n$ *s.t.* $\delta = \delta_i$*.*

As opposed to modification relaxation, the benefit of applying modification padding does not lie in the ability to create models that are necessarily easier to solve. Instead, this approach potentially reduces the computational effort of the search by avoiding redundant evaluations of modifications that affect aspects of the model that have no impact on the agent's expected utility. Particularly, we can cache values of previously computed nodes (and their padded sequences) and reuse theses values for 'similar' nodes that represent modifications that are mapped to the same padded sequence.

In Example 1, modification padding can be implemented by estimating the value of removing a single piece of furniture, by the value of removing all pieces of furniture from an entire cell (a black rectangular in Figure 1(right)).

Naturally, both techniques can be combined by first applying a modification relaxation and then padding it with a sequence of additional modifications. We call this a *relaxed padded modification*, which definition is an immediate extension of definitions 1 and 2. Note that modification relaxation is a special case of relaxed modification padding when the sequence appended to the modification is empty. Similarly, modification padding is also a special case where a modification δ is mapped to itself and then padded.

While using modification relaxation always yields admissible estimates, padding sequences may under-estimate the value of a modification. We show that when an ER-UMD model is both independent (modification sequences applied in any order yield the same result) and monotonic (no modifications can reduce agent utility), sequence padding never under-estimate a modification and can therefore be used to extract admissible estimates. Formally,

Definition 3 (monotonic model) *An ER-UMD model* ω *is* monotonic *if for every modification* $\delta \in \mathbf{\Delta}$

$$\mathcal{V}^*(\omega) \leq \mathcal{V}^*(\omega^\delta)$$

Definition 4 (independent model) *An ER-UMD model* ω *is* independent *if for any modification sequence* $\vec{\delta} \in \vec{\mathbf{\Delta}}$*, and modification sequence* $\vec{\delta}'$ *that is a permutation of* $\vec{\delta}$*,*

$$\mathcal{V}^*(\omega^{\vec{\delta}}) = \mathcal{V}^*(\omega^{\vec{\delta}'})$$

Lemma 2 *Given a monotonic independent ER-UMD model* ω*, a modification* δ *and a relaxed padded modification* $\vec{\delta}$*,*

$$\mathcal{V}^*(\omega^\delta) \leq \mathcal{V}^*(\omega^{\vec{\delta}})$$

Proof: (sketch) Since the model is independent, we can apply the modifications in $\vec{\delta}$ in any order. In particular, we can first apply the modification in $\vec{\delta}$ that dominates δ and get a value that overestimates $\mathcal{V}^*(\omega^\delta)$. Since the model is monotonic applying the additional modifications in the sequence are guaranteed to be at least as high as $\mathcal{V}^*(\omega^\delta)$. ∎

Corollary 1 *The* simplified-design *heuristic is admissible in any monotonic and independent ER-UMD model* ω*.*

The proof for Corollary 1 is immediate from Lemma 2.

Automatic Dominating Modification Generation

We now show two examples of how dominating modifications can be automatically generated. First, to characterize models where modification padding is easily implemented we focus our attention on *lifted* modifications that represent a set of parameters whose (grounded) instantiations define single modifications. Each lifted modification $\delta(p_1, \ldots, p_n)$ is characterized by a set of parameters $p_1, \ldots, p_n$ and a set of valid values $dom(p_i)$ for each parameter p_i. A (grounded)

modification $\delta(v_1, \ldots, v_n)$ is a valid assignment to all parameters s.t. $v_i \in dom(p_i)$.

For lifted modifications, modification padding can be implemented using *parameterized padding* by mapping a grounded modification to a sequence of modifications that share the same values on a set of lifted parameters. In Example 1, the lifted representation of furniture removal modifications is represented by $ClearCell(x, y)$, where parameters x and y denote the cell coordinates. The value of the grounded modification $ClearCell(1, 3)$ can be (over)estimated by the value of applying the sequence $ClearCell(1, 1)$, $ClearCell(1, 2)$, $ClearCell(1, 3)$, *etc.* This value is cached, so when modification $ClearCell(1, 2)$ is examined, it is mapped to the same padded sequence, and the pre-computed value can be reused.

In models were the essence of modifications involve changing the probability distribution of an action's outcome, we can automatically create a relaxation by creating a separate action for each of the outcomes (known in the literature as *all outcome determinization (Yoon* et al. *2007)*). Continuing with Example 1, for a modification that adds high friction tiles to reduce the probability of slipping from 50% to 10%, applying all outcome determinization creates a hypothetical dominating modification by allowing an agent to choose between two deterministic actions, either slipping or not.

Modifications for Independent Monotonic ER-UMD models

To characterize monotonic and independent models where modification padding can be used to produce admissible estimates, we define *action addition* modifications that add applicable actions to some states of the model. We then show that ER-UMD models that allow only action addition modifications are both independent and monotonic.

To formally define action addition modifications, we let $app(s, \epsilon) \subseteq A$ represent the actions applicable in state s of environment ϵ.

Definition 5 (action addition modification) *A modification δ is an* action addition *modification (ADM) if for any environment $\epsilon \in \mathcal{E}$, ϵ^δ is identical to ϵ except that for every state $s \in S$ there exists a (possible empty) set of actions $A_{s,\delta}$ s.t. $app(s, \epsilon^\delta) = app(s, \epsilon) \cup A_{s,\delta}$.*

In Example 1, action addition is exemplified by enabling safe transitions between nearby states implemented by adding to the model actions with a reduced probability of slipping.

Lemma 3 *An ER-UMD model with only action addition modifications is independent and monotonic.*

Proof: (sketch) Every action applicable in any state of the original model is applicable in the modified one. The expected utility of the initial state cannot be reduced as a result of applying a modification and is therefore monotonic. Following Definition 5, any two modifications $\delta, \delta' \in \mathbf{\Delta}$ can be applied in any order to yield the same set of applicable actions. This can be applied for any pair of modifications in a sequence, indicating that the model is independent. ■

It is worth noting that all modification used by Keren *et al.* (2017), including those implemented as initial state modifications, were in fact action addition modifications since they changed the initial state in such a way that enabled more actions in some of the states reachable from the initial state. For example, removing a piece of furniture in Example 1 can be modeled as enabling the movement to a previously occupied cell. In general, however, not all initial state modifications are monotonic. For example, when we remove from the initial state a fact that is a precondition of an action or add a fact that is a negative precondition, we may cause an action to become non-applicable and reduce utility.

Empirical Evaluation

Our empirical evaluation is dedicated to measuring the effectiveness of the proposed *simplified-design* heuristic on a variety of independent monotonic ER-UMD models, comparing it to the previously suggested *DesignComp* compilation and *simplified-environment* heuristic (Keren *et al.* 2017). We examined the benefits of using heuristic search and caching for utility maximizing design and analyze the role of different heuristics in solving the underlying MDPs.

We used a total of 20 instances from four PPDDL domains (5 of each), adapted from Keren *et al.* (2017) that included three stochastic shortest path MDPs with uniform action cost domains from the probabilistic tracks of the eighth International Planning Competition: Blocks World (IPPC08/BLOCK), Exploding Blocks World (IPPC08/EX. BLOCK), and Triangle Tire (IPPC08/TIRE). In addition, we used the vacuum cleaning robot setting adapted from Keren *et al.* (2017) and described in Example 1 (VACUUM). It is worth noting that the VACUUM domain is tailored to test the utility maximizing design setting and the ability to improve upon an initial design. In all domains, agent utility is expressed as expected cost and constraints as a design budget. For each domain, we used the modifications described by Keren *et al.* (2017), implementing all of them as action addition modifications (see Section for a detailed explanation). Accordingly, all the models we have tested are independent and monotonic, which means that all our generated estimations are admissible and therefore over-estimate $\mathcal{V}^*(\omega^\delta)$ for any model ω and modification δ.

Setup Evaluation was performed using optimal and suboptimal solvers within a time bound of five minutes. Each instance was solved using the following approaches:

- BFS - an exhaustive breadth first search in the space of modifications.
- *DesignComp* (Keren *et al.* 2017) (DC)- a compilation of the design problem to a planning problem, which embeds the design into the domain description.
- BFD (Keren *et al.* 2017) - Best First Design, a heuristic best first search in the space of modifications. For this approach we examined five heuristic approaches, the first of which was presented by Keren *et al.* (2017) and the other four are variations of the heuristic proposed in this work.

			BFS		DC		BFD rel-env		BFD rel-mod		BFD rel-combined		BFD rel-proc		BFD rel-combined-proc	
		V^*	time	nodes	time	nodes	time	nodes	time	nodes	time	nodes	time	nodes	time	nodes
BLOCKS	B=1	0.91	1.5	2709.4(95.2)	1.27	1624.2(559.4)	1.97	2709.4(95.2)	1.7	2709.4(95.2)	1.9	2709.4(95.2)	1.59	2611 (95.2,7.3)	1.7	2611 (95.2,7.3)
	B=2	0.91	2.58	42854.5(2483.5)	2.55	24442.3(7440.2)	3.43	42854.5(2483.5)	3.01	42854.5(2483.5)	3.5	42854.5(2483.5)	2.67	40396.4(2483.5,25.4)	2.86	40396(2483.5,25.4)
	B=3	0.91	37.6	441153.2(35901.4)	42.4	244523.5(67939.8)	59.48	441153.2(35901.4)	48.4	441153.2(35901.4)	58.46	441153.2(35901.4)	41.62	441153.2(35901.4,63.6)	35.8	441153.2(35901.4,63.6)
EX. BLOCKS	B=1	0.23	30.82	15724.4(41.4)	25.61	10839(118.6)	31.38	15724.4(41.4)	35.82	15724.4(41.4)	36.57	15724.4(41.4)	35.63	15720(41.4,7.6)	35.92	15720(41.4,7.6)
	B=2	0.01	272.2	13171.9(458.7)	45.55	2794.5(819.5)	275.9	13171.9(458.7)	275.3	13171.9(458.7)	282.3	13171.9(458.7)	171.7	8812.4(427.7,25.8)	185.6	8812.4(427.7,25.8)
	B=3	0.01	TO	TO	17.09	6251.2(3541.4)	TO	TO	TO	TO	TO	TO	527.832	1452884(2498,63)	523.2	1452884(2498,63)
TIRE	B=1	0.86	0.9	2343.2(35.5)	0.5	1074.2(79)	1.4	2343.2(35.5)	1.4	2343.2(35.5)	1.4	2343.2(35.5)	1.4	2343.2(35.5,5.4)	0.9	2313(35.5,5.4)
	B=2	0.83	0.58	14189.2(333.4)	0.38	6418.4(479.2)	0.67	14189.2(333.4)	0.61	14189.2(333.4)	0.62	14189.2(333.4)	0.6	14189.2(333.4,16.5)	0.6	14189.2(333.4,16.5)
	B=3	0.81	2.4	54343.41(1741.2)	1.6	54343.41(1741.2)	2.5	54343.41(1741.2)	2.4	54343.41(1741.2)	2.4	54343.41(1741.2)	2.3	54343.41(1741.2,34.3)	2.3	54343.41(1741.2,34.3)
VACUUM	B=1	0.75	6.25	5553.2(14.3)	0.91	977.4(15)	7.43	5553.2(14.3)	7.8	5553.2(14.3)	7.8	5553.2(14.3)	6.7	5542(14.3,3.4)	7.3	5542(14.3,3.4)
	B=2	0.67	18.24	15367.2(56.3)	4.68	3079.5(61.6)	23.68	15367.2(56.3)	19.04	15367.2(56.3)	31.93	15367.2(56.3)	18.59	15317.5(56.4,6)	25.5	15317.5(56.4,6)
	B=3	0.56	43.48	29257.4(150.4)	21.36	7140(182.2)	66.87	29257.4(150.4)	33.8	29257.4(150.4)	73.26	29257.4(150.4)	33.08	29115.4(150.4,8.2)	47.86	29115.4(150.4,8.2)

Table 1: Running time and expanded node count for optimal solvers with h_{BAOD}

			BFS		DC		BFD rel-env		BFD rel-mod		BFD rel-combined		BFD rel-proc		BFD rel-combined-proc	
		V^*	time	nodes	time	nodes	time	nodes	time	nodes	time	nodes	time	nodes	time	nodes
BLOCKS	B=1	0.91	1.49	2487.4(95.2)	1.97	1938.5(938.5)	1.43	2487.4(95.2)	1.9	2487.4(95.2)	4.4	2487.4(95.2)	1.9	2399.2(95.2,7.4)	4.4	2399.2(95.2,7.4)
	B=2	0.91	1.92	39425.4(2485.5)	1.89	27891.2(12867.4)	9.35	39425.4(2483.5)	3.4	39425(2483.5)	9.25	39425(2483.5)	4.2	36967(2483.5,25.4)	9.7	36967(2483.5,25.4)
	B=3	0.91	27.88	406762.4(35901.4)	38.93	271487.2(114492.2)	117.4	406762.5(35901.4)	59.60	406762.6(35901.4)	127.93	406762.5(35901.4)	54.72	370924.7(35901.4,63.6)	131.35	370924.7(35901.4,63.6)
EX. BLOCKS	B=1	0.23	44.23	505293.2(41.4)	34.83	124678.4(1354)	46.9	505293.2(41.4)	47.89	505293.2(41.4)	46.72	505293.2(41.4)	50.63	505259.2(41.4,7.3)	49.6	505259.2(41.4,7.3)
	B=2	0.01	344.32	3916380.5(458.4)	45.97	42741.6(9459.6)	348.8	3916380.5(458.4)	220.3	2551503.3(427.4)	231.4	2551503.3(427.4)	231.4	2551101.6(427.4,25.4)	225.5	2551101.6(427.4,25.4)
	B=3	0.01	TO	TO	43.2	59802.2(26763.4)	TO	TO	TO	TO	TO	TO	TO	TO	TO	TO
TIRE	B=1	0.86	1.4	2920.6(35.5)	0.7	2920.6(35.5)	1.2	2920.6(35.2)	1.2	2920.6(35.5)	1.5	2890.6(35.5)	1.2	2890.6(35.5,5.4)	1.5	2890.6(35.5,5.4)
	B=2	0.83	0.63	17635.4(333.4)	0.4	17635.4(333.4)	0.84	17635.4(333.4)	0.72	17635.4(333.4)	0.96	17635.4(333.4)	0.7	17635.4 (333.4,16.5)	0.8	17635(333.4,16.5)
	B=3	0.81	2.5	67464.2(1741.2)	1.7	31450.2(4702.2)	3.03	67464.2(1741.2)	2.7	67464.2(1741.2)	3.4	67464.2(1741.2)	2.7	65760.4(1741.2,37.3)	3.4	65760.4(1741.2,37.3)
VACUUM	B=1	0.75	7.91	6903.4(14.3)	9.02	1200.2(227.2)	14.71	6903.4(14.3)	17.67	6903.4(14.3)	37.69	6903.4(14.3)	18.03	6892.2(14.3,3.4)	36.03	6892.2(14.3,3.4)
	B=2	0.67	37.4	18617.4(56.3)	52.3	4181(202)	99.23	18617.3(56.3)	91.93	18617.3(56.3)	394.29	18617.3(56.3)	91.44	18567.4(56.3,6.2)	411.76	18567.5(56.3,6.2)
	B=3	0.56	83.46	34181.4(150.4)	79.04	10165.2(442.2)	264.49	34181.4(150.4)	228.66	34181.4(150.4)	1034.48	34181.4(150.4)	204.98	34039.2(150.4,8.2)	225.6	34039.2(150.4,8.2)

Table 2: Running time and expanded node count for optimal solvers with h_{MinMin}

- **rel-env** the *simplified-environment* heuristic where node evaluation is done on a relaxed environment.
- **rel-mod** the *simplified-design* heuristic that estimates the value of a modification by a single dominating modification.
- **rel-combined** the *simplified-design* heuristic that estimates the modification value by a single dominating modification in a relaxed environment.
- **rel-proc** the *simplified-design* heuristic that estimates the modification value using parametrized padding (on the first parameter of a modification).
- **rel-combined-proc** the *simplified-design* heuristic that estimates the value of a modification using parametrized padding of relaxed modifications (on the first parameter of a modification).

Optimal solutions were acquired using a deterministic best first search for the design space for the BFD. We used the solutions of LAO* (Hansen and Zilberstein 1998) for calculating the exact values of BFD execution nodes and the DC (compilation) with convergence error bound of $\epsilon = 10^{-6}$. Approximate solutions were obtained by replacing LAO* with FLARES (Pineda *et al.* 2017), a sampling-based SSP solver that uses short-sighted labeling to cache the value of explored states and speed-up computation; we used the parameters $t = 0$, $\epsilon = 0.01$, and a maximum of 100 iterations. We also assigned a cap of 500.0 for the maximum state cost (100.0 for the approximate case), in order to handle dead-ends (Kolobov *et al.* 2012). In the approximate solution of the compilation approach, the expected cost is computed from 100 simulations of the policy computed by FLARES. For the suboptimal solutions acquired by BFD, the cost after applying modifications is the state cost estimated by FLARES, which is guaranteed to be admissible.

Optimal solutions were tested on an Intel(R) Xeon(R) CPU X5690 machine with a budget of $1 - 3$. Approximate solutions were tested on Intel(R) Xeon(R) CPU E3-1220 3.40Ghz, with a budget of $1 - 3$. To implement the budget constraint we added a counter verifying the number of design action does not exceed the budget. Each run had a 30 minutes time limit.

For solving the underlaying MDP(for both the BFD and compilation), we used two heuristic. The *Min-Min* heuristic (Bonet and Geffner 2005)(h_{MinMin}) solves the all outcome determinization using the zero heuristic. We also implemented the *bounded all-outcome determinization* (heuristic h_{BAOD}) performs a depth-bounded BFS exploration of the all outcome determinization. Both heuristics are admissible.

Results Separated by domain and budget, Table 1 and Table 2 summarize the average results acquired for each domain and each budget ($B = i$) using an optimal solver with h_{BAOD} and h_{MinMin} as the MDP heuristics, respectively. The tables present V^* as the reduction in expected utility for each design budget with respect to the initial utility (the values are the ratio with respect to the initial value). In addition, they present the running time in seconds ($time$) and the number of expanded nodes (evaluated by the heuristic) during the search ($nodes$). The number of design nodes, representing a modification sequence being applied, is in parenthesis. For the **rel-proc** and **rel-combined-proc** heuristics the numbers in parenthesis represent the number of expanded design node and the number of explicitly calculated design nodes, nodes for which the heuristic value of the dominating modification could not be found in the cache. TO indicates a time out for problems that exceeded the time bound.

Table 3 specifies the results acquired using the approximate solver with h_{BAOD} as the MDP heuristic. V^* represents the ratio between the simulated value and the one ac-

		BFS				DC				BFD rel-env				BFD rel-mod				BFD rel-combined				BFD rel-proc				BFD rel-combined-proc			
		V^*	stderr	time	nodes	V^*	stderr	time	nodes	V^*	stderr	time	nodes	V^*	stderr	time	nodes	V^*	stderr	time	nodes	V^*	stderr	time	nodes	V^*	stderr	time	nodes
BLOCKS	B=1	1.05	0.45	0.28	4316.2(190.3)	0.95	0.43	0.48	3827.3(652.6)	1.09	0.48	0.3	4242.5(190.3)	1.14	0.49	0.28	4229.3(190.3)	1.14	0.49	0.3	4152.4(190.3)	1.07	0.45	0.28	4036.3(190.3,7.4)	1.07	0.46	0.288	4036.4(190.3,7.4)
	B=2	1.07	0.49	6.27	65772.2(4966.2)	1.02	0.44	4.02	58102.5(9388)	1.06	0.46	6.24	65772(4966.2)	1.02	0.43	5.48	66093(4966.2)	1.02	0.47	5.58	66089(4966.2)	1.02	0.45	6.5	60623(4966.2,25.5)	1.02	0.43	6.47	60623(4966.2,25.5)
	B=3	1.02	0.43	80.68	693387.5(71802.2)	1.02	12.68	58.4	582887.5(96126.9)	1.03	0.46	101.24	692854.5(71802.2)	1.02	0.45	93.78	692558.6(71802.2)	1.07	0.45	105.49	693245.6(71802.2)	1.01	0.41	82.42	622343.2(71802.2,63.5)	1.2	0.49	82.73	621769.5(71802.2,63.5)
EX. BLOCKS	B=1	1.9	23.98	12.47	134918.4(82.2)	1.17	16.49	2.8	5601.2(170.2)	1.9	12.3	12.58	134191(82.2)	1.2	16.08	42.77	132526.4(82.2)	1.9	21.2	11.7	130487.7(82.2)	1.9	13.06	13.2	129957.5 (69.4,7.5)	1.9	23.47	11.55	136625.5(69.4,7.5)
	B=2	0.01	0	73.5	768204.4(916.3)	0.01	0	16.31	25329.6(1226.3)	0.01	0	71.71	771082.4(916.3)	0.01	0	65.07	778786.5(916.3)	0.01	0	76.8	782888.7(916.3)	0.01	0	31.5	180761(420.7,25.9)	0.01	0	50.4	495917.8(455.7,25.9)
	B=3	0.01	0	443.3	2935654.4(5622.3)	0.01	0	114.9	98556.5(6009.4)	0.01	0	426.4	2940276.5(5622.2)	0.01	0	385.6	2909483.4(5622.3)	0.01	0	288.58	2925312.2(5622.3)	0.01	0	100.4	338548(2526,63)	0.01	0	102.6	334572(2526,63)
TIRE	B=1	1.48	0.2	0.41	2612.3(70.2)	2.1	24.779	0.03	1083(171.6)	1.09	0.49	0.43	2982(70.2)	1.02	0.49	0.42	3351.5(70.2)	1.02	0.48	0.43	3593.5(70.2)	1.18	0.2	0.41	2998(635.4,5.6)	1.17	0.49	0.42	3329.5(63.4,5.5)
	B=2	1.03	0.45	0.7	16672.4(666.4)	2.1	24.9	0.24	5087.5(916.5)	1.03	0.47	0.65	16608.8(666.4)	1.02	0.44	0.67	17618.6(666.4)	1.03	0.48	1.2	16580.8(666.4)	1.05	0.49	0.68	17547.5(614.6,16.4)	1.04	0.43	0.64	15789.7(494.3,16.4)
	B=3	1.02	0.45	2.52	58570.4(3482.4)	2.4	22.4	0.7	19412.5(3535.3)	1.03	0.45	2.52	58570.5(3482.2)	1.15	0.45	2.37	56487.6(3482.4)	1.13	0.21	2.54	58807.8(3482.9)	1.03	0.49	2.42	57306.7(3402.5,37.3)	1.04	0.48	2.42	55508(3407.4,37.3)
VACUUM	B=1	1.45	0.47	11.21	6075.6(728.3)	1.06	0.45	2.75	2090.3(27.5)	1.51	0.48	10.46	5848.6(28.5)	1.05	0.43	9.08	5140.3(28.5)	1.04	0.44	10.2	4917.6(28.5)	1.03	0.43	8.32	4873.6(28.5,3.3)	1.04	0.43	9.4	4895(28.5,4.4)
	B=2	1.44	0.47	26.01	13414.5(1121.4)	0.01	0.43	9.08	6736.5(102.6)	1.01	0.41	28.7	11937.5(112.4)	1.01	0.42	22.74	11679.6(112.4)	0.01	0.42	37.52	11981.7(112.4)	1.4	0.48	25.67	13028.5(112.4)	1.4	0.48	25.7	13028.5(112.4)
	B=3	1.4	0.49	33.84	20071.5(300.4)	1.01	0.43	17.95	13767.6(233.2)	1.46	0.49	16.9	19752.6(300.4)	1.4	0.46	51.92	20306.7(300.4)	1.4	0.46	109.841	20306.7(300.4)	1.01	0.41	46.4	18364.7(300.4)	1.02	0.45	70.3	18372.7(300.4)

Table 3: Running time for sub-optimal solvers using the h_{BAOD} heuristic

quired using the optimal solver, $stderr$ represents the standard deviation, while $time$ and $nodes$ have the same meaning as in the optimal solver's tables.

Our first observation is that with regards to the optimal solutions, the compilation (DC) approach outperforms the BFD approach for most domains with a shorter running time and less expanded nodes. The only exception occurs in the BLOCKS domain for budget 3 and the h_{BAOD} MDP heuristic, for which **rel-combined-proc** outperforms the other approaches and the same setting with h_{MinMin} as the MDP heuristic, for which the BFS approach was best. It is worth noting, however, that the number of design nodes, each corresponding to a modification sequence, is higher for DC then for all BFD approaches.

Our evaluation uses only independent models. Therefore, for any budget above 1, the BFD approach examines all the permutations of a given modification sequence separately, while for the compilation, the value for these nodes is computed only once. However, the use of independent models does not explain the superior performance of the compilation over the BFD approach for budget 1 as well.

To examine this trend further, we compared the number of nodes that are evaluated by the heuristic to the distinct nodes evaluated for the first time (and for which the heuristic value has not been computed). The results show that while the DC examines up to 20% less nodes, the number of distinct nodes for both BFD and DC is similar. We also performed additional evaluations on small instances of the VACCUM domain (2×2 and 3×3) where the BFD approaches, and the **rel-proc** procedure in particular, outperformed the compilation in terms of both running time and expanded nodes. These results show that the efficiency of the applied approach depends on the specific problem structure and indicate that further investigation of both the nature of the benchmarks and the LAO* algorithm are warranted to understand the results and evaluate the efficiency of our proposed methods.

Next, we analyze the use of caching by comparing **rel-env**, **rel-mod**, and **rel-combined** that do not use caching against their counterparts **rel-proc** and **rel-combined-proc** that are applied to a relaxed environment and re-use previously computed costs. The results show the newly proposed heuristics outperform the heuristics proposed by Keren *et al.* (2017) for all instances in terms of running time. This is due to saving in computation gained by the caching of similar modifications. This applies to both h_{MinMin} and h_{BAOD} heuristics.

Comparing h_{BAOD} (Table 1) with h_{MinMin} (Table 2) we note that for most instances the h_{BAOD} outperformed the h_{MinMin} heuristic, both in terms of running time and the number of explored nodes.

Exploring the different heuristic approaches for BFD calculation, we observe that for most domains, the different approaches yield the same number of explored nodes as the blind search (indicated by BFS). The only approaches that reduce the number of calculated nodes are the caching-based approaches, namely **rel-proc** and **rel-combined-proc** that reduce the computational overhead by avoiding redundant computations. This suggests that the relaxations we apply are non-informative in the domains we explore, leaving us with the wish to explore other, more elaborate domains in which the value of the heuristic approaches will be demonstrated.

For the approximate solvers (Table 3), the results indicate that in most cases the solvers we have used managed to achieve a utility reduction that deviated from the optimal design by up to 10%. Notable in particular is the ability to achieve a nearly optimal design for the EX.BLOCKS domain, which could not be solved by all but the DC in the optimal setting. Indeed, as in the optimal case, the DC compilation is the dominating approach for most domains. However, results are inconclusive since in most cases they fail to provide a single computation method that outperforms the other approaches on all measures. This, again, indicates that further investigation is needed into the pros and cons of using sub-optimal solvers.

Related Work

Environment design (Zhang *et al.* 2009) provides a general framework for modifying environments with the objective of maximizing some utility. Keren *et al.* (2017) formulated ER-UMD as a special case of environment design where the objective is to find a sequence of modifications that maximize some agent utility.

For solving ER-UMD settings, two methods were suggested by Keren *et al.* (2017), namely a compilation (DC) that embeds the design problem into a planning problem and heuristic search (BFD) in the space of modifications. For the latter, they suggest applying modifications to a relaxed environment and show it generates an admissible heuristic.

We extend this approach by offering a set of heuristics based on the relaxation of the design process. By searching in the relaxed modification space we potentially avoid the need to calculate the value of every possible modifica-

tion and use cached values to estimate the value of similar modifications. Our approach can be seen as complementary to the previous approaches, since caching and modification relaxation can be combined with environment relaxation to yield estimations that may be computed efficiently.

The modification padding technique we suggest to generate dominating modifications is inspired by pattern database(PDB) heuristic approaches, originally developed for planning problems (Culberson and Schaeffer 1998; Haslum *et al.* 2007; Edelkamp 2006). PDBs are abstraction heuristics that ignore some aspects of a search problem (the *pattern*) in order to create a problem that can be optimally solved efficiently. The key difference between padding and pattern database heuristics is that the former does not necessarily yield an easier-to-solve model. Instead, it potentially avoids redundant computations of irrelevant modification sets, those that do not affect the agent's expected utility.

As noted by Keren *et al.* (2017), it is the relationship between agent and system utility that dictates the types of methods that can be used to solve an environment design problem. In particular, for ER-UMD we exploit the correlation between agent and system utilities to develop planning-based methods for design. The heuristics we propose (and show to be admissible) are not admissible for environment design in general and in particular not for goal recognition design (Keren *et al.* 2014) or policy teaching (Zhang and Parkes 2008).

Conclusions

This work proposed a new class of heuristics for ER-UMD, called *simplified-design*, which relax the modification process by mapping each modification that is expanded during the redesign to a modification that *dominates* it. Instead of the original modification, we calculate the value of the dominating one and cache the computed value for future use. We identified conditions under which this heuristic class is admissible and discussed automatic generation of relaxations.

For future work, we intend to automate the process of selecting the best relaxation approach for a given domain. In addition, we intend to implement an approach that may alternate during the search between different levels of *relaxation granularity*; for padded modification sequences that yields a utility gain, a more accurate (and costly) estimation is acquired while for padded sequences that leave the initial utility unchanged, we use the high level value.

References

Dimitri P. Bertsekas. *Dynamic programming and optimal control*, volume 1. Athena Scientific Belmont, MA, 1995.

Blai Bonet and Héctor Geffner. mGPT: A probabilistic planner based on heuristic search. *Journal of Artificial Intelligence Research*, 24:933–944, 2005.

Blai Bonet, Gábor Loerincs, and Héctor Geffner. A robust and fast action selection mechanism for planning. In *AAAI/IAAI*, pages 714–719, 1997.

Joseph C Culberson and Jonathan Schaeffer. Pattern databases. *Computational Intelligence*, 14(3):318–334, 1998.

Stefan Edelkamp. Automated creation of pattern database search heuristics. In *International Workshop on Model Checking and Artificial Intelligence*, pages 35–50. Springer, 2006.

Eric A. Hansen and Shlomo Zilberstein. Heuristic search in cyclic AND/OR graphs. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 412–418, 1998.

Patrik Haslum, Adi Botea, Malte Helmert, Blai Bonet, Sven Koenig, et al. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *AAAI*, volume 7, pages 1007–1012, 2007.

Sarah Keren, Avigdor Gal, and Erez Karpas. Goal recognition design. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS)*, pages 154–162, 2014.

Sarah Keren, Luis Pineda, Avigdor Gal, Erez Karpas, and Shlomo Zilberstein. Equi-reward utility maximizing design in stochastic environments. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI 2017)*, August 2017.

Andrey Kolobov, Daniel Weld, et al. A theory of goal-oriented mdps with dead ends. *arXiv preprint arXiv:1210.4875*, 2012.

Luis Enrique Pineda, Kyle Hollins Wray, and Shlomo Zilberstein. Fast ssp solvers using short-sighted labeling. In *AAAI*, pages 3629–3635, 2017.

Sung Wook Yoon, Alan Fern, and Robert Givan. FF-Replan: A baseline for probabilistic planning. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling (ICAPS)*, pages 352–359, 2007.

Haoqi Zhang and David Parkes. Value-based policy teaching with active indirect elicitation. In *Proceedings of the Twenty-Third Conference on Artificial Intelligence (AAAI)*, pages 208–214, 2008.

Haoqi Zhang, Yiling Chen, and David Parkes. A general approach to environment design with one agent. In *Proceedings of the Twenty-First International Joint Conference on Artifical Intelligence*, pages 2002–2008, 2009.

Analyzing Tie-Breaking Strategies for the A* Algorithm

Augusto B. Corrêa[1], **André G. Pereira**[2] and **Marcus Ritt**[2]
[1] University of Basel, Switzerland
[2] Federal University of Rio Grande do Sul, Brazil
{abcorrea, agpereira, marcus.ritt}@inf.ufrgs.br

Abstract

For a given state space and admissible heuristic function h there is always a tie-breaking strategy for which A* expands the minimum number of states (Dechter and Pearl 1985). We say that these strategies have optimal expansion. Although such a strategy always exists it may depend on the instance, and we currently do not know a tie-breaker that always guarantees optimal expansion. In this paper, we study tie-breaking strategies for A*. We analyze common strategies from the literature and prove that they do not have optimal expansion. We propose a novel tie-breaking strategy using cost adaptation that has always optimal expansion. We experimentally analyze the performance of A* using several tie-breaking strategies on domains from the IPC and zero-cost domains. Our best strategy solves significantly more instances than the standard method in the literature and more than the previous state-of-the-art strategy. Our analysis improves the understanding of how to develop effective tie-breaking strategies and our results also improve the state-of-the-art of tie-breaking strategies for A*.

Introduction

A* is the most popular best-first heuristic search algorithm (Hart, Nilsson, and Raphael 1968). It expands states in order of increasing f-values. For a given state s, the function $f(s)$ is the sum of the cost $g(s)$ of the current path from the initial state to state s, and the heuristic cost $h(s)$ from s to a goal state. A heuristic h is *admissible* if it never overestimates the cost of a state to its closest goal state. In this case A* returns an optimal solution path of minimum cost $\mathcal{C}^*$, if there is one. The heuristic that returns the cost of an optimal path for all states is called the *perfect heuristic* h^*. During the search, it is possible to have several states with the same f-value. Hence, A* has to use an order $[f, \tau]$ with a *tie-breaking strategy* τ to select one of them to be expanded next. A* with a deterministic tie-breaking strategy τ defines a unique *expansion sequence* of states.

A state space evaluated by an admissible heuristic h is *nonpathological* if there exists some cost-optimal path where $h(s) < h^*(s)$ for all non-goal states s on it. Dechter and Pearl (1985) have shown that in this case the tie-breaker τ plays no role as the set of states with $f < \mathcal{C}^*$ contains all states expanded by A*. However, if the admissible heuristic h on the state space is *pathological*, then A* will expand all states with $f < \mathcal{C}^*$ and additionally some states with $f = \mathcal{C}^*$. This set of states is known as the *final plateau* or *final f-layer*. There is always a tie-breaking strategy τ that expands, in addition to states with $f(s) < \mathcal{C}^*$, only states on a shortest cost-optimal path in the final f-layer (i.e., states along the cost-optimal path with the least number of operators). In this case, we say that tie-breaking strategy τ has *optimal expansion*, or simply is *optimal*.

Most of the search and planning literature considers breaking ties in favor of smaller h-values to be a good practice (e.g., (Holte 2010; Hansen and Zhou 2007)). Dechter and Pearl (1985) describe A* as being agnostic with regard to the tie-breaking strategy letting it "*break ties arbitrarily, but in favor of a goal state*" and assume that only a few states s will satisfy $f(s) = \mathcal{C}^*$. However, Asai and Fukunaga (2016) showed that this is often false and A* using tie-breaking strategies that do not favor small h-values can solve more instances and expand fewer states.

In many applications the goal is to minimize the use of some resource (e.g., fuel in logistic problems), and operators that do not use this resource can be modeled as having no cost. Based on this observation Asai and Fukunaga (2016) have introduced so-called *zero-cost domains*. In such domains, the final plateau can account for a large part of the expanded states and A* can follow long zero-cost paths that can be avoided by a tie-breaking strategy.

Empirical analysis shows that all IPC instances using A* with heuristic $h^{\text{LM-cut}}$ which are solved in 5 minutes or less are pathological and more than 95% of the zero-cost instances solved using this time limit are also pathological. Hence, tie-breakers are relevant for most of the instances in both benchmarks.

In this paper we study tie-breaking strategies for A*. We first analyze previously proposed tie-breaking strategies and prove that they are not always optimal. We also propose a new strategy which is guaranteed to have optimal expansion. We experimentally analyze the performance of A* using several strategies on the set of IPC instances and instances with zero-cost operators where the perfect heuristic h^* can be computed. In practical settings using $h^{\text{LM-cut}}$ our new strategies solve more instances than other methods in the literature. Our results show how to build an optimal tie-breaking strategy given h^* and our analysis improves the understanding of how to develop tie-breakers.

Background

State Space Let $\mathcal{S} = \langle s_0, S_*, \mathcal{O}, \text{cost} \rangle$ be a *state space*, where s_0 is the initial state, S_* is a set of goal states and $\mathcal{O}$ is a set of operators. For a given state s there is a (possibly empty) subset of operators in $\mathcal{O}$ that can be applied to s to generate a set of *successor states* $\text{succ}(s)$. Every operator $o \in \mathcal{O}$ has a cost $\text{cost}(o) \in \mathbb{R}_0^+$ associated to a transition $s \to s'$, where $s' \in \text{succ}(s)$. A sequence of distinct states denoted as $s_0 \to s_1 \to \cdots \to s_n$ is called a *path*, if for every pair of consecutive states $s \to s'$ we have $s' \in \text{succ}(s)$. If $s_n \in S_*$ then the sequence is called a solution path (s-path).

Tie-Breaking Strategies The A* algorithm receives a state space $\mathcal{S}$ and a heuristic function h as input and outputs an s-path, if there is one, or "unsolvable" otherwise. A* ordering states by $[f, \tau]$ with a *tie-breaking strategy* τ (where $f = g + h$ and τ is some function over $\mathcal{S}$) expands a unique sequence of states $\langle s_0, s_1, \ldots, s_n \rangle$, called the *expansion sequence*. We assume that A* keeps a priority queue denoted as OPEN that sorts the states lexicographically in increasing order of $[f, \tau]$. To expand a state means to remove it from OPEN and to generate all its successors. Note that in this way goal states are only *processed*, i.e. removed from OPEN, but not expanded. If the expansion sequence of A* with a given tie-breaking strategy has the minimum number of states among all possible sequences we say that this strategy has *optimal expansion* – or simply that it is *optimal*. If the function f uses the perfect heuristic h^*, we denote it as $f^* = g + h^*$.

An s-path is not fully informed if $h(s) < h^*(s)$ for all $s \notin S_*$ on that path. Dechter and Pearl (1985) define a state space $\mathcal{S}$ with admissible heuristic h to be *nonpathological* if there exists at least one cost-optimal not fully informed s-path. Conversely, a state space $\mathcal{S}$ with admissible heuristic h is *pathological* if all cost-optimal s-paths are fully informed.

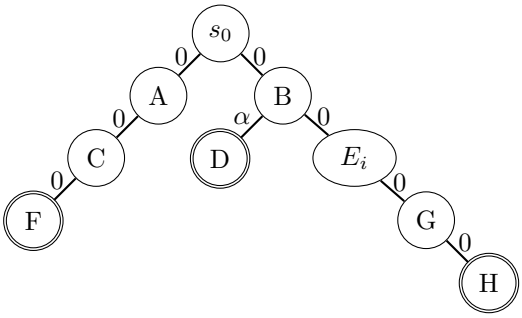
Common Tie-Breaking Strategies

In this section, we present a theoretical framework to analyze tie-breaking strategies for A*. Our framework is based on the perfect heuristic h^* as a fully informed tie-breaker. In state spaces where we can compute h^*, A* with f^* will only expand states whose f-value equals the optimal cost $\mathcal{C}^*$. In this setting, the tie-breaking strategy will have optimal expansion if it only expands states on one cost-optimal s-path with the least number of operators.

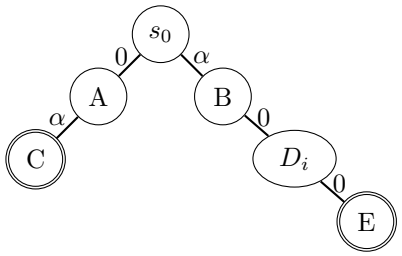
Analyzing h^* as Tie-Breaker

The heuristic search literature usually considers breaking ties by h to be a good approach. Therefore one would expect that when having h^*, we could use its value as a tie-breaker, leading to an strategy with optimal expansion. In this setting, using order $[f^*, h^*]$ means that A* uses f^* as main evaluation function and h^* as tie-breaker, and any remaining ties are solved arbitrarily.

However, using order $[f^*, h^*]$ is not optimal, as it may expand more states than another strategy. Figure 1a shows an example with two paths to goal states using only zero-cost operators. State s_0 is the initial state, doubly-circled states are goals and ellipses represent arbitrarily long transition sequences of zero cost. In this situation, $[f^*, h^*]$ provides no information. Hence, the expansion sequence depends on how remaining ties are solved, which does not guarantee optimal expansion. To reach a goal from s_0, A* may expand three states using the left s-path ($s_0 \to A \to C \to F$), or an arbitrarily large set of states using the right s-path ($s_0 \to B \to \cdots \to G \to H$).



(a) Orders $[f^*, h^*]$ and $[f^*, \hat{h}^*]$ fail.



(b) Order $[f^*, h^*_\epsilon]$ fails.

Figure 1: Instances where tie-breaking by h^*, $\hat{h}^*$, and h^*_ϵ fails.

Analyzing $\hat{h}^*$ as Tie-Breaker

Asai and Fukunaga (2017) propose to use *distance-to-go* heuristics as tie-breakers. A distance-to-go heuristic, denoted $\hat{h}$, uses the same algorithm to compute h but replaces the cost of all operators by one. Thus $\hat{h}^*(s)$ is the minimum number of operator applications necessary to reach a goal state from s. In practice, A* using $[f^*, \hat{h}^*]$ improves coverage in zero-cost domains (Asai and Fukunaga 2017).

However, order $[f^*, \hat{h}^*]$ can also fail to produce an optimal expansion, as the example of Figure 1a shows. Let $\alpha > 0$. After expanding s_0, we have $\hat{h}^*(A) = 2$, because A can reach the closest goal F applying two operators, and $\hat{h}^*(B) = 1$, because B can reach its closest goal D applying only one operator. As a consequence, A* expands state B first. However, the s-path $s_0 \to B \to D$ is not optimal because the operator that enables B to reach goal state F has cost α. Thus $[f^*, \hat{h}^*]$ expands four states ($\langle s_0, B, A, C \rangle$), and the optimal strategy only three ($\langle s_0, A, C \rangle$).

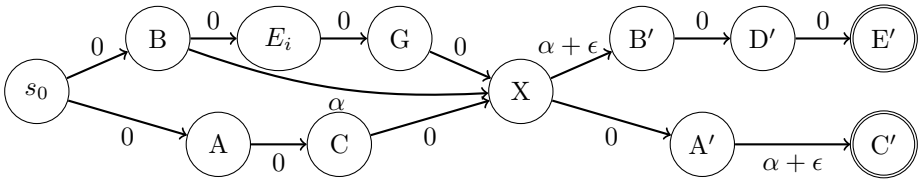


Figure 2: Example of an instance where order $[f^*, h^*_c]$ using cost adaptation fails for any value of c, and $\epsilon > 0$.

Novel Strategies based on Cost Adaptation

The tie-breaking strategy using the perfect heuristic h^* guides the search along a cost-optimal path but fails to identify the cost-optimal path with the least number of operators. The tie-breaking strategy using the distance-to-go heuristic $\hat{h}^*$ guides the search along a path with fewest operators to the goal but fails to estimate the total cost of the path. We can combine both estimates to improve the search performance.

Definition 1 (Cost-adapted heuristic)**.** *Let $\mathcal{S} = \langle s_0, S_*, \mathcal{O}, cost\rangle$ be a state space and h be a heuristic for $\mathcal{S}$. A* cost-adapted heuristic *h_c is a heuristic function for $\mathcal{S}$, where for all $o \in \mathcal{O}$ there is a new operator $o_c \in \mathcal{O}_c$ with $cost(o_c) = cost(o) + c$ and h_c computes the heuristic function by replacing $\mathcal{O}$ by $\mathcal{O}_c$.*

In other words, the cost-adapted heuristic h_c is the same algorithm to compute h on $\mathcal{S}$, but adds a constant c to each operator cost. We will call a tie-breaking strategy based on h_c a method *using cost adaptation*.

Richter, Westphal, and Helmert (2011) introduced the idea of adding one to every operator cost in the satisficing LAMA solver. The intuition is that by doing so, A* can combine the operator cost with the cost of applying an operator. In the special case used in the LAMA solver with $c = 1$ we denote h_c as h_{+1}.

Analyzing h^*_c as Tie-Breaking Strategy

Now, we analyze the behavior of h^*_c for different magnitudes of c. First, consider $c = \epsilon$ where ϵ is a small constant such that $\epsilon \ll \min_{o\in\mathcal{O}}\{\text{cost}(o) \mid \text{cost}(o) > 0\}$. The effect of making ϵ very small is that even for the longest path with l operators, the product $l\epsilon$ is still smaller than the smallest difference between a cost-optimal and a non-cost-optimal s-path. If we apply $[f^*, h^*_\epsilon]$ to the example of Figure 1a it produces the optimal expansion $\langle s_0, A, C\rangle$.

However, $[f^*, h^*_\epsilon]$ can also fail. Figure 1b shows an example where A* with $[f^*, h^*_\epsilon]$ expands three states and the optimal expansion only two. In this example, after expanding s_0, A* can expand A and B, where $h^*_\epsilon(A) = \alpha + \epsilon$ while $h^*_\epsilon(B) = 2\epsilon + |D_i|\epsilon$. Thus, B is chosen for expansion, followed by the sequence of states D_i, leading to goal state E. A* expands the path $s_0 \to B \to \cdots \to E$ instead of the shortest cost-optimal path $s_0 \to A \to C$.

An approach to solve the example of Figure 1b is to use $c = M$, where $M \gg \max_{o\in\mathcal{O}}(\text{cost}(o))$. In Figure 1b breaking ties by h^*_M produces the optimal expansion. Now, $h^*_M(A) = \alpha + M$ and $h^*_M(B) = 2M + |D_i|M$. Since $M \gg \alpha$, A* expands A instead of B, and terminates at the goal state C, leading the search to the optimal expansion sequence $\langle s_0, A\rangle$.

However, h^*_M fails to achieve the optimal expansion in the example of Figure 1a, where we have $h^*_M(A) = 2M$ and $h^*_M(B) = \alpha + M$. Since $M \gg \alpha$, we have $h^*_M(A) > h^*_M(B)$ causing the search to expand B, leading to the same problem of $[f^*, \hat{h}^*]$.

Unfortunately, there is no strategy for selecting c that works universally for any task. Figure 2 shows an example where there is no constant c such that order $[f^*, h^*_c]$ leads to an optimal expansion. The optimal strategy must expand the path $s_0 \to A \to C \to X \to A' \to C'$. However, for $c > \alpha$, after expanding the initial state s_0 we have $h^*_c(B) < h^*_c(A)$ because of the path using the operator with cost α from B to X, and thus A* will expand state B which is not optimal. For $c < \alpha + \epsilon$, on the other hand, after expanding state X, A* will next select state B' since $h^*_c(B') < h^*_c(A')$, but the optimal expansion strategy should expand A' in order to minimize the number of expansions. Since $\epsilon > 0$ for every c one of the two cases will fail. Despite this, cost adaptation will prove to be useful in defining a tie-breaking strategy with optimal expansion.

An Optimal Expansion Strategy with Cost Adaptation

Dechter and Pearl (1985) have shown that for any state space $\mathcal{S}$ and admissible heuristic function h there is always a tie-breaking strategy τ such that A* with $[f, \tau]$ presents optimal expansion. The following theorem presents a strategy using a single tie-breaker that achieves optimal expansion for admissible and consistent heuristic functions h. This expansion strategy only requires the evaluation function f to use a consistent heuristic h – not necessarily h^* – but it stills need h^* for the tie-breaker, which now also considers the g values of the states.

Theorem 1. *For an admissible and consistent heuristic h, A* with order $[g + h, \tau]$ and tie-breaker $\tau = g + h^*_\epsilon$ has optimal expansion.*

Proof. If there is no solution A* will always expand all reachable states and thus has optimal expansion. Otherwise, since h is admissible and consistent, A* will process states by non-decreasing f-values, ending with $f = \mathcal{C}^*$ at some goal state. We will show that A* with tie-breaker τ expands the least number of states in the final f-layer, from which the claim follows, since states with $f < \mathcal{C}^*$ must be expanded by all searches which find an optimal solution.

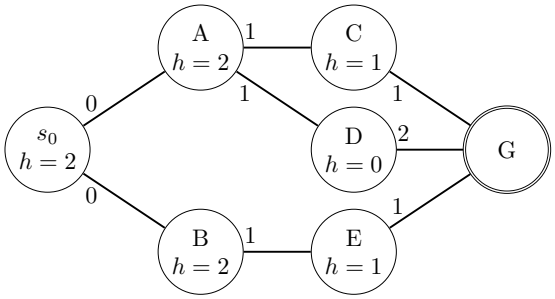


Figure 3: Example of an instance where order $[g + h, g + h^*_\epsilon]$ fails if h is inconsistent. The h-value of each state is also showed.

Consider the moment when for the first time the state of least f-value in OPEN has $f = \mathcal{C}^*$. From this point on all processed states have $f = \mathcal{C}^* = g + h^*$ and therefore are processed in τ-order. For a state s on a cost-optimal path to a goal we have $h^*_\epsilon(s) \leq h^*(s) + \epsilon\bar{d}$, where $\bar{d}$ is an upper bound on the distance from s to some goal, since a non-cost-optimal path from s to some goal costs at least $h^*(s)+\Delta$ for some $\Delta > 0$, and therefore $h^*_\epsilon(s) \leq h^*(s) + \epsilon\bar{d} < h^* + \Delta$, by choice of ϵ.[1]

Thus, for the state s of least τ-value we have $\tau(s) = g(s) + h^*_\epsilon(s) = g(s) + h^*(s) + \epsilon d^*(s)$ where $d^*(s)$ is the shortest distance from s to a goal on some cost-optimal path, and since $g + h^*$ is constant for all states with $f = \mathcal{C}^*$, they are processed in d^*-order. Now, since each state of distance d^* has at least one successor of shortest distance $d^* - 1$ on a cost-optimal path, the distance to the goal decreases in each iteration, and A* expands exactly $d^* - 1$ states before processing a goal state. Since d^* is the shortest distance on a cost-optimal path, optimal expansion follows. □

As a simple consequence of Theorem 1 we have that for A* with the perfect heuristic function h^*, tie-breaker $\tau = g+h^*_\epsilon$ has optimal expansion. Notice that optimal expansion does not imply that A* finds a shortest cost-optimal solution, since the shortest path is guaranteed only for the final f-layer.

The result from Theorem 1 is useful from the following perspective: consider an inadmissible heuristic h where $h(s) = h^*(s)$ in a significant number of states but $h(s) > h^*(s)$ in only a few. Function h cannot be used to guide an admissible search, but Theorem 1 suggests that we can use it as an effective tie-breaking strategy.

Figure 3 illustrates an instance where order $[g+h, g+h^*_\epsilon]$ fails if h is inconsistent. Heuristic values are shown inside each state. To achieve optimal expansion the algorithm should expand paths $s_0 \to A \to C \to G$ or $s_0 \to B \to E \to G$. However, whenever we expand state A, we *must* expand state D as well. Due to the inconsistency of the heuristic function h, we have $f(D) < f(A)$ and $\arg\min_{s \in \text{OPEN}} f(s) = D$, hence this successor must be expanded before than any other successor of A. Since our tie-breaking strategy $g + h^*_\epsilon$ cannot guarantee to favor the expansion of B over the expansion of A, it does not guarantee optimal expansion if h is inconsistent.

If instead of using the optimal expansion strategy from Theorem 1, we use $[f^*, g + h_\epsilon]$ (i.e., the perfect heuristic is used for the evaluation function and not the tie-breaker), we cannot guarantee optimal expansion anymore. Consider the example of Figure 1a and assume that o_α is the operator causing the transition of cost α. Let h be an approximation of h^* that is incapable of capturing the necessity of applying operator o_α – i.e., it considers the cost of operator o_α to be 0. Since A* uses f^*, we have $f^*(A) = f^*(B) = \mathcal{C}^* = 0$ for the successors A and B of s_0. To break this tie, we use $g + h_\epsilon$. We have $g + h_\epsilon(A) = 2\epsilon$ and $g + h_\epsilon(B) = \epsilon$ due to the possible path $s_0 \to B \to D$ where h cannot predict the need of o_α. Hence, B is expanded instead of A, and A* fails to expand only the cost-optimal path with the least number of operators.

Experiments

In our experiments, we tested the improvement of state expansions, search time and coverage for the different methods studied here and previously mentioned in the literature. The experiments use revision 6251 of the Fast-Downward planning system (Helmert 2006) with the modifications of Asai and Fukunaga (2017) and also the same benchmarks as them. In total, we used 1104 instances from the IPC and 620 from the zero-cost benchmarks of Asai and Fukunaga (2017). All experiments have been run on a PC with an AMD FX-8150 processor running at 3.6 GHz and 32 GB of main memory. In the case where τ cannot solve all ties, the remaining ones are broken by FIFO order.

Comparing Theory and Practice

We first focus on the question if the theoretical advantage of cost adaptation strategies translates into practice. For these experiments we use a time limit of 30 minutes, a memory limit of 4 GB, and the subset of 183 IPC and 87 zero-cost domains, which could be solved optimally by all methods given these limits and the internal limits of Fast-Downward to build h^*. Thus, this reduced set of benchmarks contains instances with smaller state spaces than usual.

Table 1 reports the geometric mean of the number of expanded states for different combinations of primary A* heuristic and tie-breaker. For each combination, the table shows the results for IPC and zero-cost domains separately. The pair at the header of each column is denoted by h^1, h^2, where h^1 was used as the heuristic for the function f and h^2 as the heuristic for the tie-breaking strategy. The best results in each column are shown in bold. We can see that using the benchmarks with a small state space, A* expands few states.

We first analyze the theoretical predictions using the perfect heuristic h^* in function f and as tie-breaker. The results are in the first two columns of Table 1. In practice, the theoretically optimal tie-breaker $g + h^*_\epsilon$ performs best, and strictly dominates the other tie-breakers on zero-cost domains.

In the second combination we relax the tie-breaker to $h^{\text{LM-cut}}$ (Helmert and Domshlak 2009). As expected, the

[1] For integer costs, we can choose $\epsilon < 1/\bar{d}$.

	h^*, h^*		h^*, $h^{\text{LM-cut}}$		$h^{\text{LM-cut}}$, h^*		$h^{\text{LM-cut}}$, $h^{\text{LM-cut}}$		$h^{\text{LM-cut}}$, h^{FF}	
	IPC	Z	IPC	Z	IPC	Z	IPC	Z	IPC	Z
$[g+h^1, h^2]$	12.05	124.49	13.34	244.44	69.92	549.94	79.24	805.68	80.79	690.19
$[g+h^1, \hat{h}^2]$	**11.78**	13.33	14.28	23.72	69.87	119.02	79.29	172.46	80.88	156.04
$[g+h^1, h^2_{+1}]$	**11.78**	13.37	13.01	**20.18**	69.87	105.57	**79.18**	147.79	80.95	**131.62**
$[g+h^1, h^2_\epsilon]$	**11.78**	13.39	**12.63**	21.33	69.88	105.57	79.36	**144.93**	**79.63**	142.02
$[g+h^1, g+h^2_\epsilon]$	**11.78**	**13.26**	31.91	65.67	**69.84**	**104.67**	80.71	145.88	81.26	141.08

Table 1: Comparison of the geometric mean of the number of expanded states using different heuristics and tie-breaking strategies in IPC domains ("IPC") and zero-cost ("Z") domains.

number of expanded states increases for all tie-breakers, showing that, in fact, tie-breaking strategy matters. The theoretical results do not guarantee an optimal expansion breaking ties by $\tau = g+h^*_\epsilon$ in this case, and indeed we can see that the strategy actually performs worse than other strategies. This can be explained by the fact the $h^{\text{LM-cut}}$ is not fully informed. Thus, when a successor state on a cost-optimal path is generated it tends to have a higher value of $g + h^{\text{LM-cut}}_\epsilon$, and leads A* to first expand less informed states. This effect is less pronounced for tie-breakers not using g.

In the remaining combinations, we switch roles and focus on not fully informed searches using heuristic $h^{\text{LM-cut}}$ with different tie-breakers. In all these cases, A* expands a significantly higher number of states. The fifth and sixth column in Table 1 show the results for breaking ties using h^*. Even though $h^{\text{LM-cut}}$ is not guaranteed to be consistent, we find that the f-values never decrease in about $90\,\%$ of the instances in both benchmarks. Hence, our result from Theorem 1 guarantees optimal expansion for $[g+h^{\text{LM-cut}}, g+h^*]$ in most instances. In fact, all cost adaptation methods have a similar performance on the IPC instances, and the theoretically optimal tie-breaker $g + h^*_\epsilon$ is the best method by a small margin.

We finally relax the tie-breaker to approximations of h^*. Following Asai and Fukunaga (2017) we have selected heuristics $h^{\text{LM-cut}}$ and h^{FF} (Hoffmann and Nebel 2001). Note that heuristic h^{FF} is not admissible, but will not change the optimality of the search when used as a tie-breaker. Both cases expand more states than the optimal strategy, as expected, but the relative performance of the tie-breakers is very similar, with little difference on the IPC benchmark. On the zero-cost domains, breaking ties by $\hat{h}$ is always the worst, and methods using cost adaption are always the best.

Table 1 quantifies the advantage of our theoretically best method on the restricted set of small instances, where h^* can be computed. Yet, some instances still need many expansions when breaking ties using h^* which is a fully informed heuristic. For example, A* with order $[g + h^{\text{LM-cut}}, h^*]$ or $[g + h^*, h^*]$ expands 349.108 states in the first instance of the ELEVATORS-UP domain, while order $[g+h^{\text{LM-cut}}, g+h^*_\epsilon]$ expands 18 states. Instance P04 of the same domain presents a similar behavior. In the ROVERS-FUEL domain, instance P05 has an optimal solution of cost $C^* = 3$ with a length of 22 operators, but the order $[f^*, h^{\text{LM-cut}}]$ expands 272.171 states, while the order $[f^*, g+h^*_\epsilon]$ expands exactly 22 states.

Method	**IPC** (1104)	**Zero-cost** (620)
$[f, h^{\text{LM-cut}}]$	525	237
$[f, \hat{h}^{\text{LM-cut}}]$	531	301
$[f, h^{\text{LM-cut}}_{+1}]$	530	299
$[f, h^{\text{LM-cut}}_\epsilon]$	532	301
$[f, g + h^{\text{LM-cut}}_\epsilon]$	524	300
$[f, h^{\text{FF}}]$	548	251
$[f, \hat{h}^{\text{FF}}]$	557	338
$[f, h^{\text{FF}}_{+1}]$	**562**	**352**
$[f, h^{\text{FF}}_\epsilon]$	559	351
$[f, g + h^{\text{FF}}_\epsilon]$	553	346
$[f, \hat{h}^{\text{FF}}, \langle d \rangle, \text{LIFO}]$	530	328

Table 2: Comparison of the number of solved instances in IPC and zero-cost benchmarks where $f = g + h^{\text{LM-cut}}$.

In summary, all cost adaptation strategies are similar on the IPC instances, but far better than the default tie-breaker h on zero-cost. Our results show that even in small state spaces and using the perfect heuristic h^*, tie-breakers are important, even when not optimal. Still, the heuristic function is more important than the tie-breaker, as the comparison between the second and the third combinations confirms. The last two combinations show that tie-breakers also make a difference in practice, and there is enough room for improvement.

Performance on the Complete Set of Instances

We now turn to the practical performance of tie-breakers using cost adaptation. Our second experiment compares the coverage of different tie-breaking strategies using $f = g + h^{\text{LM-cut}}$ to guide the search on the complete set of 1104 IPC and 620 zero-cost domains. In this experiment we have imposed limits of 4 GB and 5 min for each run, following Asai and Fukunaga (2017).

The results are shown in Table 2. We compare our main cost adaptation methods against the standard methods in the literature and the current best deterministic tie-breaker on zero-cost domains from Asai and Fukunaga (2017) (last row). (The best non-deterministic tie-breaker of Asai and Fukunaga (2017) solves in average 2.3 instances more.)

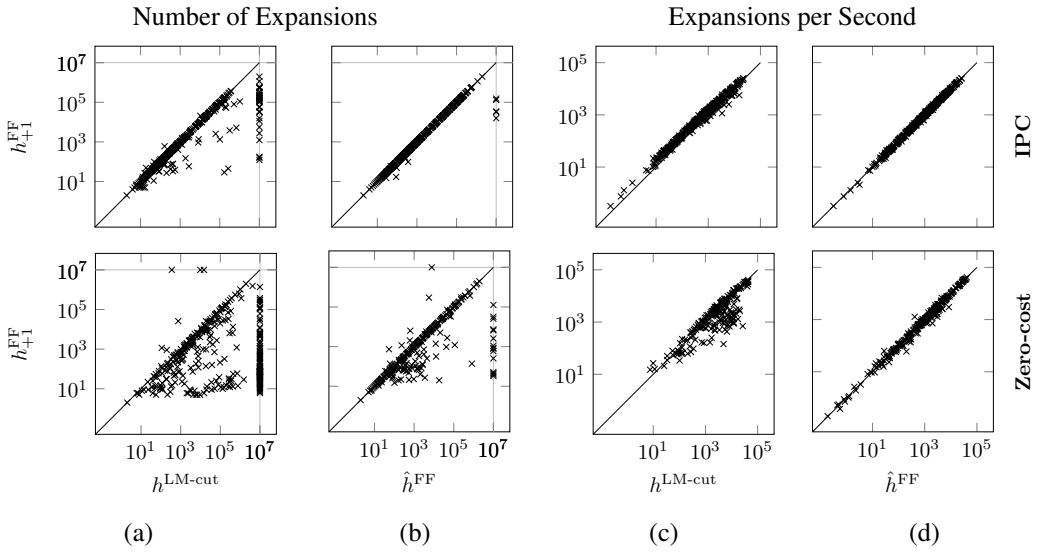


Figure 4: Expansions and expansions per second for IPC (top) and zero-cost (bottom) using A* with $h^{\text{LM-cut}}$ and different tie-breakers (axis).

Looking at the group of tie-breakers using $h^{\text{LM-cut}}$ we find that that all methods using cost adaption perform better than the standard tie-breaker h.

The second group using h^{FF} in the tie-breaker dominates the strategies using $h^{\text{LM-cut}}$ only. This confirms the observation of Asai and Fukunaga (2017) that breaking ties by h^{FF} is better than $h^{\text{LM-cut}}$. However, we find that $\hat{h}^{\text{FF}}$ also performs better on zero-cost domains than their best strategy. This can probably be explained by the difference between processor speeds. Again the tie-breaker $g + h^{\text{FF}}_{\epsilon}$ which is theoretically best for h^* is competitive. The overall best method is h^{FF}_{+1}. It solves five instances more on the IPC benchmark than $\hat{h}^{\text{FF}}$, the best tie-breaker from the literature. The best known tie-breaker for zero-cost instances is $[f, \hat{h}^{\text{FF}}, \langle d \rangle$, LIFO] (Asai and Fukunaga 2017). Here, h^{FF}_{+1} solves 24 instances more.

Figures 4a and 4b compare the number of expanded states of the best method $[f, h^{\text{FF}}_{+1}]$ against the most used method in literature, $[f, h^{\text{LM-cut}}]$ and the best method from the literature $[f, \hat{h}^{\text{FF}}]$. The plots on top show results for IPC instances, the ones on the bottom for zero-cost. We see that tie-breaking with h^{FF}_{+1} expands fewer states on most of the instances compared to $h^{\text{LM-cut}}$, in particular on the zero-cost domains. The number of expanded states compared to $\hat{h}^{\text{FF}}$ is similar in IPC but in zero-cost domains h^{FF}_{+1} outperforms $\hat{h}^{\text{FF}}$ in general.

Another important issue about tie-breaking strategies is the overhead to compute a second evaluation function. Figures 4c and 4d compare the expansions per second of the methods. We find that all methods expand about the same number of states per second, with the exception of $h^{\text{LM-cut}}$ on zero-cost domains.

In general lines, the "pure" cost adaptation methods ($[f, h_c]$) using the h^{FF} heuristic have the best performance. Tie-breaking by h^{FF}_{+1} presents the best coverage in both benchmarks.

Conclusion and Future Work

In this paper, we presented a tie-breaking strategy for A* with h^* that guarantees the minimum number of expanded states among all tie-breaking strategies. Our analysis showed that even for the perfect heuristic h^* previously proposed tie-breakers fail in producing an optimal tie-breaking strategy. Our results showed how to build an optimal tie-breaking strategy for A* for an admissible and consistent heuristic h.

Our experiments confirm the results from Asai and Fukunaga (2017) that tie-breakers have the potential to increase coverage and reduce the number of expanded states. In summary, our best method based on cost adaptation solves 152 instances more than breaking ties by h, the most common tie-breaker in the literature, and more than the two deterministic methods from Asai and Fukunaga (2017) we have tested. Our experiments showed that even in small state spaces and with the perfect heuristic h^*, the performance of A* can be improved by a better tie-breaking strategy. Our main contribution in this work is to provide an analysis that enables a better understanding of the role of tie-breaking strategies in the performance of A*.

Two ideas may be interesting to investigate further. The first is an analysis similar to the one by Helmert and Röger (2008) who investigated for specific domains the performance of A* with almost perfect heuristics. One could do the same with almost perfect tie-breakers. Second, one may study the existence of effective domain-dependent tie-breakers, not based on h^*.

Acknowledgments

This work was supported by FAPERGS as part of project $17/2551-0000867.7$ and was conducted while the first author was a student at the Federal University of Rio Grande do Sul.

References

Asai, M., and Fukunaga, A. S. 2016. Tiebreaking strategies for A^* search: How to explore the final frontier. In *AAAI Conference on Artificial Intelligence*, 673–679.

Asai, M., and Fukunaga, A. 2017. Tie-breaking strategies for cost-optimal best first search. *Journal of Artificial Intelligence Research* 58:67–121.

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

Hansen, E. A., and Zhou, R. 2007. Anytime heuristic search. *Journal of Artificial Intelligence Research* 28:267–297.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Systems Science and Cybernetics* 4(2):100–107.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: what's the difference anyway? In *International Conference on Automated Planning and Scheduling*, 162–169.

Helmert, M., and Röger, G. 2008. How good is almost perfect? In *AAAI Conference on Artificial Intelligence*, volume 8, 944–949.

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

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

Holte, R. C. 2010. Common misconceptions concerning heuristic search. In *Symposium on Combinatorial Search*.

Richter, S.; Westphal, M.; and Helmert, M. 2011. LAMA 2008 and 2011. In *International Planning Competition*, 117–124.

Completeness-Preserving Dominance Techniques for Satisficing Planning

Álvaro Torralba
Saarland University, Saarland Informatics Campus, Saarbrücken, Germany
torralba@cs.uni-saarland.de

Abstract

Dominance pruning methods have recently been introduced for optimal planning. They compare states based on their goal distance to prune those that can be proven to be worse than others. In this paper, we introduce dominance techniques for satisficing planning. We extend the definition of dominance, showing that being closer to the goal is not a prerequisite for dominance in the satisficing setting. We develop a new method to automatically find dominance relations in which a state dominates another if it has achieved more serializable sub-goals. We take advantage of dominance relations in different ways; while in optimal planning their usage focused on dominance pruning and action selection, we also use it to guide enforced hill-climbing search, resulting in a complete algorithm.

Introduction

Satisficing planning is the problem of, given an input planning task, finding a sequence of actions that go from the initial state to a state that satisfies the goal condition. Most satisficing planners use search algorithms like Greedy Best-First Search (GBFS) or Enforced-Hill Climbing (EHC) guided with heuristics such as the delete-relaxation heuristic and extensions thereof (Hoffmann and Nebel 2001; Domshlak *et al.* 2015) plus certain diversification techniques (Richter *et al.* 2011; Röger and Helmert 2010) and/or sub-goal selection strategies (Chen *et al.* 2004; Porteous *et al.* 2001; Hoffmann *et al.* 2004). Both GBFS and EHC use heuristics, but they use them in different ways. In GBFS, heuristics determine the order in which states are expanded. EHC, on the other hand, uses heuristics to compare newly generated states against the initial state, restarting the search when it finds a state with lower heuristic value than the initial state. The success of EHC highly depends on the accuracy of the heuristics. When the heuristic is accurate EHC finds solutions very quickly, but it is incomplete in tasks with unrecognized dead-end states, i.e., states that the heuristic finds promising but have no solution (Hoffmann 2005).

Dominance pruning techniques have recently been introduced for optimal planning (Hall *et al.* 2013; Torralba and Hoffmann 2015). They reduce the search space by pruning states that are dominated by others. The definition of dominance is based on goal distance: a state dominates another state if it can be proven to be at least as close to the goal.

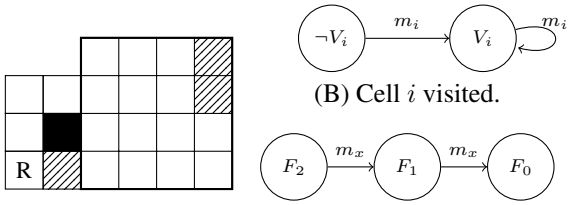
In this paper we explore the use of dominance methods to compare states in satisficing search. We redefine the notion of dominance for satisficing planning, substituting the optimality guarantee by a completeness guarantee that ensures that at least one plan (not necessarily optimal) will be preserved. We also consider how dominance relations can be used to reduce the size of the search space. Like in optimal planning, one can prune states that are dominated by others, but lifting any considerations with respect to the cost of reaching such states. Also, a state s can be replaced by any of its successors s' if s' strictly dominates s. Based on this, we define a variant of EHC that is complete.

Our work builds on previous methods to automatically find dominance relations for a given planning task. We strengthen their reasoning and specialize them for satisficing search. To do this, we define a new dominance relation that serializes the planning task, inspired by sub-goal serialization approaches (Barrett and Weld 1993). Our experiments show that these serialized dominance relations are able to identify dominance in a number of domains to guide a dominance-based EHC.

Background

A *labeled transition system* (LTS) is a tuple $\Theta = \langle S, L, T, s^I, \mathcal{S}^G \rangle$ where S is a finite set of *states*, L is a finite set of *labels*, $T \subseteq S \times L \times S$ is a set of *transitions*, $s^I \in S$ is the *start state*, and $\mathcal{S}^G \subseteq S$ is the set of *goal states*. A *plan* for a state s is a path from s to some $s_G \in \mathcal{S}^G$. A state s is *reachable* if there exists a path from s^I to s. A state is solvable if there exists a plan from s, otherwise we say that s is a dead end. By $h^*(s)$ ($g^*(s)$) we denote the length of a shortest plan for s (path from s^I to s). A plan for s is *optimal* iff its cost equals $h^*(s)$. Since our goal is to find solutions fast, regardless of their cost, we assume unit-cost domains. We also simplify the explanation of previous work on dominance for optimal planning based on this assumption.

Following previous work on dominance pruning, we consider a planning task as a set of LTSs on a common set of labels, $\{\Theta_1, \ldots, \Theta_n\}$. Given a planning task in the more common SAS+ formalism (Bäckström and Nebel 1995), the atomic transition systems representation with one LTS for each SAS+ variable can be easily obtained (Helmert *et al.* 2014). The state space of the task is the synchronized prod-



(A) Robot position. (B) Cell i visited. (C) Fuel available.

Figure 1: Example based on the Visitall domain where a robot must visit all tiles in a square grid. The robot has two units of fuel which are consumed when moving into striped cells so the robot must not enter the square grid via the shortest path.

uct of all the LTSs: $\Theta = \Theta_1 \otimes \cdots \otimes \Theta_n$. The synchronized product of two LTSs $\Theta_1 \otimes \Theta_2$ is another LTS with states $S = \{(s_1, s_2) \mid s_1 \in \Theta_1 \wedge s_2 \in \Theta_2\}$, transitions $T = \{((s_1, s_2), l, (s_1' s_2')) \mid (s_1, l, s_1') \in T_1 \wedge (s_2, l, s_2') \in T_2\}$, s.t. $(s_1, s_2) \in \mathcal{S}^G$ iff $s_1 \in \mathcal{S}_1^G$ and $s_2 \in \mathcal{S}_2^G$. We write $s \xrightarrow{l} s'$ as a shorthand for $(s, l, s') \in \Theta$. Let τ be a set of labels, we write $s \xrightarrow{\tau}{}^* s'$ to denote a path from s to s' where all labels belong to τ. We use subscripts to differentiate states in the state space Θ (e.g., s, s', t) and their projection into some Θ_i (e.g., s_i, s_i', t_i). We say that a transition $s \to s'$ in Θ affects Θ_i if it modifies its value, $s_i \neq s_i'$.

Consider a planning task represented as a set of LTSs $\{\Theta_1, \ldots, \Theta_n\}$ like our running example shown in Figure 1, where a robot must visit all tiles in a square grid. There is an LTS representing the position of the robot (A), an LTS for each cell in the square grid that represents if the cell has been visited or not (B), and an LTS describing the available fuel (C). In (A) we depict the grid. The corresponding LTS has a node for each cell, and transitions between adjacent cells. Transitions moving the robot to cell i are labeled with label m_i. Only walking into stripped cells (x) consumes fuel. All other labels have a self-loop transition in every state and they are omitted.

A heuristic is a function $h : S \to \mathbb{N}$ that estimates the distance from every state to the goal. A state is *reachable* if there exists a sequence of actions from s^I to it. A state is *alive* iff it is solvable, reachable, and not a goal state. A heuristic h is *descending* if all alive states have a successor with lower heuristic value. A heuristic is dead-end aware if $h(s) = \infty$ for all dead-end states s. Most common search algorithms in satisficing planning (e.g., hill-climbing or GBFS) will solve the planning task with at most $h(s^I)$ expansions if h is a descending and dead-end aware heuristic (Seipp *et al.* 2016).[1]

A relation $\preceq$ is a set of pairs of states. A relation $\preceq$ is a preorder iff it is reflexive and transitive. We write $s \prec t$ as a shorthand for $s \preceq t$ and $t \not\preceq s$ (i.e., $\prec$ is a strict partial-order). We say that $\preceq$ approximates a heuristic h iff $s \preceq t$ implies $h(t) \leq h(s)$. Dominance relations approximate the goal distance; whenever $s \preceq t$ (t dominates s) then t must be at least as close to the goal as s ($h^*(t) \leq h^*(s)$). Torralba and Hoffmann (2015) introduced label-dominance simulation, a method to compute a set of relations $\{\preceq_1, \ldots, \preceq_n\}$ that can be combined to derive a dominance relation $\preceq$ for Θ where $s \preceq t$ iff $s_i \preceq_i t_i$ for all i. In a best-first search with open list *open* and closed list *closed*, dominance pruning consists of removing a state s from the open list without expanding it, whenever there exists $t \in \mathit{open} \cup \mathit{closed}$ such that $s \preceq t$ and $g(t) \leq g(s)$. If $\preceq$ is a dominance relation, then at least one optimal plan is preserved.

Quantitative dominance extends the previous method by considering numeric functions instead of relations (Torralba 2017). A function $\mathcal{D} : S \times S \to \mathbb{Q} \cup \{-\infty\}$ is a quantitative dominance function (QDF) if $\mathcal{D}(s, t) \leq h^*(s) - h^*(t)$. QDFs are computed as a set of functions $\{\mathcal{D}_1, \ldots, \mathcal{D}_n\}$ such that $\mathcal{D}(s, t) = \sum_i \mathcal{D}_i(s_i, t_i)$. To guarantee that the sum of all $\mathcal{D}_i$ is a QDF, they must fulfill the equation:

$$\mathcal{D}_i(s_i, t_i) = \min_{s_i \xrightarrow{l} s_i'} \max_{u_i \xrightarrow{l'} u_i'} \mathcal{D}_i(s_i', u_i') - h^\tau(t_i, u_i) + \sum_{j \neq i} \mathcal{D}_j^L(l, l')$$

In words, whatever we can do from s_i ($s_i \xrightarrow{l} s_i'$), we can do from t_i via a path $t_i \xrightarrow{\tau}{}^* u_i \xrightarrow{l'} u_i'$, taking into account the comparison of the goal distance between the final result of both paths ($\mathcal{D}_i(s_i', u_i')$), the cost from t_i to u_i ($h^\tau(t_i, u_i)$), and how much cost we incur for applying l' instead of l in all other LTSs ($\sum_{j \neq i} \mathcal{D}_j^L(l, l')$). This requires to define h^τ and $\mathcal{D}_j^L$:

- h^τ accounts for transitions that only affect a single LTS Θ_i. A label is a τ-label for Θ_i iff it can always be applied to change the value of Θ_i without affecting other LTSs. Formally, if l is a τ-label for Θ_i then $s_j \xrightarrow{l} s_j \; \forall \Theta_j \neq \Theta_i, \forall s_j \in \Theta_j$. The τ-distance from s_i to t_i, written $h^\tau(s_i, t_i)$, is the length of a shortest path from s_i to t_i in Θ_i using only transitions with τ labels or ∞ if no such path exists. For example, moving the robot to a non-striped cell outside the square part of the grid is a τ-label because it changes the position of the robot without affecting other variables.
- $\mathcal{D}_j^L(l, l')$ measures how good it is to apply l' instead of l in Θ_j. If $\mathcal{D}_j^L(l, l') \geq 0$, it means that any time we can apply l to reach some s_j, we can also apply l' to reach t_j s.t. $\mathcal{D}(s_j, t_j) \geq 0$. For example, in the LTS that represents the available fuel, $\mathcal{D}_F^L(m_x, m_i) = 0$ for any striped cell x and non-striped cell i.

In the example of Figure 1, we can obtain a QDF $\{\mathcal{D}_1, \ldots, \mathcal{D}_n\}$ where each $\mathcal{D}_i$ is comparing states only according to their value in Θ_i. For the position of the robot we obtain $\mathcal{D}(x, y) = -d(x, y)$ where $d(x, y)$ is the distance from cell x to cell y using only movements that do not consume fuel. For the fuel, we obtain $\mathcal{D}(s, t) = 0$ if t has at least as much fuel than s or $-\infty$ otherwise. For each cell, we have a value of $-\infty$ if the cell has been visited in s and not in t and 0 otherwise. In optimal planning, t dominates s if $\sum_i \mathcal{D}_i(s_i, t_i) \geq 0$. In our example this means that a state is better if it has visited more cells, it has at least as much fuel and the position of the robot is the same.

QDFs can be used, apart for dominance pruning, to perform action selection. Action selection is a type of pruning

[1] Seipp *et al.* (2016) consider the more general case of dead-end avoiding heuristics instead of dead-end aware heuristics.

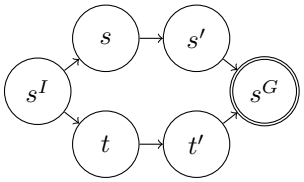


Figure 2: Example with two alternative paths to the goal.

where a state $s \in open$ may be replaced by one of its immediate successors t if $\mathcal{D}(s,t) \geq c(s,t)$ where $c(s,t)$ is the cost of the transition from s to t. In that case, such transition starts an optimal plan from s, so at least one optimal solution is always preserved.

Satisficing Dominance

In optimal planning, a dominance relation is one where for any $s \preceq t$, t should be as close to the goal as s. However, this is sometimes too restrictive for satisficing search. For example consider a problem where there are two paths to the goal, one requires solving a hard combinatorial problem and the other follows a straightforward, but potentially longer, path. Assuming that providing any guarantees about the cost of solving the combinatorial problem is hard, no dominance can be proven for optimal planning. However, it is simple to manually design a dominance relation where the states in the simpler path dominate those related to solving the combinatorial problem, directly guiding the search towards the goal. With this aim, we define a satisficing dominance relation as one that preserves solutions, no matter their cost or length.

Definition 1 (Satisficing Dominance Relation) *A preorder $\preceq$ is a satisficing dominance relation if there exists a descending and dead-end aware heuristic $h^\preceq$ such that $\preceq$ approximates $h^\preceq$ ($s \preceq t \implies h^\preceq(t) \leq h^\preceq(s)$).*

Intuitively, $h^\preceq$ should be dead-end aware so that unsolvable states do not dominate solvable states, and descending to avoid the case where a state dominates all its successors, hence rendering the search incomplete. Note that simply requiring each state to not dominate one of its solvable successors is not enough to guarantee that a plan is preserved. Consider the example of Figure 2, where dominance pruning with a relation where $t' \preceq s$ and $s' \preceq t$ could prune both s' and t', causing all solutions to be pruned.

This is a generalization of dominance relations used in optimal planning, since the perfect heuristic h^* is descending and dead-end aware. Note that any descending and dead-end aware heuristic can be defined via computing h^* after changing the cost of the transitions in Θ. Therefore, Definition 1 can also be interpreted as a dominance relation for an instance with a different cost function.

In optimal planning, dominance relations have been used in two different ways: for dominance pruning (eliminating states that are dominated by others) and action selection pruning (automatically applying an action if this action is guaranteed to start an optimal plan). Next, we adapt these types of pruning to satisficing planning. Dominance pruning can be applied in a similar way as in optimal planning, but slightly stronger since the cost of reaching each state does not matter.

Theorem 1 *Let $\preceq$ be a satisficing dominance relation. Then, a best-first search with open list $open$, and closed list $closed$ in which a state $s \in open$ may be pruned if there exists another $t \in open \cup closed$ such that $s \preceq t$ is complete.*

Proof Sketch: Let $h^\preceq$ be the dead-end aware and descending heuristic approximated by $\preceq$. s was pruned so there must exist $t \in open \cup closed$ such that $h^\preceq(t) \leq h^\preceq(s)$. Let u be the state with lowest $h^\preceq$ value in the open list. Then, $h^\preceq(u) \leq h^\preceq(t)$ since if t is closed, one of its successors with lower $h^\preceq$ value ($h^\preceq$ is descending) was inserted in $open$. Since $h^\preceq(u) \leq h^\preceq(s) < \infty$, u is solvable. As $h^\preceq$ is descending, there exists a plan for u that does not contain any state dominated by s. □

Action selection can also be adapted for the satisficing case. In this case, we do not care about the solution cost so quantitative dominance is not required anymore. Instead, we consider strict dominance to avoid loops in which two states that dominate each other are constantly replaced by one another. We can also generalize action selection to consider not only immediate successors, but also any successor that is reached by a sequence of actions. This is far more useful in satisficing than in optimal planning because the cost of the action sequence can be ignored.

Theorem 2 *Let $\prec$ be a strict satisficing dominance relation. A best-first search where a state $s \in open$ can be replaced by some t such that t is the result of executing any sequence of actions in s and $s \prec t$ is complete.*

Proof Sketch: As $h^\prec(t) \leq h^\prec(s)$, and $h^\prec$ is descending, t must have a solution that does not traverse s, since all states t_i in the solution have $h^\prec(t_i) < h^\prec(s)$ so $t_i \neq s$ and $t_i \not\prec s$. By transitivity neither t nor any state in its solution can be substituted by s or any state s' such that $s' \prec s$. □

It should be noted that both types of pruning can be applied at the same time, but only if they use the same relation.

Theorem 3 *Let $\preceq$ be a satisficing dominance relation. Then:*

1. *Let $\prec$ be a strict relation such that $s \prec t$ iff $s \preceq t$ and $t \not\preceq s$. Performing dominance pruning with $\preceq$ and action selection with $\prec$ is always safe.*
2. *Let $\prec'$ be a different strict satisficing dominance relation. Then, there exist cases where performing dominance pruning with $\preceq$ and action selection with $\prec'$ is not safe.*

Proof Sketch: To show (2.) consider again the example of Figure 2. Let $\preceq$ be a relation such that $s \preceq s' \preceq s^I \preceq t \preceq t' \preceq s^G$ and $\prec'$ be a strict relation such that $t \prec' t' \prec' s^I \prec' s \prec' s' \prec' s^G$. Then, by Theorem 2, s^I can be replaced by t, and then later t' could be pruned according to Theorem 1 because it is dominated by a previously expanded state (s^I).

To show (1.), if both relations $\preceq$ and $\prec$ approximate the same heuristic $h^\preceq$, then the minimum $h^\preceq$ value of any state in the open list monotonically decreases. A loop like the one in the example above cannot happen because the value

of $h^\prec$ can only decrease along the path to the goal. Since $h^\prec(t') < h^\prec(t)$, t' cannot be pruned by any state that has a larger $h^\prec$ value (e.g., any state that is replaced by t). □

Dominance-Based Enforced Hill-Climbing

Enforced Hill-Climbing (EHC) is a well-known search algorithm for planning (Hoffmann and Nebel 2001). EHC performs a breadth first search from the initial state s^I until finding a state s such that $h(s) < h(s^I)$. At that point, if s is a goal state a plan has been found. Otherwise, the initial state is replaced by s and the search is restarted.

According to Theorem 2, any time we find a state strictly better than s according to a satisficing dominance relation $\preceq$, we can remove s and all other of its successors from consideration. This may be expensive to do for all states in the search, but can be easily done for the special case where s is the initial state s^I. In that case, the search is restarted from the newly found state that dominates s^I. This is a form of EHC, where the search is restarted whenever a state better than s^I is found, substituting the heuristic function by a dominance relation to determine which states are better than s^I.

Also, while the original EHC algorithm used breadth-first search to escape the current plateau, there is no reason to not consider other best-first search algorithms with different priority functions as well. We define the dominance-based EHC DEHC$_\prec(X)$ algorithm relative to any best-first search algorithm X and strict preorder $\prec$. DEHC$_\prec(X)$ runs algorithm X until finding a goal state or any state s such that $s^I \prec s$. In the latter case, it restarts from s.

Theorem 4 *Let X be a sound and complete best-first search algorithm, and let $\prec$ be a strict preorder such that for any pair of reachable states s, t, if $s \prec t$ and s is solvable then t is solvable. Then DEHC$_\prec(X)$ is sound and complete.*

Proof: Soundness follows from soundness of X. Completeness: If the instance is solvable, each run of X can finish either finding a goal, or finding a state t such that $s^I \prec t$ and another instance of X is started from t. Then, as t must be solvable, X can never be restarted on an unsolvable state. The algorithm always terminates because $\prec$ is a strict preorder so the number of times X may be called is bounded by the number of reachable states which is always finite. □

The conditions required for $\prec$ are weaker than what is required for a satisficing dominance relation. If $\prec$ is based on a satisficing dominance relation $\preceq$, then by Theorem 3, dominance pruning can be used in X. In this case, any state dominated by the initial state in each call of X can be pruned, thereby ensuring that the search does not re-expand any previous initial state. However, DEHC can also be used with relations defined from heuristic functions in the following way.

Definition 2 (Heuristic-based Relation for DEHC) *Let $\mathcal{D}$ be a quantitative dominance function and h be any heuristic. We define $\prec^h$ as a relation such that $s \prec^h t$ if and only if $\mathcal{D}(s,t) > -\infty$ and $h(t) < h(s)$.*

As $\mathcal{D}(s,t) > -\infty$ implies that $h^*(s) - h^*(t) > -\infty$ this means that if s is solvable, t must be solvable as well. This results in a complete variant of EHC with any heuristic function that uses dominance only to avoid dead-ends. The role of $\prec$ in this context is to select when to be more or less greedy following the heuristic advice, interpolating between GBFS (when no dominance is found) and EHC.

Practical Methods for Computing Satisficing Dominance Relations

In this section, we introduce a new method to compute dominance relations for satisficing planning.

Serialized Dominance Relations

Consider the example of Figure 1. Dominance relations for optimal planning will consider a state better if more cells have been visited, the robot has at least as much fuel and the position of the robot is the same. The latter condition is an important limitation because, in order to find a state that dominates the initial state, the robot must go back to the initial position every time that it visits more cells. This is undesirable since it will be harder to find a state that dominates s^I and it will result in longer plans.

Intuitively, we prefer states where more cells have been visited, regardless of the position of the robot. This is possible because these sub-goals are serializable, i.e., no sub-goal must be undone in order to achieve the rest. To obtain such relation, we serialize the LTSs so that a state dominates another if it is as good for the first $j-1$ LTSs (has not unvisited any position), it is strictly better in Θ_j (has visited a new position), and there exists a solution for all other LTSs without using any label that is "dangerous" for the previous LTSs. We define a label as dangerous for an LTS Θ_i according to $\preceq_i$ if applying it on some state s_i requires to go to a potentially worse state s_i' s.t. $s_i \not\preceq_i s_i'$.

Definition 3 (Dangerous label) *Let $\preceq_i$ be a relation for Θ_i. We say that a label l is dangerous for $\preceq_i$ if there exists a state $s_i \in \Theta_i$ such that there exists $s_i \xrightarrow{l} s_i'$ and there does not exist $s_i \xrightarrow{l} t_i$ s.t. $s_i \preceq_i t_i$.*

For example, labels associated with movements that consume fuel are dangerous for the LTS that represents the amount of available fuel. However, movements of the robot are not dangerous for the LTSs that correspond to whether a cell has been visited or not. Now, we can serialize the LTSs that define our task. The serialized dominance gives preference to those states that are better according to the first LTS, as long as a solution can be found for the other LTSs without using any label that is dangerous for the first LTS (i.e., the sub-goals achieved do not need to be undone). To model this, we re-define label dominance (i.e., the component $\mathcal{D}_j^L(l,l')$ used in the equation that defines a QDF) so that dangerous labels do not dominate any label.

Definition 4 (Serialized Dominance) *Let $\langle \preceq_1, \ldots, \preceq_n \rangle$ be a label-dominance simulation for a list of LTSs $\langle \Theta_1, \ldots, \Theta_n \rangle$ and $\langle \mathcal{D}_1, \ldots, \mathcal{D}_n \rangle$ a list of functions that satisfy the equations of a QDF where $\mathcal{D}_j^L(l,l') = -\infty$ for all $l' \in L$ and $l \in L$ that is dangerous for $\preceq_i$ for any $i < j$. We define the serialized dominance relation as $s \preceq^S t$ iff*

$s_j \preceq_j t_j$ *for all* $j \in [1, n]$ *or exists* i *such that* $s_j \preceq_j t_j$ *for all* $j \in [1, i)$, $s_i \prec_i t_i$ *and* $\mathcal{D}(s_j, t_j) > -\infty$ *for all* $j \in (i, n]$.

Theorem 5 *A serialized dominance* $\preceq^S$ *is a satisficing dominance relation.*

Proof Sketch: We show that $\preceq^S$ approximates a descending and dead-end aware heuristic function $h^\preceq$ ($s \preceq^S t \implies h^\preceq(t) \leq h^\preceq(s)$). As $h^\preceq$ is dead-end aware, if s is a dead-end then $h^\preceq(s) = \infty$ and the condition holds. If s is not a dead-end then t cannot be a dead end because $s \preceq^S t$ implies $\sum \mathcal{D}_i(s_i, t_i) > -\infty$. Therefore, it suffices to consider the case where s and t are both solvable.

We define $h^\preceq$ as the perfect goal distance under a cost function constructed from $\preceq$ such that (i) costs of all transitions affecting Θ_i cost more than those that only affect Θ_j for $i < j$ and (ii) if $s_i \prec_i t_i$ transitions from s_i cost more than transitions from t_i. In both cases, the cost difference must be large enough so that the most expensive transition dominates the cost of the entire path.

To prove that $s \preceq^S t \implies h^\preceq(t) \leq h^\preceq(s)$, we assume WLOG that $s_1 \prec_1 t_1$.[2] Then, for any path from s_1, $\pi^s = s_1 \xrightarrow{l_1} s_1^1 \dots \xrightarrow{l_k} s_1^k$, there exists a path from t_1, $\pi^t = t_1 \xrightarrow{l'_1} t_1^1 \dots \xrightarrow{l'_k} t_1^k$ such that $s_1^i \preceq t_1^i$ for all $i \in [1, k]$. Then, the cost of π^t is lower than that of π^s because the first transition is more expensive from s_1 ($s_1 \prec_1 t_1$) and the rest are not.

Since $\mathcal{D}_i(s_i, t_i) > -\infty$ for all $i \in [2, n]$, by the properties of a QDF, the path π^t can always be extended into a plan for t by inserting additional actions. As $\mathcal{D}_i^L(l, l') = -\infty$ these additional actions are not dangerous for $\preceq_1$. Since in our cost function the cost of the most expensive transition dominates the overall cost, the complete path for t is still cheaper than the one for s under this cost function so $h^\preceq(t) < h^\preceq(s)$. □

The resulting dominance relation is heavily influenced by the ordering chosen for the LTSs. To preserve completeness, this order must be the same throughout the entire search. However, one does not need to decide the order a priori, but rather it can be dynamically chosen during the search. Initially, we keep a set with all $\{\Theta_1, \dots, \Theta_n\}$ unsorted and the list of serialized LTSs is initialized empty. When comparing a state s against s^I, we check whether $s_i \prec_i s_i^I$ for some i and insert Θ_i in the list of serialized LTSs if and only if thanks to this we get that $s^I \prec^S s$. Using this policy in our running example, the order in which the cells are serialized in the dominance relation is exactly the order in which they are found during the search.

Recursive and Positive τ-Labels

The method above is most interesting in situations where dominance relations in optimal planning cannot prove t to be closer to the goal than s, but where it can show that t is not a dead-end, i.e., $-\infty < \mathcal{D}(s, t) < 0$. For this, τ-labels are of great importance. Having more τ-labels can only decrease the tau distance between states (h^τ), which may in turn increase the value of $\mathcal{D}$. Previous work considered l a τ-label for Θ_i if it has self-loop transitions for any state in all other Θ_j. In other words, transitions labeled with l may be used to modify the value of Θ_i in any state without affecting the value of other LTSs. Hence, all τ-labels had to fulfill two properties:

1. They do not have preconditions on other LTSs so it is always applicable, and
2. They do not have side effects on other LTSs.

Here, we extend the notion of τ-labels in two different ways, relaxing each of these assumptions in order to find coarser dominance relations.

Recursive τ-labels Some labels are not τ-labels because they have preconditions on other LTSs. For example, in a typical logistics transportation task, loading a package at some location is not a τ-label for the position of the package because it is not applicable in all states (the truck must be at the same location). However, the truck can always be driven from any given location to the position of the package, load it, and then drive back to the original position, reaching a state where the package is in the truck without affecting any other variable. Hence, we could introduce new transitions that correspond to those macro-actions. We use this in order to redefine the set of τ-labels for each LTS.

For every s_i such that there exists Θ_j with a path $\pi_l^\tau(s_i, s_j) = (s_i, s_j) \xrightarrow{\tau}{}^*(s_i, s'_j) \xrightarrow{l} (s'_i, s'_j) \xrightarrow{\tau}{}^*(s'_i, s_j)$ for all $s_j \in \Theta_j$, we may introduce a new transition $s_j \xrightarrow{l} s_j$. The cost of this new transition is equal to the cost of the τ actions in $\pi_l^\tau(s_i, s_j)$. Thanks to these self-loops, l may become a τ label. In that case, we do not introduce these transitions to the definition of the planning task. Instead, we simply consider label l to be a τ label with a cost equal to the maximum $\pi_l^\tau(s_i, s_j)$ for any (s_i, s_j). After introducing new τ-labels, the process can be repeated.

Positive τ-labels Some labels are not τ-labels because they have side effects. In our running example movements are not τ-labels for the LTSs representing the position of the robot because they have the side-effect of marking a cell as visited. However, these side-effects are always positive according to our dominance relation so they can be ignored for the computation of τ labels. A label is a positive τ-label for Θ_i iff $\forall \Theta_j \neq \Theta_i, \forall s_j \in \Theta_j$, there exists $s_j \xrightarrow{l} t_j \in \Theta_j$ s.t. $\mathcal{D}_j(s_j, t_j) \geq 0$.

When using this definition one must be careful, due to the circular dependency between the values of $\mathcal{D}$ and the set of τ-labels. $\mathcal{D}$ is typically computed by assuming a very coarse dominance relation and then iteratively refining it until a fixpoint is reached. However, the values of $\mathcal{D}$ during this computation have not been proven correct until it ends, so positive τ-labels cannot be defined in terms of the $\mathcal{D}$ that is being computed. Hence, to avoid such circular dependencies we first compute $\mathcal{D}$ based on the previous notion of τ-labels,

[2] Let j be the smallest index for which $s_j \prec_j t_j$. If $j > 1$, we can consider instead the synchronized product $\Theta_1 \otimes \dots \otimes \Theta_j$. By the properties of label-dominance simulation, $(s_1, \dots, s_j) \prec_{1,\dots,j} (t_1, \dots, t_j)$.

and then compute a new set of τ-labels and re-compute $\mathcal{D}$. This process can be repeated until no more labels are added to the set of τ labels.

Experiments

We run experiments on all satisficing-track STRIPS planning instances from the international planning competitions (IPC'98 – IPC'14). All experiments were conducted on a cluster of Intel Xeon E5-2650v3 machines with time (memory) cut-offs of 30 minutes (4 GB). Our goal is to evaluate the potential of current dominance techniques to enhance search in satisficing planning. As a simple baseline, we use lazy GBFS in Fast Downward (Helmert 2006) with the h^{FF} heuristic (Hoffmann and Nebel 2001), and compare the results against dominance-based EHC guided with blind search (h^B) and the h^{FF} heuristic. We also include the performance of LAMA (Richter and Westphal 2010) and Mercury (Katz and Hoffmann 2014; Domshlak *et al.* 2015) as representatives of more modern planners.

We run several configurations comparing our new serialized dominance ($\preceq^S$) against quantitative dominance relations ($\preceq^{\mathcal{D}}$) used in previous work on optimal planning (Torralba 2017). However, satisficing planning benchmarks are much larger than those for optimal planning so we change the dominance pruning setting in two important aspects. On the one hand, previous work considered using merge and shrink (Helmert *et al.* 2014) to reduce the number of LTSs. But, as the overhead is too large for these benchmarks, we consider instead only the atomic transition systems. This reduces the number of domains in which state-of-the-art methods are effective to find dominance. On the other hand, previous work considered pruning states that are dominated by any previously expanded state. This check is too expensive in satisficing planning so instead we only compare each state against its parent and the initial state.

Table 1 shows coverage results on domains where dominance has a non-negligible effect either by restarting the search from states that dominate the initial state (top part of the table) or by pruning states (bottom part). The results show that our dominance techniques are able to find useful dominance relations in a number of IPC domains, even when only considering atomic transition systems. Compared to the baseline, the results are quite good in most domains. Results could be even better but our implementation is not able to finish the preprocessing in the largest tasks of some domains (e.g. Logistics, Rovers, Satellite, and Visitall). This explains why not all instances of Visitall are solved by DEHC(h^B) and the difference wrt. the baseline in domains where no dominance pruning occurs.

The general trend is that serialized dominance relations ($\preceq^S$) are most useful to compare states against the initial state in the context of DEHC (domains in the upper part of the table), while the dominance relation based on the distance to the goal $\preceq^{\mathcal{D}}$ is more effective for dominance pruning and action selection. The reason is that the serialization is global for the entire search, slightly reducing the ability of dominance pruning and action selection.

Focusing on domains where dominance is useful for DEHC, one can observe that there is no much synergy with heuristics and they can even be harmful, like in Scanalyzer. The reason is that the heuristic is not aware of the dominance relation so it may guide the search in a direction where no states dominating the initial state can be found.

	Baseline			Pruning					
Domain	GBFS (h^{FF})	L	M	DEHC(h^B) $\preceq^{\mathcal{D}}$	DEHC(h^B) $\preceq^S$	DEHC(h^{FF}) $\preceq^{\mathcal{D}}$	DEHC(h^{FF}) $\preceq^S$	GBFS(h^{FF}) $\preceq^{\mathcal{D}}$	GBFS(h^{FF}) $\preceq^S$
Logistics (63)	54	**63**	**63**	43	43	57	**58**	53	53
Miconic (150)	**150**	**150**	**150**	**150**	**150**	**150**	**150**	**150**	**150**
Openstacks (30)	**30**	**30**	**30**	7	7	**30**	**30**	**30**	**30**
Rovers (40)	23	**40**	**40**	14	12	22	**24**	23	23
Satellite (36)	**30**	**36**	**36**	10	10	21	25	22	26
Scanalyzer (50)	44	**50**	**50**	**46**	**46**	44	44	44	44
Visitall (40)	5	**40**	**40**	3	**31**	6	**31**	4	4
Woodwork (50)	**49**	**50**	**50**	6	6	**49**	**49**	**49**	**49**
Zenotravel (20)	**20**	**20**	**20**	13	13	**20**	**20**	**20**	**20**
$\sum$	405	**479**	**479**	292	318	399	**431**	395	399
Floortile (40)	8	8	8	8	8	**14**	**14**	**14**	**14**
Maintenance (20)	5	2	**7**	0	0	**6**	**6**	**6**	**6**
Nomystery (20)	9	13	15	**20**	11	**20**	17	**20**	16
Parking (40)	**29**	**40**	**40**	0	0	26	28	27	28
Pathways (30)	11	24	**30**	4	4	**12**	**12**	**12**	**12**
Pipes-NT (50)	**30**	43	**44**	15	14	29	29	29	29
Tidybot (20)	**14**	**16**	14	1	1	7	13	7	12
TPP (30)	22	**30**	**30**	6	6	**23**	22	21	**23**
$\sum$	128	176	**188**	54	44	137	**141**	136	140
Total (1636) $\sum$	1231	1462	**1491**	624	640	1217	**1252**	1212	1219

Table 1: Coverage on IPC instances. We highlight the best configuration apart from LAMA (L) and Mercury (M). At the top are domains containing an instance where DEHC restarts from a state that dominates the initial state at least 10 times in a single instance. At the bottom, domains where dominance pruning has a non-negligible effect on coverage.

Even though the results of DEHC are quite good on these domains compared to the baseline, they are still far behind LAMA and Mercury, which easily solve all instances in those domains. LAMA uses landmarks in order to achieve sub-goals very greedily so is extremely effective in domains where all sub-goals are serializable. Therefore, one may wonder whether there are cases where DEHC can beat LAMA. One example is Nomystery, where action selection pruning is specially effective. But our running example illustrates the strengths of dominance even better.

Figure 3 shows the comparison of DEHC$_{\preceq^S}$ against LAMA in our running example. The particular feature of our running example is that it combines sub-goals that are easily serializable (visiting normal cells), with others that are not (visiting stripped cells). Heuristic approaches that greedily try to maximize the number of achieved sub-goals fall into a dead-end trap and are unable to solve the task. However, $\preceq^S$ is able to identify which sub-goals are safe and which ones could potentially be dangerous. As the heuristics are not aware of the dominance relation, this only works in combination with blind search. Otherwise the search is guided by the heuristic towards a part of the state space where no dominance can be found (correctly so, because it is a dead-

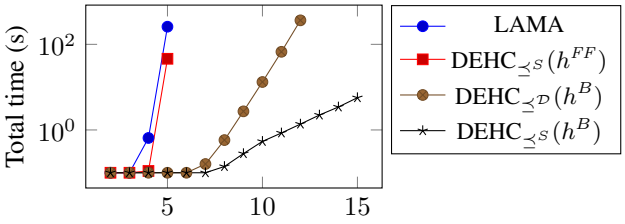


Figure 3: Total time of DEHC and LAMA in our running example.

end trap). DEHC$_{\preceq^D}(h^B)$ also beats LAMA in this domain, but it is still worse than DEHC$_{\preceq^S}(h^B)$ because using dominance purely based on goal distance the robot needs to go back to the initial state every time it visits a new cell, which is hard to do without any heuristic guidance towards there.

Related Work

The notion of dominance is related to approaches that characterize when a task can be solved in polynomial time. Our serialized dominance relation reminds of serializable sub-goals (Korf 1987; Barrett and Weld 1993). A set of sub-goals is serializable if they can always be achieved sequentially without undoing any of them. Our dominance relation also imposes a serialization on the LTSs that form the planning task. This is slightly different from sub-goals in that we may obtain dominance if progress has been made in an LTS (e.g. a package being in the truck is better than at the initial position) while sub-goals only consider its goal value (the package being at its destination). If a problem has several serializable sub-goals, we can always construct a dominance relation that represents this information. Up to the best of our knowledge, there are no automatic algorithms to prove that a set of sub-goals is serializable. Serialized dominance could be tailored for this purpose.

There is a long list of works that identify tractable fragments of the optimal and satisficing planning problem (Bäckström and Klein 1991; Jonsson and Bäckström 1998; Brafman and Domshlak 2003; Giménez and Jonsson 2008; Katz and Domshlak 2008; Chen and Giménez 2010). Our dominance techniques can capture some of the structure exploited by these tractable fragments, like acyclic causal graphs (using recursive τ-labels). But, at the same time, we are not limited by such features of the planning tasks (e.g. the causal graph of our running example is not acyclic). Moreover, dominance relations can be useful in tasks where planning is intractable but some part of the problem is easy to solve. In those cases, the use of dominance techniques can still dramatically reduce the search space.

There are several parametrized search algorithms that run in polynomial time in a width parameter w. These algorithms are based on a substantial amount of pruning either by only allowing to change the value of up to w variables (Chen and Giménez 2007) or pruning states with a novelty greater than w (Lipovetzky and Geffner 2012). In both cases, these algorithms do not solve the entire planning task, but are used to find a state "better" than the initial state in terms of the achieved sub-goals. Dominance relations offer an alternative way to compare states, which is more general than the criterion used by Chen and Giménez and offers completeness guarantees unlike simple sub-goal based criteria suggesting potential in combining these approaches.

Seipp *et al.* (2016) introduced another notion of width based on how hard is to represent a dead-end aware and descending potential heuristic (Pommerening *et al.* 2015). Many typical domains have a width of 2, meaning that it is easy to represent a heuristic that solves them in polynomial time. No method to automatically find such heuristic is known yet but, satisficing dominance relations approximate these kind of heuristics so any such algorithm could potentially be used to obtain dominance relations as well.

A question that naturally comes up is what is the advantage of using dominance relations over heuristic functions. Dominance relations are more expressive than heuristics because they are partial preorders while heuristics are total preorders. For example, we may have relations where $s \preceq t$ and $s' \preceq t'$, but the relation between s, t and s', t' remains unknown (e.g. $s \not\preceq t'$ and $s' \not\preceq t$). However, no assignment of heuristic values can represent this relation. In practice, this matters most in cases where dominance is able to discover some local information that can be exploited independently of the rest of the problem. Consider the Nomystery domain, where a truck transports a set of packages using a limited amount of fuel. The use of dominance allows us to identify that having a package at its destination is always good so we can unload it directly without considering any other alternative. The problem with heuristics is that they aggregate all estimations into a single value, making it very difficult to identify in which parts of the state space the heuristic is wrong. In Nomystery, most heuristics will correctly estimate that all packages need to be loaded and unloaded exactly once, but underestimate the number of truck movements. However, the search will equally explore the possibility of loading/unloading packages in different locations due to the heuristic inaccuracies.

Our Dominance-Based Enforced Hill-Climbing algorithm uses quantitative dominance functions to guarantee completeness by ensuring that the search is never restarted from a dead end. Recently, there have been other approaches based on heuristic refinement that also devise a variant of EHC that preserves completeness (Fickert and Hoffmann 2017).

Discussion

In this paper, we have introduced the notion of dominance for satisficing planning. Dominance can be used for dominance pruning as well as to identify states that are strictly better than others. This allows the search algorithm to be extremely greedy, restarting the search from any state that dominates the initial state, but still preserving completeness. We have adapted the algorithms to automatically find dominance relations for these purposes. Dominance can be a very powerful instrument to compare states, specially in instances with a mixture of complex and simple sub-goals.

Our experiments show the ability of dominance to guide EHC search. However, there is still a gap when compared

against state-of-the-art planners. Our results also point out some limitations of current dominance techniques that may be worth exploring in future work. Considering more than one variable at a time could help to find stronger dominance relations in many domains. Also, heuristics could use the information captured by the dominance relation, increasing the synergy between them.

Acknowledgments

This paper has been supported by the German Research Foundation (DFG), under grant HO 2169/6-1, "Star-Topology Decoupled State Space Search".

References

[Bäckström and Klein 1991] Christer Bäckström and Inger Klein. Planning in polynomial time: The SAS-PUBS class. *Computational Intelligence*, 7(4), 1991.

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

[Barrett and Weld 1993] Anthony Barrett and Daniel S. Weld. Characterizing subgoal interactions for planning. pages 1388–1393, 1993.

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

[Chen and Giménez 2007] Hubie Chen and Omer Giménez. Act local, think global: Width notions for tractable planning. In Mark Boddy, Maria Fox, and Sylvie Thiebaux, editors, *Proc. of the 17th International Conference on Automated Planning and Scheduling (ICAPS'07)*, pages 73–80, 2007.

[Chen and Giménez 2010] Hubie Chen and Omer Giménez. Causal graphs and structurally restricted planning. *Journal of Computer and System Sciences*, 76(7):579–592, 2010.

[Chen *et al.* 2004] Y. Chen, C. Hsu, and B. Wah. SGPlan: Subgoal partitioning and resolution in planning. In Stefan Edelkamp, Jörg Hoffmann, Michael Littman, and Håkan Younes, editors, *Proc. of the 4th International Planning Competition*, 2004.

[Domshlak *et al.* 2015] Carmel Domshlak, Jörg Hoffmann, and Michael Katz. Red-black planning: A new systematic approach to partial delete relaxation. *Artificial Intelligence*, 221:73–114, 2015.

[Fickert and Hoffmann 2017] Maximilian Fickert and Jörg Hoffmann. Complete local search: Boosting hill-climbing through online heuristic-function refinement. In *Proc. of the 27th International Conference on Automated Planning and Scheduling (ICAPS'17)*, 2017.

[Giménez and Jonsson 2008] Omer Giménez and Anders Jonsson. The complexity of planning problems with simple causal graphs. *Journal of Artificial Intelligence Research*, 31:319–351, 2008.

[Hall *et al.* 2013] David Hall, Alon Cohen, David Burkett, and Dan Klein. Faster optimal planning with partial-order pruning. In Daniel Borrajo, Simone Fratini, Subbarao Kambhampati, and Angelo Oddi, editors, *Proc. of the 23rd International Conference on Automated Planning and Scheduling (ICAPS'13)*, 2013.

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

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

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

[Hoffmann *et al.* 2004] Jörg Hoffmann, Julie Porteous, and Laura Sebastia. Ordered landmarks in planning. *Journal of Artificial Intelligence Research*, 22:215–278, 2004.

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

[Jonsson and Bäckström 1998] Peter Jonsson and Christer Bäckström. State-variable planning under structural restrictions: Algorithms and complexity. *Artificial Intelligence*, 100(1–2):125–176, 1998.

[Katz and Domshlak 2008] Michael Katz and Carmel Domshlak. New islands of tractability of cost-optimal planning. *Journal of Artificial Intelligence Research*, 32:203–288, 2008.

[Katz and Hoffmann 2014] Michael Katz and Jörg Hoffmann. Mercury planner: Pushing the limits of partial delete relaxation. In *IPC 2014 planner abstracts*, pages 43–47, 2014.

[Korf 1987] Richard E. Korf. Planning as search: A quantitative approach. *Artificial Intelligence*, 33(1):65–88, 1987.

[Lipovetzky and Geffner 2012] Nir Lipovetzky and Hector Geffner. Width and serialization of classical planning problems. In Luc De Raedt, editor, *Proc. of the 20th European Conference on Artificial Intelligence (ECAI'12)*, pages 540–545, 2012.

[Pommerening *et al.* 2015] Florian Pommerening, Malte Helmert, Gabriele Röger, and Jendrik Seipp. From non-negative to general operator cost partitioning. In Blai Bonet and Sven Koenig, editors, *Proc. of the 29th AAAI Conference on Artificial Intelligence (AAAI'15)*, pages 3335–3341, 2015.

[Porteous *et al.* 2001] Julie Porteous, Laura Sebastia, and Jörg Hoffmann. On the extraction, ordering, and usage of landmarks in planning. In A. Cesta and D. Borrajo, editors, *Proc. of the 6th European Conference on Planning (ECP'01)*, pages 37–48, 2001.

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

[Richter *et al.* 2011] Silvia Richter, Matthias Westphal, and Malte Helmert. LAMA 2008 and 2011 (planner abstract). In *IPC 2011 planner abstracts*, pages 50–54, 2011.

[Röger and Helmert 2010] Gabriele Röger and Malte Helmert. The more, the merrier: Combining heuristic estimators for satisficing planning. In Ronen I. Brafman, Hector Geffner, Jörg Hoffmann, and Henry A. Kautz, editors, *Proc. of the 20th International Conference on Automated Planning and Scheduling (ICAPS'10)*, pages 246–249, 2010.

[Seipp *et al.* 2016] Jendrik Seipp, Florian Pommerening, Gabriele Röger, and Malte Helmert. Correlation complexity of classical planning domains. In Subbarao Kambhampati, editor, *Proc. of the 25th International Joint Conference on Artificial Intelligence (IJCAI'16)*, pages 3242–3250, 2016.

[Torralba and Hoffmann 2015] Álvaro Torralba and Jörg Hoffmann. Simulation-based admissible dominance pruning. In Qiang Yang, editor, *Proc. of the 24th International Joint Conference on Artificial Intelligence (IJCAI'15)*, pages 1689–1695, 2015.

[Torralba 2017] Álvaro Torralba. From qualitative to quantitative dominance pruning for optimal planning. In Carles Sierra, editor,

Proc. of the 26th International Joint Conference on Artificial Intelligence (IJCAI'17), pages 4426–4432, 2017.

Online Refinement of Cartesian Abstraction Heuristics

Rebecca Eifler and **Maximilian Fickert**
Saarland University
Saarland Informatics Campus
Saarbrücken, Germany
{eifler,fickert}@cs.uni-saarland.de

Abstract

In classical planning as heuristic search, the guiding heuristic function is typically treated as a black box. While many heuristics support *refinement* operations, they are typically only used for its initialization before search, but further refinement during search could make use of additional information not available in the initial state. We explore online refinement for additive Cartesian abstraction heuristics. These abstractions are computed through counter-example guided abstraction refinement, which can be applied online as well to further improve the abstractions. We introduce three operations, *refinement*, *merging*, and *reordering*, which are combined to a converging online-refinement algorithm. We describe how online refinement can effectively be used in A* and evaluate our approach on the IPC benchmarks, where it outperforms offline-generated abstractions in many domains.

Introduction

Heuristic search is one of the most successful approaches to classical planning. Many heuristics have a parameter to increase the level of precision which typically implies a trade-off with respect to the computational complexity when evaluating the heuristic. For *abstraction heuristics* (Edelkamp 2001; Helmert et al. 2014; Seipp and Helmert 2013), the size of the abstraction can be chosen to range from the null heuristic $h^0 = 0$ to the perfect heuristic h^*. *Partial delete relaxation heuristics* (Keyder, Hoffmann, and Haslum 2014; Domshlak, Hoffmann, and Katz 2015; Fickert, Hoffmann, and Steinmetz 2016) interpolate between fully relaxed semantics and real semantics.

These heuristics are often instantiated through iterative *refinement* operations. The heuristic starts out with a basic relaxation, and is repeatedly refined until a time or memory bound is reached. Given sufficiently large bounds the heuristic may *converge*, making the relaxation exact (e.g. (Haslum et al. 2007; Seipp and Helmert 2013; Helmert et al. 2014; Keyder, Hoffmann, and Haslum 2014). This process is traditionally done offline, i.e. once before search, and the resulting heuristic is treated as a black box throughout search.

However, as search progresses and new information becomes available, this additional knowledge might be used to further improve the heuristic, e.g. to eliminate flaws in the relaxation that were not apparent in the initial construction of the heuristic and were only detected later in the search process. Additional refinement steps performed online can address such issues and further improve the heuristic.

So far, online refinement of heuristic functions is mostly unexplored. Fickert and Hoffmann (2017) introduced online refinement for the h^{CFF} heuristic in an enforced hill-climbing setting. The heuristic is refined whenever search is stuck in a local minimum, thus effectively removing local minima from the search space surface instead of attempting to escape them through brute-force search.

There are several other forms of online learning that do not refine a heuristic function. One such technique is updating values on a per-state basis, e.g. in transposition tables (Akagi, Kishimoto, and Fukunaga 2010) or LRTA* (Korf 1990). Similarly, Wilt and Ruml (2013) use backward search to improve the heuristic estimation: Since the h^* value is known for a backward expanded node, it can be used to compute the minimal error of the heuristic and use it to update the heuristic values during forward search. Another example is refining combinations of multiple heuristics (e.g. (Felner, Korf, and Hanan 2004; Katz and Domshlak 2010)), but the individual heuristics remain unchanged.

In this work we introduce online refinement of additive Cartesian abstraction heuristics (Seipp and Helmert 2014). The refinement operation for these heuristics is based on splits of abstract states, which allows a locally restricted refinement in small steps and is well suited for online refinement. Seipp briefly touched online refinement in his Master's Thesis (Seipp 2012), but the explored design space is small and the approach was restricted to single abstractions.

Our online-refinement algorithm defines three basic operations: *refinement*, *merging* and *reordering*. Refinement extends individual abstractions, using the same procedure that is also applied in offline refinement. The merge operation is necessary to preserve convergence against h^* when multiple additive abstractions are used. Finally, the reordering operation provides an alternative way to improve the heuristic by generating new orderings for the cost partitioning, as different orders are useful in different states (Seipp, Keller, and Helmert 2017). We combine these three operations to a monotone online-refinement procedure that converges to h^*.

We show how online refinement of Cartesian abstraction heuristics can be used in A* (Hart, Nilsson, and Raphael 1968) to improve the heuristic during search. We evaluate our approach on the IPC benchmarks and compare it to

offline-generated Cartesian abstraction heuristics.

Preliminaries

In the following we consider classical planning using the finite-domain representation (FDR) (Bäckström 1995). A planning task is a 5-tuple $\Pi = (V, A, c, I, G)$, where

- V is a finite set of *state variables* where each $v \in V$ has a finite domain $\mathcal{D}(v)$. A variable/value pair $v = d$ with $v \in V$ and $d \in \mathcal{D}(v)$ is called a *fact*.
- A is a finite set of *actions*. Each action $a \in A$ is a pair $(\mathsf{pre}_a, \mathsf{eff}_a)$ of partial variable assignments which are called preconditions and effects respectively.
- $c : A \mapsto \mathbb{R}_0^+$ is the *cost function*, mapping each action to a non-negative real number.
- I is a complete assignment of variables describing the *initial state*.
- G is a partial assignment of variables describing the *goal*.

The *state space* of Π is the labeled transition system $\Theta_\Pi = (S, L, c, T, I, S^G)$. The *states* S are the complete variable assignments. The value of a variable in a state $s \in S$ is denoted by $s(v)$. An action is *applicable* in a state s if $\mathsf{pre}_a \subseteq s$. In this case, the values for all variables $v \in V$ in the state $appl(s, a)$ resulting from applying a in s are defined as $appl(s, a)(v) := \mathsf{eff}_a(v)$ if $\mathsf{eff}_a(v)$ is defined and $appl(s, a)(v) := s(v)$ otherwise. The labels L of the state space correspond to the actions A and the cost function c to that of Π. The transition relation $T \subseteq S \times L \times S$ is defined as $T = \{s \xrightarrow{a} appl(s, a) \mid \mathsf{pre}_a \subseteq s\}$. The initial state I is the same as in Π. The *goal states* $S^G = \{s \in S \mid G \subseteq s\}$ are the states that satisfy G. A *plan* for Π is an iteratively applicable sequence of actions which starts in I and leads to a goal state $s \in S^G$. A plan is *optimal* if the summed up cost of all actions is minimal among all plans of I.

A heuristic function $h : S \mapsto \mathbb{R}_0^+ \cup \{\infty\}$ maps each state to a non-negative real number or ∞. We write $h[c_i]$ to denote that the heuristic h is computed on a modification of Π where the cost function c is replaced by c_i. The *perfect heuristic* h^* assigns each state s its *remaining cost*, which is the cost of an optimal plan for s, or ∞ if no plan for s exists. A heuristic h is *admissible* if $h(s) \leq h^*(s)$ for all $s \in S$ and *consistent* if $h(s) \leq h(s') + c(a)$ for all transitions $s \xrightarrow{a} s'$. Given the transition system $\Theta = (S, L, c, T, I, S^G)$, an *abstraction* of Θ is a surjective function $\alpha : S \mapsto S^\alpha$. The *abstract state space* induced by α, written Θ^α, is the transition system $\Theta^\alpha = (S^\alpha, L, c, T^\alpha, I^\alpha, S^{\alpha G})$ with $I^\alpha = \alpha(I)$, $S^{\alpha G} = \{\alpha(s) \mid s \in S^G\}$ and $T^\alpha = \{(\alpha(s), l, \alpha(t)) \mid (s, l, t) \in T\}$. By $\sim^\alpha$ we denote the *induced equivalence relation* on Θ, defined by $s \sim^\alpha t$ iff $\alpha(s) = \alpha(t)$ and the equivalence classes by $[s]$. The *heuristic function induced by* α, written h^α, is the heuristic function which maps each state $s \in S$ to $h^*_{\Theta^\alpha}(\alpha(s))$.

A *cost partitioning* for a planning task with actions A is a set of functions $\mathcal{C} = \{c_1, \ldots, c_n : A \mapsto \mathbb{R}_0^+\}$ such that for all $a \in A : \sum_{i=1}^n c_i(a) \leq c(a)$. We say that an admissible heuristic h has a *local error* in state $s \in S$ if it does not satisfy the *Bellman optimality equation*: $h(s) \leq min_{(s,a,s') \in T} h(s') + c(a)$.

Additive Cartesian Abstraction Heuristics

An abstraction is *Cartesian* if all its states are Cartesian sets, i.e., they have the form $A_1 \times \cdots \times A_n$, where $A_i \subseteq \mathcal{D}(v_i)$ for all $1 \leq i \leq n$. The abstraction is built starting with the trivial abstraction and iteratively splitting states using counterexample-guided abstraction refinement (Seipp and Helmert 2013), which we summarize in the following.

In every iteration, an optimal solution as a trace $\tau = \langle [s'_0], a_1, \ldots, [s'_{n-1}], a_n, [s'_n] \rangle$, an alternating sequence of abstract states and actions, is computed. If no solution exists, the problem is unsolvable. Otherwise we check if τ can be converted to a solution of the concrete state space. During iteratively applying the actions in τ, resulting in a sequence of concrete states $s_0, s_1, \ldots, s_n$, we check if one of the following flaws occurs:

1. The concrete state s_i does not fit the abstract state $[s'_i]$ in τ, i.e. $[s_i] \neq [s'_i]$.
2. The concrete trace is completed, but s_n is not a goal state.
3. The action a_{i+1} is not applicable in the concrete state s_i.

If none of the flaws occurs, we found a solution. Otherwise, a state can be split according to the following rules (the numbers correspond to the cases above):

1. Split $[s_{i-1}]$ into $[t']$ and $[u']$ such that $s_{i-1} \in [t']$ and a_i does not lead from a state in $[t']$ to a state in $[s'_i]$.
2. Split $[s_n]$ into $[t']$ and $[u']$ such that $s_n \in [t']$ and $[t']$ does not contain a goal state.
3. Split $[s_i]$ into $[t']$ and $[u']$ in such a way that $s_i \in [t']$ and a_{i+1} is inapplicable in all states in $[t']$.

As the size of the abstract state space grows larger, the number of refinement iterations that are necessary to result in an increase of the heuristic estimate also becomes larger. In order to avoid this problem, a set of multiple small abstractions can be used instead. Multiple abstractions can be generated by only considering one goal fact in each abstraction, such that each abstraction covers different parts of the planning task (Seipp and Helmert 2014).[1]

Cost partitionings can be used to admissibly combine a set of heuristics. The *saturated cost partitioning (SCP)* is an effective way to construct an additive ensemble of multiple Cartesian abstractions (Seipp and Helmert 2014).

For a heuristic h and cost function c, the *saturated cost function* $saturate(h, c)$ is defined as the minimal cost function $c' \leq c$ with $h[c'](s) = h[c](s)$ for all states s. Given a set of heuristic functions $\mathcal{H} = \{h_1, \cdots, h_n\}$ for Π and an order $\omega = (h_1, \cdots, h_n)$ of those functions, the *saturated cost partitioning* $\mathcal{C} = c_1, \cdots c_n$ and the *remaining cost functions* $\bar{c}_0, \cdots, \bar{c}_n$ are defined as

$$\bar{c}_0 = c$$
$$c_i = saturate(h_i, \bar{c}_{i-1})$$
$$\bar{c}_i = \bar{c}_{i-1} - c_i$$

[1] Seipp and Helmert also define a decomposition based on landmarks, which we do not consider here as it requires non-trivial extensions to the online refinement and merging procedures.

If h is an abstraction heuristic based on an abstract transition system T^α of Π with labels L, then the saturated cost function $\hat{c}(a)$ for $a \in L$ is defined as $\hat{c}(a) = \max_{s \xrightarrow{a} s' \in T^\alpha} \max\{0, h(s) - h(s')\}$. This ensures that each abstraction only uses the minimal amount of cost required to preserve the cost of an optimal plan from each state.

Running Example Our example consists of a robot who has to visit certain cells on a small grid (Figure 1).

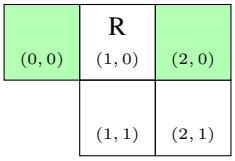


Figure 1: Sample task: the robot R must visit the green cells.

The state variables are the robot position *at* (which can be any of the five locations, initially $(1, 0)$) and the boolean variables v_{00} and v_{20} indicating if the corresponding cells have been visited (initially 0, must be 1 in the goal). The robot can move between adjacent cells $x, y \in \mathcal{D}(at), x \neq y$ with a move action $m(x, y)$ with preconditions $\{at = x\}$ and effects $\{at = y\}$. If the target position of the move is either one of the goal locations ($y = (0, 0)$ or $y = (2, 0)$), the effects include achieving the corresponding visited fact ($v_{00} = 1$ or $v_{20} = 1$ respectively). All action costs are 1, except the move action from $(1, 0)$ to $(1, 1)$, which costs 2.

The offline-refined Cartesian abstractions of the example are shown in Figure 2. The procedure starts with a trivial abstraction of a single abstract state for each goal. Initially, the abstract solution is empty because the abstract initial state is already a goal state. To prevent this flaw, the abstract state is split on the goal fact $v_{00} = 1$ respectively $v_{20} = 1$. This results in the abstractions shown in Figure 2. Since now the abstract solution corresponds to the concrete solution in the individual goal abstractions, the refinement terminates.

(a) $\mathcal{A}_1$ (b) $\mathcal{A}_2$

Figure 2: Abstractions of the running example after offline refinement. If a variable is not mentioned in a state all values are possible. Self loops are omitted. The SCP order is $\omega = \{\mathcal{A}_1, \mathcal{A}_2\}$. Goal states are marked in green. Actions are abbreviated, e.g. as $m(10, 00)$ instead of $m((1, 0), (0, 0))$.

Online Refinement Operations

In the following, we describe the three operations *refinement*, *merging*, and *reordering*, that make up our online-refinement approach.

Refinement of Additive Cartesian Abstractions

The *refinement* operation is based on the refinement algorithm described in the previous section. The essential modification for online refinement is the start state of the trace τ. While offline refinement always starts from the initial state, online refinement uses the current search state. If the solution for each individual goal is short, but the goals influence each other strongly, the abstractions refined offline largely underestimate the remaining cost. The reason is that an abstraction refined offline does not consider going into a wrong direction first, and states that are not on an optimal path for the initial state in the abstraction are never refined further.

If the sample abstractions are refined on the state $s_{ru} = \{at = (2, 0), v_{00} = 0, v_{20} = 1\}$ where the robot is in the right upper cell, $\mathcal{A}_1$ changes as shown in Figure 3. The action $m(10, 00)$ of the abstract solution is not applicable in s_{ru}, so the starting cell of the robot is split from the other cells. As a result, the heuristic value of the refined state increases from 1 to 2. The abstraction $\mathcal{A}_2$ does not change because it is refined on a goal state.

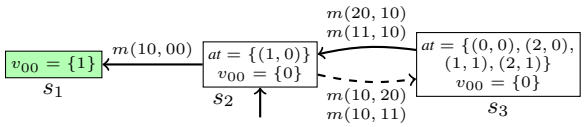


Figure 3: $\mathcal{A}_1$ after online refinement on the state $\{at = (2, 0), v_{00} = 1, v_{20} = 0\}$. Solid transitions have cost 1, dashed ones have cost 0.

Influence on Cost Partitioning After every refinement of the abstractions the cost partitioning needs to be recomputed. Here, two undesirable effects can occur. The first abstraction absorbs more and more of the cost. Thereby the impact of the additive component of the abstractions at the end of the cost partitioning order is diminished. Secondly, it is possible that the heuristic estimation of a state decreases after the cost is redistributed by the saturated cost partitioning algorithm, as illustrated in the following example.

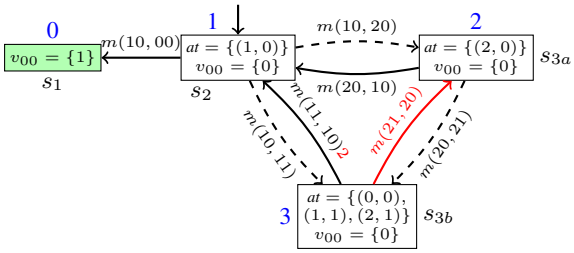


Figure 4: $\mathcal{A}_1$ after online refinement on state $at = (2, 1), v_{00} = 0, v_{20} = 1$. The blue numbers correspond to the remaining cost of the state.

If $\mathcal{A}_1$ in Figure 3 is further refined on the state $s_{rl} = \{at = (2, 1), v_{00} = 0, v_{20} = 1\}$, state s_3 is split as shown in Figure 4. The first action of the abstract solution $m(20, 10)$ is not applicable in s_{rl}, so the precondition $at = (2, 0)$ is split from s_3. The solid arrows indicate the actions which retain their cost after the *saturation* of the abstraction. As the action $m(10, 11)$ has a cost of 2, the cost of the action $m(21, 20)$ is necessary to preserve the optimal plan cost of s_{3b}. As a result, there is no cost for $m(10, 20)$ remaining in $\mathcal{A}_2$, which now evaluates to 0 for any state. Overall, in the

additive heuristic of $\mathcal{A}_1$ and $\mathcal{A}_2$, the heuristic estimation for the states abstracted by s_{3b} in Figure 4 increased by 1, while for all others it decreased by 1.

Both problems, the dominance of the first abstractions in the order and the decreasing estimation, can be solved by a slight adaptation of the SCP algorithm. Instead of completely redistributing the cost, every abstraction can keep the cost of the previous iteration, and only gains new cost from the cost which is not used by any other abstraction in the previous iteration. In the following, this cost is called *unused cost*. For the abstraction in Figure 4 this means that it can not use the cost of the action $m(21, 20)$, because it is already used by $\mathcal{A}_2$. Therefore, the heuristic estimation does not decrease for any state.

Definition 1 *Given the cost partitioning* $\mathcal{C}^{l-1} = \{c_1^{l-1}, \cdots, c_n^{l-1}\}$ *of the previous iteration, the* online saturated cost partitioning (OSCP) $\mathcal{C}^l = \{c_1^l, \ldots, c_n^l\}$ *and the remaining cost functions* $\bar{c}_0^l, \ldots, \bar{c}_n^l$ *are defined as*

$$\bar{c}_0^l = c - \sum_{j=1}^{n} c_j^{l-1} \quad \textit{(unused cost)}$$

$$c_i^l = \textit{saturate}(h_i, \bar{c}_{i-1}^l + c_i^{l-1})$$

$$\bar{c}_i^l = \bar{c}_{i-1}^l - c_i^l$$

Useful Splits Every split of an abstract state increases the memory size of the abstraction and the evaluation time of the heuristic. Hence, it is only useful to split an abstract state if this could increase the heuristic value of some state. If a state s is split into the states s' and s'', the heuristic can only increase if the cost of all actions in at least one direction between s' and s'' is greater than 0. Otherwise, it is still possible to move between these states for free and the split has no impact on the remaining cost of any abstract state. Exactly this behavior happens in the split of state s_3 in Figures 3 and 4. When performing the OSCP, none of the actions between the states s_{3a} and s_{3b} has a cost larger than zero. Therefore, the heuristic estimation can not increase.

In the following, a split of a state s is called *useful*, if all actions in at least one direction between the resulting states s' and s'' have a non-zero cost after recomputing the cost partitioning. The check if a split is useful is implemented by testing if there is still *unused cost* or cost reserved by the abstraction (in any order in $\mathcal{O}$, c.f. Section Reordering) for all actions in at least one direction between s' and s''. Since a non-useful split can sometimes be necessary to make a useful split reachable in refinement, it is possible that the useful split check prevents heuristic from increasing.

Merging

Originally the reason to use multiple small abstractions instead of one large abstraction was a slow increase in the heuristic estimation. But this separation of the goal facts prevents a convergence of the heuristic against h^*. This behavior can be observed for the initial state of the sample task. The heuristic value based on the two abstractions will never be 3 for the initial state, independent of the number of refinement operations. We can restore this convergence property by replacing two abstractions $\mathcal{A}_1$ and $\mathcal{A}_2$ by their *synchronized product* whenever further improvement based on refinement it not possible. The synchronized product are the non-empty intersections of the abstract states of $\mathcal{A}_1$ and $\mathcal{A}_2$. The merge result is Cartesian because the intersection of two Cartesian sets is again Cartesian (Seipp 2012).

Considering again our example, the synchronized product of $\mathcal{A}_1$ (Figure 3) and $\mathcal{A}_2$ (Figure 2) is shown in Figure 5.

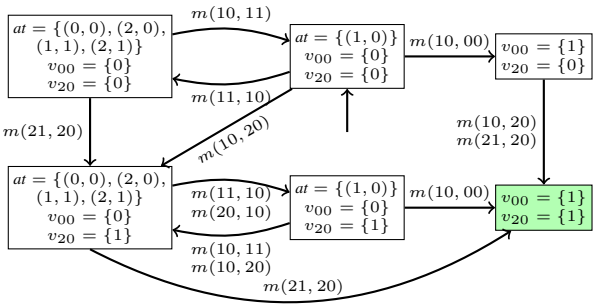


Figure 5: *Synchronized product* of the abstractions $\mathcal{A}_1$ (Figure 3) and $\mathcal{A}_2$ (Figure 2).

While the merge operation itself does not change the heuristic value, it allows further refinement operations to be performed on the resulting abstraction. Afterwards, the cost partitioning for all abstractions is recomputed using the sum of the cost functions of $\mathcal{A}_1$ and $\mathcal{A}_2$ as the cost function of the merge result.

Reordering

The order in which the cost functions for the saturated cost partitioning are computed can have a huge impact on the informativeness of the heuristic. The performance of the heuristic can be improved by using a set of orders $\mathcal{O}$. When evaluating a state, the heuristic can use the maximum estimation of all cost partitionings corresponding to the orders $\mathcal{O}$ (Seipp, Keller, and Helmert 2017). Diverse orders are obtained by generating several potentially useful orders, and only retaining those that lead to an improved estimation on at least one randomly sampled state.

These approaches can be transferred to the online phase to potentially gain better orders, because instead of random sample states, actual search states can be used.

We start out with one order based on the h^{add} value of the goal fact of the abstractions, following the default configuration by Seipp and Helmert (2014). If, during search, an order leading to a higher estimation for the current search state is found, it is added to $\mathcal{O}$ and can be used in all following states. If the structure of any abstraction changes, either by refinement or merging, the cost partitioning for each order $\omega \in \mathcal{O}$ is recomputed through OSCP.

When generating a new order, we order the abstractions by their impact on the current search state. More specifically, the abstractions are ordered descendingly according to their individual goal distance, using the original cost function. This strategy worked best in preliminary experiments.

Converging Online Refinement

We now describe our converging online-refinement procedure that combines the three introduced operations (Algorithm 1). Our approach relies on the Bellman equation to identify states with a local error, which the online refinement algorithm aims to correct.

Algorithm 1: Online Refinement

Input: An additive Cartesian abstraction heuristic h with abstractions $\mathcal{A}_1, \dots, \mathcal{A}_n$ and orders $\mathcal{O}$, and a state s where h does not satisfy Bellman

$\omega' :=$ FINDORDER(h, s)
$c_{\omega'} :=$ SCP(h, ω')
if $h(s)$ *increases when using* $c_{\omega'}$ **then**
  $\mathcal{O} := \mathcal{O} \cup \{\omega'\}$
while $\neg$BELLMAN(h, s) **do**
  for $i := 1, \dots, n$ **do**
    REFINE$(\mathcal{A}_i, s)$
  if *no abstraction* $\mathcal{A}_i$ *was modified* **then**
    Let $\mathcal{A}_x, \mathcal{A}_y$ be the two abstractions in h with the fewest abstract states
    MERGE$(\mathcal{A}_x, \mathcal{A}_y)$
  for $\omega \in \mathcal{O}$ **do**
    OSCP(h, ω)

First the algorithm tries to improve the heuristic by finding a better cost partitioning order for the current state. If this does not suffice to satisfy the Bellman equation, all abstractions are refined on the current search state until either there is no local error anymore or no further refinement is possible. In the latter case, the two smallest abstractions are merged to gain new refinement opportunities.

Theoretical Properties

Theorem 1 *Let $\Pi = (V, A, c, I, G)$ be a planning task, $\mathcal{H}$ be a set of Cartesian Abstraction heuristics, and $\mathcal{O}$ a set of orderings for $\mathcal{H}$. Then the heuristic estimation for any state can not decrease after applying any of the introduced operations (refining (i), merging (ii) or reordering (iii)) and subsequent recomputation of the cost partitioning for (i) and (ii).*

Proof Sketch:
For (i): Refinement of an abstraction without changing the cost function is monotone. As the OSCP does not decrease the cost of any action (unless decreasing the cost preserves the optimal plan cost for all states) it can not lead to a cheaper solution for any state. If the refinement of all heuristics is monotone then so is the sum of them.
For (ii): For any state s, an optimal plan p for s in the synchronized product of two abstractions $\mathcal{A}_1$ and $\mathcal{A}_2$ is also a (not necessarily optimal) plan in both $\mathcal{A}_1$ and $\mathcal{A}_2$. Let c_i be the cost function of $\mathcal{A}_i$ and c_M of the merge result. Then it suffices to show that $\sum_{a \in p} c_M(a) \geq \sum_{a \in p} c_1(a) + \sum_{a \in p} c_2(a)$ because an optimal plan in $\mathcal{A}_i$ is as most as expensive as p. The inequality holds since c_M is defined as $c_M = c_1 + c_2$. The recomputation of the cost partitioning is monotone as shown in (i).
For (iii): As orders are only added to $\mathcal{O}$ and we always take the maximum estimation of all orders, the estimation can only increase for any state.

Theorem 2 *Let $\Pi = (V, A, c, I, G)$ be a planning task, and h an additive Cartesian abstraction heuristic. Then using the refinement procedure described in Algorithm 1 the heuristic converges towards h^*.*

Proof Sketch:
Whenever no further refinement operations are possible, two abstractions are merged. If necessary, in the end this results in one big abstraction containing all goal facts. This leads to a convergence against h^* in every planning task, as the optimal plan in the merged abstraction will also be an optimal plan in the original task in the limit.

Online Refinement in A^*

The A* search algorithm needs one adaption to handle a dynamically changing heuristic. The open list stores the search nodes according to the sum of the heuristic estimation and the shortest known distance from the initial state. When the heuristic function changes, the open list must be resorted in order to always use the best known estimation in the expansion order. Other possibilities would be to restart the search or spawning parallel search processes (Arfaee, Zilles, and Holte 2011). Both approaches seem unsuitable if the heuristic changes frequently but locally restricted. Not updating the open list would lead to an admissible but *inconsistent* heuristic resulting in reopened search nodes. As the heuristic function can only increase for any state (Theorem 1), it is not necessary to reorder the entire open list. Instead, we can do this lazily: Every time a state is expanded, we check if the heuristic value using the current heuristic is the same as the one when the state was inserted into the open list. If this is the case the state is expanded, otherwise it is reinserted into the open list with the updated heuristic value. Whenever a state that is currently being expanded has a local error, the refinement procedure is called.

Experiments

We implemented our techniques in Fast Downward (FD) (Helmert 2006) based on the existing implementation of Cartesian abstraction heuristics (Seipp and Helmert 2013; 2014). The experiments were run on Intel Xenon E5-2650 v3 processors with a clock rate of 2.3 GHz. The time and memory limit were set to 30 minutes and 4 GB. As benchmarks we use all domains from the optimal tracks of all IPCs up to 2014 (excluding the trivial Movie domain), for a total of 1637 problem instances.

First, we look at the search behavior of our online-refinement algorithm and analyze the overhead added by online-refinement. As this overhead can sometimes be prohibitive, we devise additional configurations in an attempt to reduce this. We compare our configurations to offline-refined Cartesian abstraction heuristics.

Overview

For our base configuration h^{on}, we initialize the heuristic with 1000 (offline-refined) abstract states in total. During se-

arch, we apply our online-refinement procedure in each state until the Bellman equation is satisfied.

As our comparison baseline, we use an offline-refined heuristic h^{off} with a refinement timeout of 15 minutes. The cost partitioning order uses the default setting of a descending order of the h^{add} values of the goal facts that correspond to the individual abstractions.

Domain	h^{on}	h^{off}	$h^{\text{on}}_{0.1}$	*exp*	*time*	h^{SCP}_{div}
Airport (50)	23	**34**	33	0.02	0.57	30
Barman (34)	0	**4**	**4**	1.02	2.90	**4**
Blocksworld (35)	10	18	22	0.24	1.07	**28**
Childsnack (20)	0	0	0	–	–	0
Depot (22)	2	5	9	0.11	0.38	**11**
Driverlog (20)	8	11	**14**	0.08	0.41	**14**
Elevators (50)	36	37	42	0.52	1.43	**44**
Floortile (40)	0	2	**4**	0.22	1.08	2
FreeCell (80)	6	19	20	0.11	1.04	**65**
GED (20)	5	15	**16**	1.09	4.37	15
Grid (5)	1	2	**3**	0.02	0.30	**3**
Gripper (20)	6	**8**	7	0.77	3.84	**8**
Hiking (20)	6	12	**13**	0.71	3.35	**13**
Logistics (63)	28	26	30	0.07	0.50	**39**
Miconic (150)	104	63	100	< 0.01	0.19	**144**
Mprime (35)	26	**29**	**29**	0.32	1.87	27
Mystery (30)	16	**18**	**18**	0.06	2.58	17
Nomystery (20)	11	16	**20**	0.10	0.37	**20**
Openstacks (100)	18	49	45	0.55	7.10	**51**
Parcprinter (50)	28	20	32	0.13	0.86	**39**
Parking (40)	0	0	3	–	–	**8**
Pathways (30)	4	4	**5**	0.18	0.98	4
Pegsol (50)	6	46	**48**	0.30	4.47	**48**
Pipesw.-NT (50)	4	17	21	0.21	0.78	**23**
Pipesw.-T (50)	4	14	**16**	0.47	1.21	**16**
PSR (50)	48	**49**	**49**	1.16	2.02	**49**
Rovers (40)	4	8	**10**	0.25	0.76	7
Satellite (36)	4	6	**7**	0.03	0.55	**7**
Scanalyzer (50)	11	21	**23**	0.16	1.04	**23**
Sokoban (50)	44	41	**45**	0.53	1.50	**45**
Storage (30)	10	**16**	15	1.26	2.43	**16**
Tetris (17)	1	**9**	**9**	0.25	0.66	**9**
Tidybot (40)	3	26	**30**	1.45	2.24	22
TPP (30)	7	**11**	8	–	0.98	8
Transport (70)	7	24	**28**	0.68	2.07	25
Trucks (30)	3	10	10	0.17	0.70	**12**
Visitall (40)	12	13	13	0.16	1.07	**16**
Woodw. (50)	18	21	**35**	< 0.01	0.24	32
Zenotravel (20)	8	12	**13**	0.17	0.80	**13**
aggregate (1637)	562	766	879	0.19	1.11	**987**

Table 1: Coverage for the basic online-refinement approach h^{on}, the baseline h^{off}, and online-refinement with restricted refinement time $h^{\text{on}}_{0.1}$ in the leftmost columns. The middle columns show the ratio of the expansions until the last f-layer is reached and search time for $h^{\text{on}}_{0.1}$ compared to h^{off}. The rightmost column shows coverage data for a state-of-the-art configuration of Cartesian abstraction heuristics.

In the two leftmost columns of Table 1, the coverage of both configurations is displayed. The online-refinement version solves a total of 562 tasks, 204 less than the offline version. In 32 domains h^{off} solves more tasks than h^{on}, in 3 domains they solve equally many, and in 4 domains h^{on} solves more tasks. Our online refinement approach works best in the Miconic domain, where it solves 104 instances compared to 63 with h^{off}. Online refinement seems unsuitable for the Openstacks, Pegsol, and Tidybot domains, as the overall coverage drops by 31, 40, and 23 respectively.

The left side of Figure 6 compares the number of expansions until the last f-layer is reached for commonly solved instances of h^{on} and h^{off}. With very few exceptions, h^{on} needs significantly fewer expansions, up to 5 orders of magnitude fewer in some instances. On larger instances, this observation becomes more pronounced, as the heuristic is more frequently refined and in the end much more informative than h^{off}. Since initially h^{off} may have more abstract states than h^{on}, on some (very few) smaller instances there are cases where more expansions are necessary.

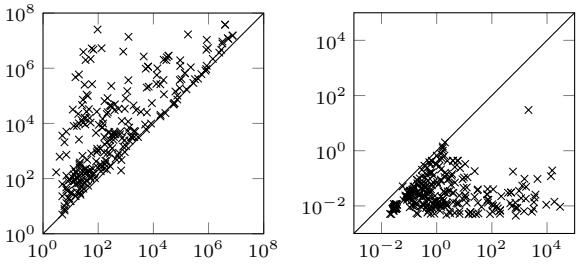


Figure 6: Left: Number of expansions until the last f-layer. Right: Search time per expanded state per task in ms. The x- and y-axes correspond to h^{on} respectively h^{off}.

This decrease in expansions comes with a trade-off in search time, as shown on the right side of Figure 6. While the maximum time for each expansion is consistently low in h^{off}, h^{on} can spend a lot of time in refinement and use up almost the entire search time to refine a few states. On commonly solved instances, the search time for h^{on} is 15 times larger than for h^{off} on average. Exceptions are Logistics, Miconic, and Woodworking, where h^{on} has lower search time, resulting in higher coverage in Miconic and Logistics.

Operation Time Distribution A significant fraction of the search time is used to improve the heuristic. This time is distributed over the three operations *refine*, *merge* and *reorder* (each including the recomputation of the cost partitioning and the updating of the stored h^* values in the abstraction), evaluating the *Bellman* equation, and updating the *open list*. Figure 7 shows the time distribution for each domain. The percentages are averaged over all instances of the domain (including unsolved ones).

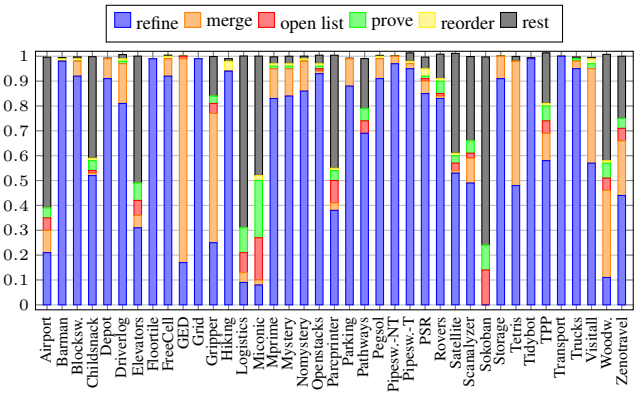


Figure 7: Average ratio between the time to improve the heuristic and the search time. The improvement time is split in five parts. Displayed is the average ratio per domain excluding tasks which have been solved in 0.01s or less.

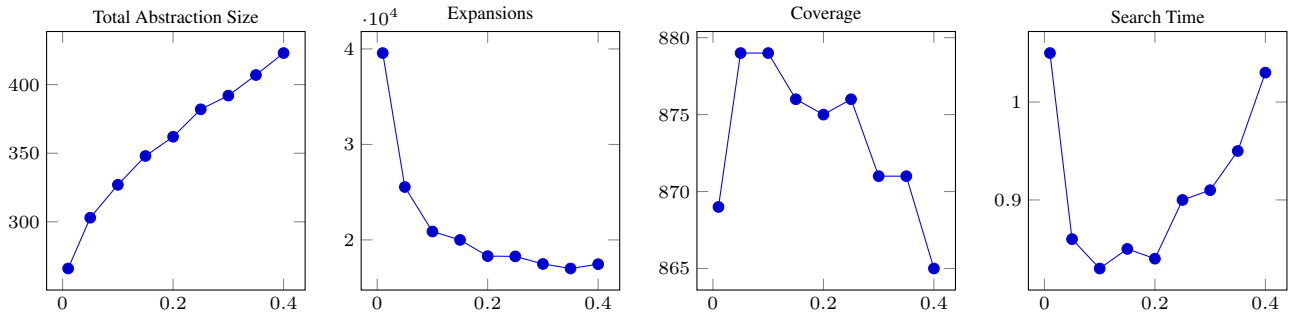


Figure 8: Results for h^{on}_p with p ranging from 0.01 to 0.4.

On average, about two thirds of the search time is used only to refine abstractions, but the variance is high and heavily depends on the domain. The extreme cases are Sokoban, where *no* time at all is spent on refinement, and Transport, with 99%. In Sokoban, there are applicable zero-cost actions in every state leading to states with equal heuristic value, so the Bellman equation is always satisfied. The impact of the *merge* operation depends on the number of goals facts. In domains with many goal facts, e.g. GED or Gripper, there are many small abstractions which need to be merged to enable further refinement operations. With about 1% the *reorder* time is negligible in all domains, which can be attributed to our simple ordering strategy. The time spent on the *Bellman* equation check highly depends on the branching factor of the domain as more heuristic values must be compared. In most domains, this accounts for less than 10% of the overall time, the only exceptions are Miconic (17%) and Sokoban (14%). The *open list* time is below 5% in almost all domains. This is mainly due to the fact that the open list stays relatively small due to the low number of expanded states.

Used Cost The OSCP algorithm is based on the assumption that there is still *unused cost*. In Figure 9 the average fraction of used cost per domain is shown, both for the initial abstractions and the final value when the instance is solved or the timeout is reached.

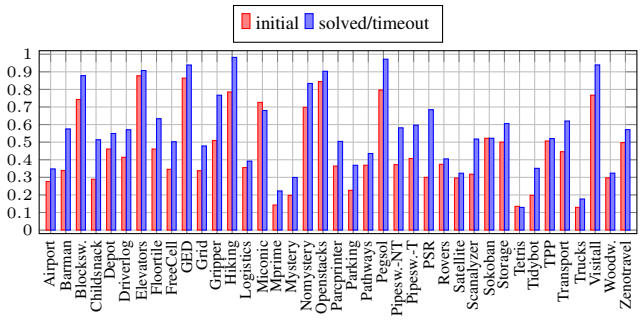


Figure 9: Fraction of the actions which are used in any abstraction with a non-zero cost. Displayed is the average over all orders per domain for the initial abstractions, and after the instance is solved or the time limit is reached.

On average only 45% of the actions are used with non-zero cost in any abstraction in the beginning of the search, so typically there is enough unused cost that can be distributed among the abstractions in the OSCP. However, this number has a high variance depending on the domain, and there are some domains where almost the entire cost is used in the beginning already (e.g. Elevators and GED). In some domains, there is still a large amount of unused cost remaining, even at the end of the search (Mprime, Mystery, Tetris, Trucks).

Reducing the Refinement Overhead

If every local erroneous state is refined, the overhead introduced by the online-refinement procedure is very high. This leaves only little time remaining for the actual search process (c.f. the grey bars in Figure 7).

Addressing this issue, we experimented with additional configurations h^{on}_p where a parameter p is introduced that limits the time spent on refinement to a fixed fraction of the overall search time. The refinement process is only executed, if currently the fraction of the overall search time that is spent on refinement is below that threshold. The time spent to satisfy the Bellman equation in one state can still be very high. Therefore, we only perform at most one refinement operation in each state and do not merge any abstractions.

A graphical overview of the results for these configurations, using the values 0.01, 0.05, 0.10, ..., 0.40 is shown in Figure 8 and 0.1 for p. As we increase the refinement parameter p, the final abstraction size increases and with it the number of expansions needed to reach the final f-layer decreases. On the other hand, too much refinement overhead is also detrimental to the overall performance of this approach. The sweet spot lies at $p = 0.1$, where the highest overall coverage of 879 and lowest average search time is reached.

Compared to our base configuration h^{on}, the increase in coverage is consistent across almost all domains (the only exception is Miconic). Our configuration with restricted refinement diminishes the negative effect of the refinement overhead, and considerably improves over both h^{on} and h^{off}. It has a higher coverage than h^{off} in 26 domains, and only loses in 5 domains. On average, the number of expansions until the last f-layer is reduced by 81% (c.f. column "exp" in Table 1). However, the search time on commonly solved instances is often greater due to the added overhead of online refinement on instances where it is not required (c.f. column "time" in Table 1).

Comparison to State of the Art As a comparison to the state of the art in Cartesian abstractions, we compare our best performing configuration $h^{\text{on}}_{0.1}$ to an additive Cartesian

abstractions heuristic $h^{\text{off}}_{\text{div}}$ that also uses landmark decomposition, and uses a diversified set of greedily instantiated orders for the saturated cost partitioning (Seipp 2017).

The results for $h^{\text{off}}_{\text{div}}$ are shown in the rightmost column in Table 1. In terms of overall coverage, $h^{\text{off}}_{\text{div}}$ beats our approach by a large margin, but mostly due to the big gaps in the FreeCell ($+45$) and Miconic ($+44$) domains. In 14 domains $h^{\text{off}}_{\text{div}}$ has higher coverage than $h^{\text{on}}_{0.1}$, while our approach works better in 10 domains. The biggest advantage for online refinement can be observed in Tidybot ($+8$).

Useful Splits In order to best evaluate the impact of the *useful split* check we use our h^{on}_p configuration with $p = 1$. Enabling this check can prevent the Bellman equation from being satisfied, so we can not use h^{on}.

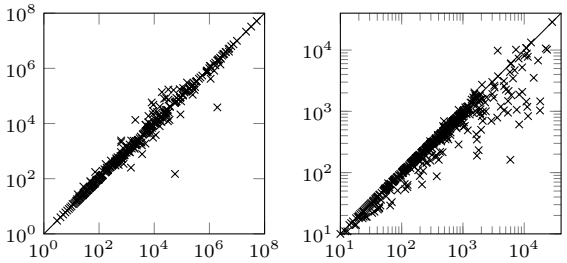


Figure 10: Left: Number of expansions until last f-layer. Right: Number of abstract states. The x- and y-axes correspond to without respectively with useful splits.

Figure 10 shows the number of expansions until the last f-layer is reached and the size of the abstractions. It shows that using the useful splits check can sometimes significantly reduce the size of the resulting abstractions while retaining very similar heuristic informativeness.

This improvement also translates to a higher coverage (709 with vs. 682 without useful splits). The domains benefitting the most are Pegsol ($+8$) and Scanalyzer ($+4$), but there are also domains where the coverage decreases, e.g. -3 in GED. For $h^{\text{on}}_{0.1}$, enabling the useful split check did not improve the overall results.

Online vs. Offline Refinement

Finally, we want to examine whether refinement based on the current search states leads to a more informed heuristic than doing refinement only in the initial state. While we already showed that h^{on} can reach the final f-layer with much fewer expansions than h^{off} (Figure 6), in that comparison the online-refined abstractions were allowed to grow a lot bigger than those generated offline.

In order to create a fair environment, both abstractions should have the same number of abstract states. The abstract state space size using goal abstractions and only offline refinement is often strictly limited. Hence, for this comparison, we use only a single abstraction containing all goal facts.

For this experiment, we first do a run with online refinement, starting from the trivial abstraction, and performing the online refinement procedure until each state satisfies the Bellman equation. After this run (when a solution is found or a time limit of 15 minutes is reached), we restart the search and use the resulting abstraction $\mathcal{A}^{\text{on}}$ without further online refinement. We compare this setting to a run with an offline-refined abstraction $\mathcal{A}^{\text{off}}$, using the number of abstract states of $\mathcal{A}^{\text{on}}$ as the abstract state space size bound during offline refinement.

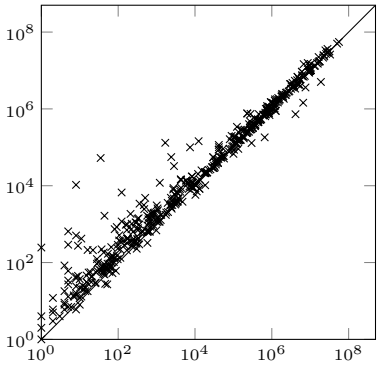


Figure 11: Comparison of the expansions until the last f-layer. The x- and y-axes correspond to $\mathcal{A}^{\text{on}}$ respectively $\mathcal{A}^{\text{off}}$.

Figure 11 compares the number of expansions until reaching the final f-layer for both resulting heuristics. The online-refined abstractions tend to need fewer expansions, on commonly solved instances the expansions are reduced to a factor of only 0.66 compared to $\mathcal{A}^{\text{off}}$. In only 4 out of 37 domains $\mathcal{A}^{\text{off}}$ is better, in all other domains using $\mathcal{A}^{\text{on}}$ results in a smaller search space. The greatest search space reduction can be observed in the domains Grid (0.12), Hiking (0.14), and Mprime (0.09). This also leads to a better overall coverage for $\mathcal{A}^{\text{on}}$ (690 vs. 679). Interestingly, the initial heuristic value is often much lower with $\mathcal{A}^{\text{on}}$ compared to $\mathcal{A}^{\text{off}}$ (4.2 vs. 14.2, geometric mean over all instances). This shows that the fineness of the abstract state space is distributed more evenly in the online-refined abstraction.

Conclusion

We introduced a monotone converging online-refinement procedure for a set of additive Cartesian abstraction heuristics consisting of the three operations *refine*, *merge*, and *reorder*. Our results show that online refinement considerably improves the accuracy of the heuristic, but it has to be used carefully to avoid prohibitive overhead. When this overhead is bounded to a managable amount, our approach significantly improves over a heuristic using basic offline-refined abstractions, and even beats a heuristic using additional techniques such as landmark decomposition and greedily instantiated cost partitioning orders on many domains. In principle, these techniques could be combined with online refinement as well, so there is still more potential.

Another interesting direction for future work is devising more sophisticated strategies for refinement (i.e. which states to refine and how much). Similarly, different strategies to select which abstractions to merge could be tried, in particular for domains with many goals (and thus, many individual abstractions that can be merged).

References

Akagi, Y.; Kishimoto, A.; and Fukunaga, A. 2010. On transposition tables for single-agent search and planning: Summary of results. In Felner, A., and Sturtevant, N. R., eds., *Proceedings of the 3rd Annual Symposium on Combinatorial Search (SOCS'10)*. Stone Mountain, Atlanta, GA: AAAI Press.

Arfaee, S. J.; Zilles, S.; and Holte, R. C. 2011. Learning heuristic functions for large state spaces. *Artificial Intelligence* 175(16-17):2075–2098.

Bäckström, C. 1995. Expressive equivalence of planning formalisms. *Artificial Intelligence* 76(1–2):17–34.

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

Edelkamp, S. 2001. Planning with pattern databases. In Cesta, A., and Borrajo, D., eds., *Proceedings of the 6th European Conference on Planning (ECP'01)*, 13–24. Springer-Verlag.

Felner, A.; Korf, R.; and Hanan, S. 2004. Additive pattern database heuristics. *Journal of Artificial Intelligence Research* 22:279–318.

Fickert, M., and Hoffmann, J. 2017. Complete local search: Boosting hill-climbing through online heuristic-function refinement. In *Proceedings of the 27th International Conference on Automated Planning and Scheduling (ICAPS'17)*. AAAI Press.

Fickert, M.; Hoffmann, J.; and Steinmetz, M. 2016. Combining the delete relaxation with critical-path heuristics: A direct characterization. *Journal of Artificial Intelligence Research* 56(1):269–327.

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(2):100–107.

Haslum, P.; Botea, A.; Helmert, M.; Bonet, B.; and Koenig, S. 2007. Domain-independent construction of pattern database heuristics for cost-optimal planning. In Howe, A., and Holte, R. C., eds., *Proceedings of the 22nd National Conference of the American Association for Artificial Intelligence (AAAI'07)*, 1007–1012. Vancouver, BC, Canada: AAAI Press.

Helmert, M.; Haslum, P.; Hoffmann, J.; and Nissim, R. 2014. Merge & shrink abstraction: A method for generating lower bounds in factored state spaces. *Journal of the Association for Computing Machinery* 61(3).

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

Katz, M., and Domshlak, C. 2010. Optimal admissible composition of abstraction heuristics. *Artificial Intelligence* 174(12–13):767–798.

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

Korf, R. E. 1990. Real-time heuristic search. *Artificial Intelligence* 42(2-3):189–211.

Seipp, J., and Helmert, M. 2013. Counterexample-guided Cartesian abstraction refinement. In Borrajo, D.; Fratini, S.; Kambhampati, S.; and Oddi, A., eds., *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS'13)*, 347–351. Rome, Italy: AAAI Press.

Seipp, J., and Helmert, M. 2014. Diverse and additive cartesian abstraction heuristics. In Chien, S.; Do, M.; Fern, A.; and Ruml, W., eds., *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS'14)*. AAAI Press.

Seipp, J.; Keller, T.; and Helmert, M. 2017. Narrowing the gap between saturated and optimal cost partitioning for classical planning. In Singh, S., and Markovitch, S., eds., *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI'17)*, 3651–3657. AAAI Press.

Seipp, J. 2012. Counterexample-guided abstraction refinement for classical planning. Master's thesis, University of Freiburg, Germany.

Seipp, J. 2017. Better orders for saturated cost partitioning in optimal classical planning. In Fukunaga, A., and Kishimoto, A., eds., *Proceedings of the 10th Annual Symposium on Combinatorial Search (SOCS'17)*. AAAI Press.

Wilt, C. M., and Ruml, W. 2013. Robust bidirectional search via heuristic improvement. In desJardins, M., and Littman, M., eds., *Proceedings of the 27th AAAI Conference on Artificial Intelligence (AAAI'13)*. Bellevue, WA, USA: AAAI Press.

Accounting for Partial Observability in Stochastic Goal Recognition Design: Messing with the Marauder's Map

Christabel Wayllace[†] and **Sarah Keren**[‡] and **William Yeoh**[†]
and **Avigdor Gal**[‡] and **Erez Karpas**[‡]
[†]Washington University in St. Louis
[‡]Technion – Israel Institute of Technology

Abstract

Given a stochastic environment and a set of allowed modifications, the task of goal recognition design is to select a valid set of modifications that minimizes the expected maximal number of steps an agent can take before his goal is revealed to an observer. This paper extends the *stochastic goal recognition design* (S-GRD) framework in the following two ways: (1) Agent actions are unobservable; and (2) Agent states are only partially observable. These generalizations are motivated by practical applications such as agent navigation, where agent actions are unobservable yet his state (current location) can be (at least partially) observed, using possibly low sensor (e.g., GPS) resolution, forcing nearby states to become indistinguishable. In addition to the generalized model, we also provide accompanying algorithms that calculate the expected maximal number of steps, offer new sensor refinement modifications that can be applied to enhance goal recognition, and evaluate them on a range of benchmark applications.

Introduction

Goal recognition aims at discovering the goals of an agent according to his observed behavior, collected online (Carberry 2001; Ramírez and Geffner 2010; Sukthankar *et al.* 2014). *Goal recognition design* (GRD) (Keren *et al.* 2014) is the offline task of redesigning environments (either physical or virtual) to allow efficient online goal recognition.

Typically, a GRD problem has two components: (1) The goal recognition setting being analyzed and a measure of the efficacy of goal recognition and (2) a model of possible design changes one can make to the underlying environment. In the seminal work by Keren *et al.* [2014], they proposed the *worst case distinctiveness* (*wcd*) metric, which aims at capturing the maximum number of steps an agent can take without revealing its goal, as a measure of the goal recognition efficacy. Removal of actions was considered as a possible design change to the environment. This definition is made for the problem under three key assumptions:

- **Assumption 1**: Agents in the system execute optimal plans to reach their goals;
- **Assumption 2**: The environment is fully observable (i.e., both states and actions of agents are observable); and
- **Assumption 3**: Agent actions are deterministic.

The GRD problem has been since generalized to relax each of the three assumptions (Keren *et al.* 2015; 2016a; 2016b; Wayllace *et al.* 2016; 2017; Keren *et al.* 2018). Aside from these relaxations, Wayllace *et al.* [2017] have also proposed a new metric called *expected case distinctiveness* (*ecd*), which weighs the possible goals based on their likelihood of being the true goal. Additionally, Keren *et al.* [2016b] have proposed the refinement of sensors, which decreases the degree of observation uncertainty on tokens produced by actions (rather than states) of an agent, as a possible design change to the environment. Table 1 summarizes the generalizations, metrics, and possible designs of existing GRD models.

In this work, we go beyond the state-of-the-art by extending the *Stochastic GRD* (S-GRD) model (Wayllace *et al.* 2017) to also relax Assumption 2 and handle partially observable environments. The new model, which we call *Partially-Observable Stochastic GRD* (POS-GRD), assumes that actions of the agent are now no longer observable and states of the agent are now partially observable. This relaxation is motivated by practical applications such as agent navigation, where agent actions are unobservable yet his state (current location) can be (at least partially) observed. The partial observability of agent states is due to low sensor (e.g., GPS) resolution – several nearby states may be indistinguishable from one another. Finally, we also consider sensor refinement as a possible design to the environment, which contributes to the observability of states.

Our empirical evaluation shows that partial observability increases the *wcd* required to recognize the agent's goal and that sensor refinement always reduces this value. The analysis also suggests that, given a limited number of possible modifications, the initial sensor configuration affects the value of *wcd* and its reduction ratio; therefore, it might be possible to reduce the *wcd* even further using the same number of sensors with the same resolution but in a different configuration.

Illustrating Example

To illustrate the setting of this work, we present an example from the wizarding world of Harry Potter, who is back at the Hogwarts School of Witchcraft and Wizardry. He is tasked with establishing a security system that can detect as early as possible a student who enters the school from the main entrance and is heading towards Professor Snape's office armed with a wand made of oak wood. All students use

	Generalizations			Metrics		Possible Designs	
	Suboptimal Plans	Partially Obs. Env.	Stochastic Actions	*wcd*	*ecd*	Action Removal	Sensor Refinement
Keren *et al.* [2014]				✓		✓	
Son *et al.* [2016]				✓		✓	
Keren *et al.* [2015]	✓			✓		✓	
Keren *et al.* [2016a]	✓	✓		✓		✓	
Keren *et al.* [2016b]	✓	✓		✓		✓	✓
Wayllace *et al.* [2016]			✓	✓		✓	
Wayllace *et al.* [2017]			✓	✓	✓	✓	
Our proposed model		✓	✓	✓		✓	✓

Table 1: Properties of Current Goal Recognition Design Models

the staircase chamber to move around the school. The staircase is stochastic, especially when a student is walking up the staircase. Therefore, a student aiming at a certain part of the school may find herself at a different location. For example, a student heading to Professor Snape's office or the dining hall may find herself at the hallway, in which case she needs to retrace to the entrance and try reaching her destination again. Figure 1(top) depicts the locations as nodes and the transitions (and their probabilities) as edges.

To accomplish his task, Potter plans to use the Marauder's Map, a magical artifact that reveals the whereabout of all witches and wizards at Hogwarts. The map can show where witches and wizards are, but due to some dark magic, it can no longer identify them by their names. Further, it was not created with the ability to detect whether a student carries a wand, not to mention the type of wood of which the wand is made.

Potter can cast exactly *one* spell to either reveal the name of the witches and wizards on the map or to reveal the type of wand that they are carrying, if any. Knowing that all wands are forbidden in the dining hall, Potter realizes that his best choice is to cast the spell that reveals wands and the type of wood of which they are made. This will guarantee that anyone ending up at the hallway with a wand made of oak wood and heads back to the entrance has the intention of reaching Professor Snape's office, and such a recognition can occur after at most two actions, namely moving towards Professor Snape's office but ending up in the hallway and returning to the entrance. Figure 1(bottom) illustrates the problem after the modification spell; each node now represents a $\langle$ location, wand $\rangle$ tuple, where nodes in red represent the states with oak wands.

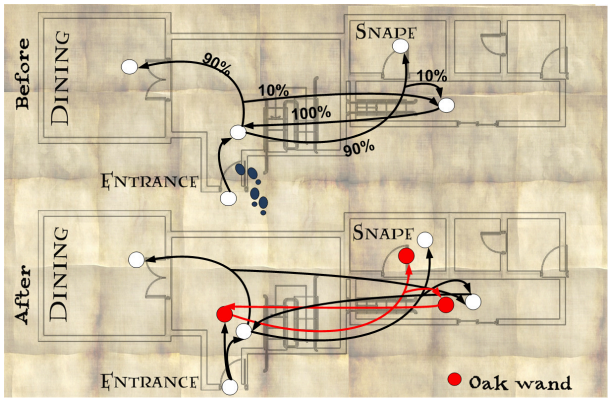


Figure 1: Marauder's Map Before (top) and After (bottom) Potter's Modification Spell

Background

Markov Decision Process (MDP)

A *Stochastic Shortest Path Markov Decision Process* (SSP-MDP) (Mausam and Kolobov 2012) is represented as a tuple $\langle \mathbf{S}, s_0, \mathbf{A}, \mathbf{T}, \mathbf{C}, \mathbf{G} \rangle$. It consists of a set of states $\mathbf{S}$; a start state $s_0 \in \mathbf{S}$; a set of actions $\mathbf{A}$; a transition function $\mathbf{T} : \mathbf{S} \times \mathbf{A} \times \mathbf{S} \to [0, 1]$ that gives the probability $T(s, a, s')$ of transitioning from state s to s' when action a is executed; a cost function $\mathbf{C} : \mathbf{S} \times \mathbf{A} \times \mathbf{S} \to \mathbb{R}$ that gives the cost $C(s, a, s')$ of executing action a in state s and arriving in state s'; and a set of goal states $\mathbf{G} \subseteq \mathbf{S}$. The goal states are terminal, that is, $T(g, a, g) = 1$ and $C(g, a, g) = 0$ for all goal states $g \in \mathbf{G}$ and actions $a \in \mathbf{A}$.

An SSP-MDP must also satisfy the following two conditions: (1) There must exist a *proper policy*, which is a mapping from states to actions with which an agent can reach a goal state from any state with probability 1. (2) Every *improper policy* must incur an accumulated cost of ∞ from all states from which it cannot reach the goal with probability 1. In this paper, we will focus on SSP-MDPs and will thus use the term MDPs to refer to SSP-MDPs. A *solution* to an MDP is a policy π, which maps states to actions. Solving an MDP means finding an optimal policy, that is, a policy with the smallest expected cost. Finally, we use *optimal actions* to denote actions in an optimal policy.

Value Iteration (VI) and Topological VI (TVI)

Value Iteration (VI) (Bellman 1957) is one of the fundamental algorithms to find an optimal policy. It uses a value function V to represent expected costs. The expected cost of an optimal policy π^* for the starting state $s_0 \in \mathbf{S}$ is the expected cost $V(s_0)$, and the expected cost $V(s)$ for all states $s \in \mathbf{S}$ is calculated using the Bellman equation (Bellman

1957):

$$V(s) = \min_{a \in \mathbf{A}} \sum_{s' \in \mathbf{S}} T(s, a, s') \big[C(s, a, s') + V(s') \big] \quad (1)$$

The action chosen by the policy for each state s is then the one that minimizes $V(s)$.

VI suffers from a limitation that it updates each state in every iteration even if the expected cost of some states have converged. *Topological VI* (TVI) (Dai *et al.* 2011) addresses this limitation by repeatedly updating the states in only one *strongly connected component* (SCC) until their values converge before updating the states in another SCC. Since the SCCs form a directed acyclic graph, states in an SCC only affect the states in upstream SCCs. Thus, by choosing the SCCs in reverse topological sort order, it no longer needs to consider SCCs whose states have converged in a previous iteration.

Goal Recognition Design (GRD)

A *Goal Recognition Design* (GRD) problem (Keren *et al.* 2014) is represented as a tuple $T = \langle P, \mathcal{D} \rangle$, where P is an initial goal recognition model and $\mathcal{D}$ is a design model. The initial model P, in turn, is represented by the tuple $\langle D, G \rangle$, where D captures the domain information and $\mathbf{G}$ is a set of possible goal states of the agent. The *worst case distinctiveness* (*wcd*) of problem P is the length of a longest sequence of actions $\pi = \langle a_1, \ldots, a_k \rangle$ that is the prefix in *cost-minimal* plans $\pi^*_{g_1}$ and $\pi^*_{g_2}$ to distinct goals $g_1, g_2 \in \mathbf{G}$. Intuitively, as long as the agent executes π, he does not reveal his goal to be either g_1 or g_2.

A design model $\mathcal{D}$ (Keren *et al.* 2018) includes three components: The set $\mathcal{M}$ of modifications that can be applied to a model; a modification function δ that specify the effect each modification $m \in \mathcal{M}$ has on the goal recognition setting to which it is applied; and a constraint function ϕ that specifies the modification sequences that can be applied to a goal recognition model. In the original GRD problem definition, action removals are the only modifications allowed in the design model.

The objective in GRD is to find a feasible modification sequence that, when applied to the initial goal recognition model P, will minimize the *wcd* of the problem. This optimization problem is subject to the requirement that the minimal cost to achieve each goal $g \in \mathbf{G}$ is the same before and after the modifications.

Researchers have proposed a number of extensions to support different goal recognition and goal recognition design models, tabulated in Table 1.

Stochastic GRD (S-GRD)

Stochastic Goal Recognition Design (S-GRD) (Wayllace *et al.* 2016; 2017) extends the GRD framework by assuming the actions executed by the agent, which are fully observable, have stochastic outcomes. Similar to GRD, it is represented as a tuple $T = \langle P, \mathcal{D} \rangle$, where $P = \langle D, G \rangle$ is an initial goal recognition model, $\mathcal{D}$ is a design model, D captures the domain information, and $\mathbf{G}$ is a set of possible goal states of the agent.

The elements of $D = \langle \mathbf{S}, s_0, \mathbf{A}, \mathbf{T}, \mathbf{C} \rangle$ of S-GRD problems are as described in MDPs, except that the cost function $\mathbf{C}$ is restricted to positive costs. It is assumed that the cost of all actions is 1 for simplicity. The *worst case distinctiveness* (*wcd*) of problem P is the largest *expected* cost incurred by the agent without revealing his true goal. The *wcd* of a problem assumes that all goals are of equal likelihood of being the true goal. The *expected case distinctiveness* (*ecd*) weighs the expected cost of each policy for a goal by the likelihood of that goal to be the true goal.

Augmented MDP for S-GRDs: Given a regular MDP $\langle \mathbf{S}, s_0, \mathbf{A}, \mathbf{T}, \mathbf{C}, \mathbf{G} \rangle$, an augmented MDP $\langle \tilde{\mathbf{S}}, \tilde{s}_0, \tilde{\mathbf{A}}, \tilde{\mathbf{T}}, \tilde{\mathbf{C}}, \tilde{\mathbf{G}} \rangle$ augments each component of the tuple in the following way:

- Each state $\tilde{s} \in \tilde{\mathbf{S}}$ is represented by $\langle s, \mathbf{G}' \rangle$ where $s \in \mathbf{S}$ and $\mathbf{G}' \subseteq \mathbf{G}$ is the set of possible goals for s. Two augmented states are different if any of their components are different.
- The augmented start state is $\tilde{s}_0 = \langle s_0, \mathbf{G} \rangle$.
- Each augmented action $\tilde{a} \in \tilde{\mathbf{A}}$ is a tuple $\langle a, \mathbf{G}' \rangle$, where $a \in \mathbf{A}$ and $\mathbf{G}'$ is the set of all goals for which that action is an *optimal action*.
- The new transition function $\tilde{\mathbf{T}} : \tilde{\mathbf{S}} \times \tilde{\mathbf{A}} \times \tilde{\mathbf{S}} \to [0, 1]$ gives the probability $\tilde{T}(\tilde{s}, \tilde{a}, \tilde{s}')$, where $\tilde{s} = \langle s, \mathbf{G}' \rangle$, $\tilde{a} = \langle a, \mathbf{G}'' \rangle$, and $\tilde{s}' = \langle s', \mathbf{G}' \cap \mathbf{G}'' \rangle$. $\tilde{T}(\tilde{s}, \tilde{a}, \tilde{s}') = T(s, a, s')$ if $|\mathbf{G}' \cap \mathbf{G}''| > 1$ and equals 0 otherwise.
- The cost function $\tilde{C} : \tilde{S} \times \tilde{A} \times \tilde{S} \to \mathbb{R}^+$ gives the cost $\tilde{C}(\tilde{s}, \tilde{a}, \tilde{s}')$ of executing action $\tilde{a}$ in augmented state $\tilde{s}$ and arriving in $\tilde{s}'$. This cost equals the cost $\tilde{C}(\tilde{s}, \tilde{a}, \tilde{s}') = C(s, a, s')$ under the same conditions as above.
- The augmented goal states $\tilde{\mathbf{G}} \subseteq \tilde{\mathbf{S}}$ are those augmented states $\langle s, \mathbf{G}' \rangle$ for which any execution of an augmented action will transition to an augmented state $\langle s', \mathbf{G}''' \rangle$ with one goal or no goals (*i.e.*, $|\mathbf{G}'''| \leq 1$) in the regular MDP.

S-GRD algorithms use augmented MDPs and VI-like algorithms to compute the *wcd* by finding the maximum expected cost from the augmented starting state to any augmented goal.

Partially-Observable S-GRD (POS-GRD)

A key assumption in S-GRDs is that the actions of the agents are observable. However, in applications such as those involving agent navigation, agent actions are not observed and only his state (current location) is available. Further, while in deterministic settings (Assumption 3), one can accurately infer the action of an agent by continuously observing his state, this does no longer hold in S-GRDs. Therefore, the S-GRD algorithms proposed thus far cannot be used off-the-shelf directly.

Towards this end, we define the *Partially-Observable S-GRD* (POS-GRD) problem, where (1) only states (rather than actions) can be observed; and (2) states are partially observable, so that several states are indistinguishable from one another. The degree of observation uncertainty is defined by the resolution of sensors in the problem.

We follow Keren *et al.* [2018] by modeling a POS-GRD

problem with two components: A *goal recognition model* that describes the goal recognition problem and a *design model* that specifies the possible ways one can modify the goal recognition model. We formulate each of these components separately before integrating them into a POS-GRD model.

Definition 1 (Goal Recognition Model) *A partially-observable goal recognition (POS-GRD) model with stochastic action outcomes is represented as a tuple* $PO = \langle D, \mathbf{G}, N \rangle$ *where*

- $D = \langle \mathbf{S}, s_0, \mathbf{A}, \mathbf{T}, \mathbf{C} \rangle$ *captures the domain information;*
- $\mathbf{G}$ *is a set of possible goals; and*
- N *represents a sensor function that partitions* $\mathbf{S}$ *into observation sets* $\mathbf{O}_1^N, ..., \mathbf{O}_n^N$*, where* $\forall s_i, s_j \in \mathbf{S}, s_i \neq s_j : s_i, s_j \in \mathbf{O}_k^N \iff N(s_i) = N(s_j)$*. Each set* $\mathbf{O}_k^N$ *corresponds to a different observation and we refer to* $N(s)$ *as the projected observation of* s*.*

The above model generalizes the stochastic goal recognition setting proposed by Wayllace *et al.* [2016] by including a sensor model N that defines the degree of partial observability of the states in the problem. If all the states are fully observable, then observation set $\mathbf{O}_i$ is composed of *exactly* one state.

In this paper, we focus on two types of possible modifications in the design model $\mathcal{D}$:

- ACTION REMOVAL: A modification that removes some actions from the set of applicable actions in the model, and
- SENSOR REFINEMENT: A modification that allows the observer to distinguish between two states that have the same observation.

Definition 2 *[refinement] A sensor model* N' *is a* refinement *of sensor model* N *if there exists a set* $\mathbf{O}_j^N$ *such that* $\mathbf{O}_i^{N'} \subseteq \mathbf{O}_j^N$ *for each observation* $\mathbf{O}_i^{N'}$ *of* N'*.*

We let PO^m represent the POS-GRD model that results from applying m to PO and let N^m and N denote the sensor models of PO^m and PO, respectively. We define sensor refinement as follows.

Definition 3 *[sensor refinement] A modification* m *is a* sensor refinement *modification if for every goal recognition model* PO*,* PO^m *is identical to* PO *except that* N^m *is a refinement of* N*.*

Note that as opposed to the sensor refinement suggested by Keren *et al.* [2016b], where the sensor model is defined over tokens emitted by performed actions, sensor refinement defined here applies to settings where the state of the agent may be only partially observed and the observer has a way to improve its observability by sensing features of the environment.

Definition 4 *A partially-observable goal recognition design (POS-GRD) problem is given by the pair* $T = \langle PO_0, \mathcal{D} \rangle$*, where*

- PO_0 *is an initial goal recognition model, and*
- $\mathcal{D}$ *is a* design model*.*

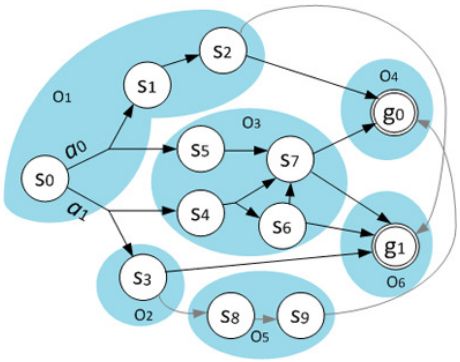


Figure 2: Example Illustration

Due to low sensor resolution, more than one state can map to the same observation. We represent this fact by grouping states with the same observation together. Figure 2 illustrates the states and their observations, where each shaded area represents one observation. If the agent moves between states with the same observation, it is impossible for the observer to know whether the agent stayed in its current state or moved to a different state with the same observation. If the agent moves to a state that has a different observation, the observer may gain some information. To do so, the observer needs to keep track of the set of possible goals given the sequence of observations observed. Wayllace *et al.* [2017] used an augmented MDP to keep track of the history of states of the agent and to discard some possible goals along the path. To solve a POS-GRD problem, we make use of a similar augmented MDP structure, where we modify it to account for partial observability. Below, we provide new definitions that will be useful to explain the construction of the new augmented MDP.

Definition 5 (Starting State) *State* s *is a starting state if* $\exists s' \in \mathbf{S}, a \in \mathbf{A} : \mathbf{T}\langle s', a, s \rangle > 0 \wedge N(s) \neq N(s') \vee s = s_0$*, where* s_0 *is the initial state.*

Definition 6 (Set of Connected States) *Given a starting state* s_i*, the set of connected states* $\mathbf{C}_i$ *of* s_i *is the set of all states that are reachable from* s_i *and whose observation is the same as* s_i*. More precisely,* $\mathbf{C}_i = \{s_i\} \cup \{s_j \mid \mathbf{T}(s_i, \cdot, s_j) > 0 \wedge N(s_j) = N(S_i)\}$*.*

Definition 7 (Connected Observation) *Given a set of starting states* $\mathbf{S}'$*, its connected observation is* $\tilde{\mathbf{O}}(\mathbf{S}') = \cup_{s_i \in \mathbf{S}'} \mathbf{C}_i$*, where* $\mathbf{C}_i$ *is the set of connected states of starting state* s_i*.*

Definition 8 (Ending State) *Given a connected observation* $\mathbf{O}(\mathbf{S}')$*, state* s *is an ending state of* $\mathbf{O}(\mathbf{S}')$ *if* $s \in \mathbf{O}(\mathbf{S}') \wedge \exists a \in \mathbf{A}, s' \notin \mathbf{O}(\mathbf{S}') : \mathbf{T}\langle s, a, s' \rangle > 0$*.*

To demonstrate, in Figure 2, the *starting states* are $s_0, s_3, s_4, s_5, s_8, g_0$, and g_1; the set of *connected states* of s_0 is $\mathbf{C}_0 = \{s_0, s_1, s_2\}$ and the set of *connected states* of s_4 is $\mathbf{C}_4 = \{s_4, s_6, s_7\}$; the *connected observation* of $\tilde{\mathbf{O}}(\{s_4, s_5\}) = \{s_4, s_5, s_6, s_7\}$; and the *ending states* are $s_2, s_3, s_6, s_7, s_9, g_0$, and g_1. Note that a state can be starting and ending state at the same time.

Definition 9 (Predecessor) *Given two connected observations $\tilde{\mathbf{O}}(\mathbf{S}')$ and $\tilde{\mathbf{O}}(\mathbf{S}'')$, $\tilde{\mathbf{O}}(\mathbf{S}')$ is the predecessor of $\tilde{\mathbf{O}}(\mathbf{S}'')$ if $\exists a \in \mathbf{A}, s \in \tilde{\mathbf{O}}(\mathbf{S}'), s' \in \tilde{\mathbf{O}}(\mathbf{S}'') : \mathbf{T}\langle s, a, s'\rangle > 0 \wedge N(s) \neq N(s')$.*

In Figure 2, $O_1 = \tilde{\mathbf{O}}(\{\mathbf{s_0}\})$ is a *predecessor* of O_2, O_3, O_4, and O_6.

To construct the augmented MDP for a POS-GRD problem, we start by augmenting states, actions, and transitions from the original MDP $\langle \mathbf{S}, s_0, \mathbf{A}, \mathbf{T}, \mathbf{C}, \mathbf{G}\rangle$, that is, we generate $\langle \tilde{\mathbf{S}}, \tilde{s}_0, \tilde{\mathbf{A}}, \tilde{\mathbf{T}}\rangle$ following the procedure described by Wayllace *et al.* [2017]. It is important to point out that when a state s is augmented to a state $\tilde{s} = \langle s, \mathbf{G}'\rangle$, $\tilde{s}$ projects the same observation as s, that is, $N(s) = N(\langle s, \mathbf{G}'\rangle)$ for any set $\mathbf{G}' \subseteq \mathbf{G}$.

Augmented MDPs for POS-GRD

For example, for the original MDP shown in Figure 3(a), the corresponding augmented states and actions are shown in Figure 3(b). Note that all the augmented states generated from the same state (e.g., s_2 or s_4) have the same observation. Every state is augmented with the set of possible goals for that state and every action is augmented with the set of goals for which that action is optimal (non-optimal actions in gray are not taken into account); the set of possible goals of a successor is found by intersecting the set of possible goals of its predecessor with the set of possible goals of the action executed to transition to that state.

If all states and actions were observable in the example depicted in Figure 3, the agent would reveal its goal as soon as one action is executed (since a_1 is an optimal action only for goal g_1 and a_0 is optimal only for goal g_0). However, in our partially-observable setting, where actions are not observable and all connected states have the same observation, the agent is able to hide its true goal for longer. For example, in Figure 3(b), when the observer observes $\langle O_1, O_2\rangle$, it could be due to the agent executing action a_1 from state s_0 and transitioning to state s_2 or it could be due to the agent executing action a_0 and transitioning to the same state s_2. Even though the two actions are optimal actions for two different goals g_1 and g_2, the observer is not able to distinguish between them. As a result, the agent is able to hide its true goal after executing either of those two actions if it transitions to state s_2. However, if the observer observes $\langle O_1, O_4\rangle$, then it knows with certainty that the agent executed action a_1 and can infer that the agent's goal is g_1.

To do this type of inference, we keep track of the set of possible goals with respect to the projected observations by extending Definitions 5 to 9 to the augmented domain. The extensions are trivial – in every definition, it is sufficient to substitute any reference to a state, action, or transition with their augmented counterpart. For instance, Definition 5 is extended as follows:

Definition 10 (Augmented Starting State) *Augmented state $\tilde{s}$ is an augmented starting state if $\exists \tilde{s}' \in \tilde{\mathbf{S}}, \tilde{a} \in \tilde{\mathbf{A}} : \mathbf{T}\langle \tilde{s}', \tilde{a}, \tilde{s}\rangle > 0 \wedge N(\tilde{s}) \neq N(\tilde{s}') \vee \tilde{s} = \tilde{s}_0$, where $\tilde{s}_0$ is the augmented initial state.*

Definition 11 (Augmented Observation) *An augmented observation is a tuple $\langle \tilde{\mathbf{O}}(\tilde{\mathbf{S}}'), \mathbf{G}'\rangle$ where $\tilde{\mathbf{O}}(\tilde{\mathbf{S}}')$ is a connected observation and $\mathbf{G}' = \bigcup_{\langle s, \mathbf{G}''\rangle \in \tilde{\mathbf{S}}'} \mathbf{G}''$.*

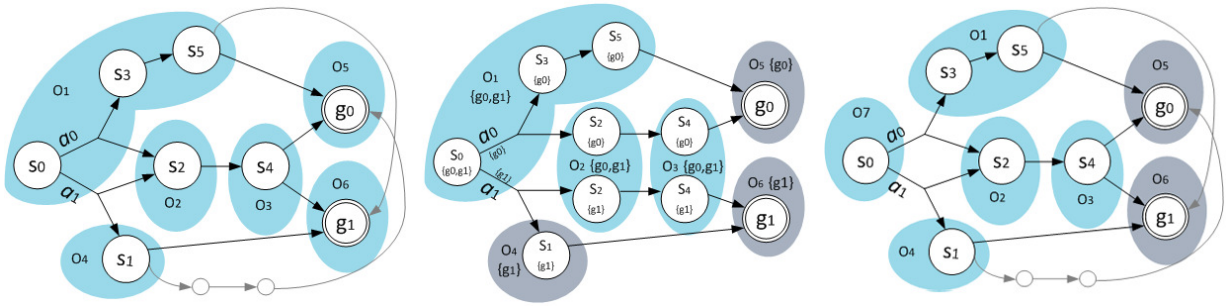
Once the tuple $\langle \tilde{\mathbf{S}}, \tilde{s}_0, \tilde{\mathbf{A}}, \tilde{\mathbf{T}}\rangle$ is built, a structure of augmented observation sets (**AOS**) is constructed following Algorithm 1. This structure implements the model represented by the shaded regions in Figure 3(b). Specifically, each shaded region corresponds to one augmented observation and the predecessor for any augmented observation in **AOS** can be easily found.

Two connected observations can have states projecting the same observation. For example, given $\tilde{\mathbf{O}}(\tilde{\mathbf{S}}')$ and $\tilde{\mathbf{O}}(\tilde{\mathbf{S}}'')$, if $\tilde{\mathbf{S}}'' \cap \tilde{\mathbf{S}}' \neq \emptyset$, then by Definition 7, $\forall \tilde{s_i} \in \tilde{\mathbf{S}}', \tilde{s_j} \in \tilde{\mathbf{S}}'' : N(\tilde{s_i}) = N(\tilde{s_j})$. If their augmented observations are $\langle \tilde{\mathbf{O}}(\tilde{\mathbf{S}}'), \mathbf{G}'\rangle$ and $\langle \tilde{\mathbf{O}}(\tilde{\mathbf{S}}''), \mathbf{G}''\rangle$ respectively, and $\mathbf{G}' \neq \mathbf{G}''$, then both augmented observations are considered different. Since one state should belong to only one augmented observation, we need to create duplicates from all common augmented states and modify their transition functions accordingly. The function $Create$ (line 13) will create a new augmented state $\tilde{s}'$ every time that an explored state $\tilde{s}$ belongs to another augmented observation.

Once Algorithm 1 is executed, the augmented observations in **AOS** contain the information that is available to the observer. Therefore, we can now use it to define a new augmented MDP that will allow us to correctly solve the initial goal recognition model (Definition 4).

Definition 12 (Augmented MDP for POS-GRD) *For a POS-GRD problem $PO = \langle D, \mathbf{G}, N\rangle$ with domain information $D = \langle \mathbf{S}, s_0, \mathbf{A}, \mathbf{T}, \mathbf{C}\rangle$ and the set of augmented observations **AOS** built following Algorithm 1, an augmented MDP is defined by a tuple $\langle \tilde{\mathbf{S}}, \tilde{s}_0, \tilde{\mathbf{A}}, \tilde{\mathbf{T}}, \tilde{\mathbf{C}}, \tilde{\mathbf{G}}\rangle$ that consists of the following:*

- *a set $\tilde{\mathbf{S}}$ of augmented states $\tilde{s}$, where $\tilde{s} \in \tilde{\mathbf{S}} \iff \forall \langle \tilde{\mathbf{O}}(\tilde{\mathbf{S}}'), \mathbf{G}'\rangle \in \mathbf{AOS} : \tilde{s} \in \tilde{\mathbf{O}}(\tilde{\mathbf{S}}')$;*
- *an augmented start state $\tilde{s_0} = \langle s_0, \mathbf{G}\rangle$;*
- *a set of augmented actions $\tilde{\mathbf{A}} = \langle a, \mathbf{G}''\rangle$, where $\mathbf{G}''$ is the set of all goals for which $a \in \mathbf{A}$ is an optimal action;*
- *a transition function $\tilde{\mathbf{T}} : \tilde{\mathbf{S}} \times \tilde{\mathbf{A}} \times \tilde{\mathbf{S}} \to [0, 1]$ that gives the probability $\tilde{T}(\langle s, \mathbf{G}'\rangle, \langle a, \mathbf{G}''\rangle, \langle s', \mathbf{G}' \cap \mathbf{G}''\rangle)$ of transitioning from augmented state $\langle s, \mathbf{G}'\rangle$ to augmented state $\langle s', \mathbf{G}' \cap \mathbf{G}''\rangle$ when augmented action $\langle a, \mathbf{G}''\rangle$ is executed; this probability equals $T(s, a, s')$ if $|\mathbf{G}' \cap \mathbf{G}''| > 1$ and equals 0 otherwise;*
- *a cost function $\tilde{\mathbf{C}} : \tilde{\mathbf{S}} \times \tilde{\mathbf{A}} \times \tilde{\mathbf{S}} \to \mathbb{R}^+$ that gives cost $\tilde{C}(\tilde{s}, \tilde{a}, \tilde{s}') = 0$ if $\exists \langle \tilde{\mathbf{O}}(\tilde{\mathbf{S}}'), \mathbf{G}^o\rangle \in \mathbf{AOS} : \tilde{s}' \in \tilde{\mathbf{O}}(\tilde{\mathbf{S}}') \wedge |\mathbf{G}^o| \leq 1$ and $\tilde{C}(\tilde{s}, \tilde{a}, \tilde{s}') = C(s, a, s')$ otherwise; and*
- *the set of augmented goals $\tilde{\mathbf{G}} \subset \tilde{\mathbf{S}} = \{\tilde{s} \mid \forall \langle \tilde{\mathbf{O}}(\tilde{\mathbf{S}}'), \mathbf{G}^o\rangle \in \mathbf{AOS} : |\mathbf{G}^o| = 1 \wedge \tilde{s} \in \mathbf{S}'\}$*



(a) Original Example (b) Augmented Observation Sets (c) After Sensor Refinement

Figure 3: Example

Algorithm 1: CONSTRUCTION OF THE AUGMENTED STATE SPACE

Input: $\langle \tilde{\mathbf{S}}, \tilde{s}_0, \tilde{\mathbf{A}}, \tilde{\mathbf{T}} \rangle$
$\mathbf{AOS} \leftarrow \{\langle \tilde{\mathbf{O}}(\{\tilde{\mathbf{s}}_0\}), \mathbf{G} \rangle\}$
$Q \leftarrow \tilde{\mathbf{O}}(\{\tilde{\mathbf{s}}_0\})$
while $Q \neq \emptyset$ **do**
  $\tilde{\mathbf{O}}(\tilde{\mathbf{S}}_i) \leftarrow$ Q.dequeue()
  Find all ending states $s_e \in \tilde{\mathbf{O}}(\tilde{\mathbf{S}}_i)$
  foreach $s_e \in \tilde{\mathbf{O}}(\tilde{\mathbf{S}}_i)$ **do**
    if $\exists \tilde{a} \in \tilde{\mathbf{A}}$ **then**
      Find all starting states $\tilde{s}_s$ with $\mathbf{T}(\tilde{s}_e, \tilde{a}, \tilde{s}_s) > 0 \wedge N(\tilde{s}_e) \neq N(\tilde{s}_s)$
      Group all $\tilde{s}_s$ with same $N(\tilde{s}_s)$
    foreach *group* $\mathbf{S}'$ **do**
      if $\nexists \langle \tilde{\mathbf{O}}(\tilde{\mathbf{S}}'), \mathbf{G}' \rangle \in \mathbf{AOS} : \mathbf{G}' = \bigcup_{\langle s_s, \mathbf{G}'' \rangle \in \tilde{\mathbf{S}}'} \mathbf{G}''$ **then**
        $\langle \tilde{\mathbf{O}}(\tilde{\mathbf{S}}'), \mathbf{G}' \rangle \leftarrow$ Create $(\{\tilde{s}_s | \tilde{s}_s \in \tilde{\mathbf{S}}'\}, \mathbf{G}', \mathbf{AOS})$
        if $\nexists \langle \tilde{\mathbf{O}}(\tilde{\mathbf{S}}''), \mathbf{G}' \rangle \in \mathbf{AOS} : \tilde{\mathbf{S}}' \neq \tilde{\mathbf{S}}'' \wedge \tilde{\mathbf{O}}(\tilde{\mathbf{S}}') = \tilde{\mathbf{O}}(\tilde{\mathbf{S}}'')$ **then**
          $\mathbf{AOS} \leftarrow \mathbf{AOS} \cup \langle \tilde{\mathbf{O}}(\tilde{\mathbf{S}}'), \mathbf{G}' \rangle$
          Enqueue $\langle \tilde{\mathbf{O}}(\tilde{\mathbf{S}}'), \mathbf{G}' \rangle$ in Q

Function Create$(\tilde{\mathbf{S}}_s, \mathbf{G}', \mathbf{AOS})$

$\tilde{\mathbf{S}}'_s \leftarrow \emptyset$
foreach $\tilde{s} \in \tilde{\mathbf{S}}_s$ **do**
  if $\exists \tilde{\mathbf{O}}(\tilde{\mathbf{S}}) \in \mathbf{AOS} : \tilde{s} \in \tilde{\mathbf{O}}(\tilde{\mathbf{S}})$ **then**
    $\tilde{s}' \leftarrow newState(\tilde{s})$
    $\tilde{\mathbf{S}}'_s \leftarrow \tilde{\mathbf{S}}'_s \cup \tilde{s}'$
    $\tilde{\mathbf{S}} \leftarrow \tilde{\mathbf{S}} \cup \tilde{s}'$
  else
    $\tilde{\mathbf{S}}'_s \leftarrow \tilde{\mathbf{S}}'_s \cup \tilde{s}$
$\tilde{\mathbf{O}}(\tilde{\mathbf{S}}') \leftarrow connectedObs(\tilde{\mathbf{S}}')$
return $\langle \tilde{\mathbf{O}}(\tilde{\mathbf{S}}'), \mathbf{G}' \rangle \leftarrow augmentedObs(\tilde{\mathbf{O}}(\tilde{\mathbf{S}}'), \mathbf{G}')$

Computing the *wcd*

Definition 13 (*wcd*) *The wcd of a POS-GRD problem PO is defined as:*

$$wcd(PO) = \max_{\tilde{\pi} \in \tilde{\mathbf{\Pi}}} V_{\tilde{\pi}}(\tilde{s}_0) \tag{2}$$

$$V_{\tilde{\pi}}(\tilde{s}) = \sum_{\tilde{s}' \in \tilde{\mathbf{S}}} \tilde{T}(\tilde{s}, \tilde{\pi}(\tilde{s}), \tilde{s}') \left[\tilde{C}(\tilde{s}, \tilde{\pi}(\tilde{s}), \tilde{s}') + V_{\tilde{\pi}}(\tilde{s}') \right] \tag{3}$$

where $\tilde{\mathbf{\Pi}}$ is the set of augmented policies in the augmented MDP as specified in Definition 12 and $V_{\tilde{\pi}}(\tilde{s}_0)$ is the expected cost for s_0 with augmented policy $\tilde{\pi}$ computed recursively using Equation 3.

The baseline algorithm to compute the *wcd* is to follow Equations 2 and 3, that is, for each possible augmented policy $\tilde{\pi}$, run a VI-like algorithm, where instead of using the Bellman equation (Equation 1) we use Equation 3, run the algorithm until convergence, and store the value $V_{\tilde{\pi}}(\tilde{s}_0)$ to

find the maximum among all policies. Finding the maximum expected cost should be done in one execution of the algorithm using the following equation:

$$V^*(\tilde{s}) = \max_{\tilde{a} \in \tilde{A}} \sum_{\tilde{s}' \in \tilde{\mathbf{S}}} \tilde{T}(\tilde{s}, \tilde{a}, \tilde{s}') \big[\tilde{C}(\tilde{s}, \tilde{a}, \tilde{s}') + V^*(\tilde{s}') \big] \quad (4)$$

Observe that this equation is equivalent to Equations 2 and 3, and it differs from the Bellman equation only in the operator: this one uses the maximization instead of minimization. The main problem of maximizing costs is the existence of infinite loops since the optimal policy is to accumulate cost infinitely. The augmented MDP does not have infinite loops because the augmented actions are constructed using only optimal actions to arrive to any possible goal, hence, the only case of a cycle could be if an agent transitioning from state $\tilde{s}$ to $\tilde{s}'$ executes an augmented action $\langle \tilde{a}, \mathbf{G}' \rangle$ and to transition from $\tilde{s}'$ to $\tilde{s}$ executes action $\langle \tilde{a}', \mathbf{G}'' \rangle$, where $\mathbf{G}' \cap \mathbf{G}'' = \emptyset$, that is, if both actions are optimal for a different set of possible goals. However, this is impossible because the set of possible goals of an augmented state is always a subset of the set of possible goals of its predecessors. A formal sketch proof of this property was presented by Wayllace *et al.* [2017] for the augmented MDP for S-GRD problems. The property remains true for POS-GRD since the new augmented states are duplicates of others; therefore, the possible augmented actions, successors and predecessors remain the same.

Therefore, a VI-like algorithm using Equation 4 can be used. Even further, we took advantage of the structure of the augmented MDP that usually allows to group augmented states into *strongly connected components* (SCC), therefore, algorithms similar to TVI can be used. The Tarjan's algorithm (Tarjan 1972) was used to form SCCs. Once the SCCs are constructed, a VI-like algorithm is executed on each SCC in reverse topological order.

It is worth mentioning that non-reachable states in the augmented MDP can be prunned, as well as states that belong to an augmented observation whose predecessors have only one possible goal.

Reducing the *wcd*

Once the wcd of the model has been computed, we propose to apply two types of modifications with the objective to reduce the *wcd*: (1) Sensor refinement and (2) Action removal.

Sensor Refinement: In a POS-GRD problem, partial observability is due to low-resolution sensors that make it impossible for an observer to distinguish a number of similar states. As a result, all the states covered by a sensor have the same observation and the observer does not know which is the actual state of the agent. By refining sensor resolution, the observer can gain better observability of the states covered by a particular sensor. The design objective is thus to identify which sensors refine such that the resulting *wcd* is minimized under the specified constraints. Ideally, all sensors should be refined so that the all states are fully observable as this will guarantee that the *wcd* is minimized. However, we assume that there is a limited budget available and only a limited number of sensors can be refined.

Algorithm 2: Sensor Refinement

```
27 create a queue Q ← combination of up to k states
28 wcd* ← ∞
29 S* ← ∅
30 while Q ≠ ∅ do
31     S_cand ← Q.deque
32     wcd_cand ← compute_wcd(S_cand)
33     if wcd_cand < wcd* then
34         wcd* ← wcd_cand
35         S* ← S_cand
36 foreach s_i ∈ S* do
37     O_i^N = getObs(s_i)
38     O_i^N ← O_i^N \ {s_i}
39     add mapping s_i → O_{n+i}^N to sensor function N
```

In this paper, we use a simple implementation of sensor refinement (as defined in Definition 3 by allowing making a single state fully observable. In other words, if the agent is in any of the *refined states*, the observer is able to observe it with full certainty. Figure 3(c) shows an example of sensor refinement for the original example in Figure 3(a), where state s_0, previously mapped to the same observation O_1 as states s_3 and s_5, is now mapped to a new observation O_7. This makes it possible to distinguish s_0 from the other two states. In our setting, there is a maximum of k sensor refinement modifications that can be performed.

Algorithm 2 describes the sensor refinement pseudocode for choosing the k states to *refine*. It first constructs a queue that contains all subsets of up to k states (line 27). Then, it iteratively evaluates each set by computing the *wcd* if those refined states (line 32). If the resulting *wcd* is smaller than the minimal *wcd* found so far, it updates the best *wcd* with that value and stores that set of states as the best set (lines 33-35). After evaluating all sets of states in the queue, it updates the sensor function N by replacing the observation of each of the k states with new observations (lines 36-39).

Action Removal: Another (more classical) modification that can be performed to the model is action removal, where there is also a constraint on the number of modifications. Specifically, the objective is to minimize the *wcd* by removing at most k actions. Similar to the algorithms previously used by Wayllace *et al.* [2017], we enumerate through all possible combinations of up to k actions, compute the *wcd* for each combination, and choose the combination that reduces the *wcd* the most.

Empirical Evaluation

The domain used to run the experiments is a modification of the domain called ROOM, which was used in the Non-Deterministic Track of the 2006 ICAPS International Plan-

ning Competition.[1] It is a grid world where the actions as well as the transition probabilities are defined individually for each state. Each instance of this domain is defined by the x- and y-dimensions of the room and the number of possible goals. The initial setting for partial observability was added, specifically, four contiguous states were mapped to the same observation.

Three type of experiments were performed: (1) Partial observability with sensor refinement (SR); (2) Partial observability with action removal (AR); and (3) Full observability of states with action removal (FO), (note that this is different to S-GRD since the actions are non-observable here). The smaller instances used a budget k=2 in all settings while the larger ones used k=1; they timed out with k=2. The parameter k represents the maximum number of states to refine in SR and the maximum number of actions to remove in AR and FO. Both SR and AR start with the same initial partially-observable problem. The only difference is in the type of modifications allowed. FO assumes that all states are fully observable and only the actions are hidden to the observer. The experiments were conducted on a 3.1 GHz Intel Core i7 with 16 GB of RAM and a timeout of two days was imposed.

We make the following observations:

- The initial *wcd* is larger for all the instances with partially-observable states (SR and AR) compared to if they are fully observable (FO). However, the ratio between both values differs across instances, which is interesting because it shows that not only the resolution (number of states covered by one sensor), but also the placement (which states are covered by the same sensor) of sensors matters. Thus, in the future, we plan to conduct additional experiments to optimize sensor placement, even without improving their resolution.
- The *wcd* reduced for all instances when we applied sensor refinement. However, the *wcd* reduced for only one instance when we applied action removal (two others also show some reduction, but it might be due to rounding errors). This is because the domain in general has few policies that are common to more than one goal and removing actions increases the initial expected cost or causes the goal to become unreachable.
- In SR, only two instances were able to match the *wcd* value of FO after reduction, but four other instances were close. This does not depend on the size of the state space (one match occurred for instance 4-4-3 and the other for 32-32-3), which also suggests that the initial sensor mapping could affect the quality of goal recognition.
- The running time grows exponentially with the size of the reachable state space and, as expected, with the number of modifications k.

Conclusions and Future Work

Previous work in GRD did not account for partially-observable states, which is relevant to many applications such as agent navigation, where only the current state is observable, not the intention of movement. Additionally, observations depend on the resolution of the sensor. Thus, some states can be perceived as identical to other states. In response to these observations, this paper proposes the *Partially Observable S-GRD* (POS-GRD) problem where (1) actions are not observable and (2) states are partially observable. New algorithms taking partial observability into account to compute the *wcd* and to perform sensor refinement in POS-GRD problems were proposed. Experimental results show that sensor refinement always reduces the *wcd* and suggest that the initial sensor configuration affects the reduction ratio when the number of possible modifications is limited.

Future work includes the use of heuristics to prune the search space for higher values of k. Since the modifications start from 1 to k, the idea is to find all augmented states that have less or equal expected cost than the current (minimized) value of *wcd* and prune the rest of the searching space. We are also interested in use other metrics and design mechanisms in the POS-GRD context.

[1] http://idm-lab.org/wiki/icaps/ipc2006/probabilistic/

Acknowledgments

This research is partially supported by NSF grant 1540168. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations, agencies, or the U.S. government.

References

Richard Bellman. *Dynamic Programming*. Princeton University Press, 1957.

Sandra Carberry. Techniques for plan recognition. *User Modeling and User-Adapted Interaction*, 11:31–48, 2001.

Peng Dai, Mausam, Daniel S Weld, and Judy Goldsmith. Topological value iteration algorithms. *Journal of Artificial Intelligence Research*, 42:181–209, 2011.

Sarah Keren, Avigdor Gal, and Erez Karpas. Goal recognition design. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pages 154–162, 2014.

Sarah Keren, Avigdor Gal, and Erez Karpas. Goal recognition design for non-optimal agents. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 3298–3304, 2015.

Sarah Keren, Avigdor Gal, and Erez Karpas. Goal recognition design with non-observable actions. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 3152–3158, 2016.

Sarah Keren, Avigdor Gal, and Erez Karpas. Privacy preserving plans in partially observable environments. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 3170–3176, 2016.

Sarah Keren, Avigdor Gal, and Erez Karpas. Strong stubborn sets for efficient goal recognition design. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 2018.

Domain Instances	k	Sensor Refinement (SR)		Action Removal (AR)		Full State Observability (FO)	
		wcd Reduction	Runtime (s)	*wcd* Reduction	Runtime (s)	*wcd* Reduction	Runtime (s)
4-4-3	2	4.98 → 3.71	0.25	4.98 → 4.98	0.18	3.71 → 3.71	0.26
8-8-2	2	16.05 → 16.03	19.76	16.05 → 16.05	23.48	16.02 → 16.02	22.65
8-8-3	2	10.64 → 9.22	51.78	10.64 → 10.64	27.96	9.09 → 9.09	13.18
12-12-3	2	16.67 → 16.53	231.48	16.67 → 16.67	102.05	16.42 → 16.42	82.26
16-16-3	2	16.71 → 8.16	3,238.08	16.71 → 16.71	1,538.90	6.62 → 6.62	955.53
20-20-3	2	53.42 → 40.27	14,595.43	53.42 → 53.33	28,649.10	38.59 → 38.59	5,845.37
24-24-3	2	19.28 → 12.61	53,846.64	19.28 → 19.28	8,322.37	12.15 → 12.15	1,948.80
32-32-2	1	79.67 → 48.17	395.39	79.67 → 79.50	481.14	39.28 → 38.91	105.14
32-32-3	1	87.57 → 86.57	455.83	87.57 → 87.57	632.53	86.57 → 86.57	145.22
44-44-3	1	92.74 → 87.21	1,519.19	39.28 → 39.28	13.18	73.76 → 73.76	1,116.03

Table 2: Experimental Results

Mausam and Andrey Kolobov. *Planning with Markov Decision Processes: An AI Perspective*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2012.

Miquel Ramírez and Hector Geffner. Probabilistic plan recognition using off-the-shelf classical planners. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2010.

Tran Cao Son, Orkunt Sabuncu, Christian Schulz-Hanke, Torsten Schaub, and William Yeoh. Solving goal recognition design using ASP. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 3181–3187, 2016.

Gita Sukthankar, Christopher Geib, Hung Hai Bui, David Pynadath, and Robert P Goldman. *Plan, activity, and intent recognition: Theory and practice*. Newnes, 2014.

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

Christabel Wayllace, Ping Hou, William Yeoh, and Tran Cao Son. Goal recognition design with stochastic agent action outcomes. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 3279–3285, 2016.

Christabel Wayllace, Ping Hou, and William Yeoh. New metrics and algorithms for stochastic goal recognition design problems. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 4455–4462, 2017.

Unchaining the Power of Partial Delete Relaxation, Part II: Finding Plans with Red-Black State Space Search

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

Abstract

Red-black relaxation in classical planning allows to interpolate between delete-relaxed and real planning. Yet the traditional use of relaxations to generate heuristics restricts relaxation usage to tractable fragments. How to actually tap into the red-black relaxation's interpolation power? Prior work has devised *red-black state space search (RBS)* for intractable red-black planning, and has explored two uses: proving unsolvability, generating seed plans for plan repair. Here, we explore the generation of plans directly through RBS. We design two enhancements to this end: (A) use a known tractable fragment where possible, use RBS for the intractable parts; (B) check RBS state transitions for *realizability*, spawn relaxation refinements where the check fails. We show the potential merits of both techniques on IPC benchmarks.

Introduction

Relaxations are prominently used in AI Planning for the generation of heuristic functions (e. g. (Bonet and Geffner 2001; Hoffmann and Nebel 2001; Helmert and Domshlak 2009; Helmert et al. 2014)). The *delete relaxation* in particular has been highly influential. Under this relaxation, state variables accumulate their values rather than switching between them.

The delete relaxation cannot account for *having to move to-and-fro*, and it ignores *resource consumption*. Hence there is a lot of work on taking some deletes into account (e. g. (Fox and Long 2001; Helmert and Geffner 2008; Haslum 2012; Coles et al. 2013; Keyder, Hoffmann, and Haslum 2014)). Here we consider *red-black planning* (Domshlak, Hoffmann, and Katz 2015), a *partial delete relaxation* method that allows to force delete-relaxed plans to behave like real plans in the limit. A subset of ("red") variables take the delete-relaxed semantics, accumulating values, while the remaining ("black") ones retain the true semantics.

The partition into red and black variables is called a *painting*, and its choice obviously allows to interpolate between delete-relaxed and real planning. Yet for use as a heuristic function, the painting must be chosen so that red-black plan generation is tractable. Prior work therefore restricts the black variables to what we will refer to as ACI, with *acyclic* causal-graph dependencies and *invertible* value-transitions.

Acyclic dependencies and invertible value-transitions occur only in small parts of practical planning tasks, so ACI is typically very far from real planning. How can we actually tap into the interpolation power of red-black planning?

We follow up on prior work on this question (Gnad et al. 2016) (*Gnad16* in what follows). Gnad16 have shown how to generate red-black plans for arbitrary paintings, via *red-black state space search (RBS)*, a hybrid of forward search and delete-relaxed planning, where every transition contains a local delete-relaxed planning step over the red variables. Gnad16 explored 1) the generation of red-black seed plans for plan repair with LPG (Gerevini, Saetti, and Serina 2003; Fox et al. 2006); and 2) proving planning tasks unsolvable within the red-black relaxation, via an iteration of more and more refined RBS searches (more and more black variables).

Here, we explore the use of RBS for generating plans. This is the natural complement of 2), in what we envision as a red-black relaxation refinement process. The challenge is to make RBS produce real plans early on, with few black variables. We design two enhancements to this end:

A) We create synergy between RBS and ACI, by replacing delete-relaxed planning with ACI planning in RBS. This uses ACI where possible (e. g., moving to-and-fro on an invertible road map), and uses RBS where not (e. g., non-invertible resource consumption). We identify a maximally permissive condition on the black-variable dependencies under which this combination is possible.

B) We design an adaptive variant of refinement, locally within a single RBS search space where needed. We check every transition $s \xrightarrow{a} s'$ for *realizability* of the red parts, i. e., whether the delete-relaxed plan here works in reality. Non-realizable transitions are pruned, and spawn *refinement options*: red-black planning tasks starting at s, with additional black variables addressing the non-realizability of $s \xrightarrow{a} s'$. The refinement options become search nodes in an overall heuristic search.

We evaluate our techniques on the IPC benchmarks. In overall performance, A) is competitive, while B) often suffers from too many refinement options. Compared to Gnad16's approach 1), A) is better overall, and both A) and B) are highly complementary to 1) per domain. In five domains, our best configurations outperform the state-of-the-art systems LAMA and Mercury by large margins.

Preliminaries

We use the *finite-domain representation (FDR)* framework (Bäckström and Nebel 1995; Helmert 2009). An FDR **planning task** is a tuple $\Pi = (V, A, I, G)$. V is a set of **variables** v, each with a finite domain D_v. A complete assignment to V is a **state**. I is the **initial state**, and the **goal** G is a partial assignment to V. A is a finite set of **actions**, where each $a \in A$ is a triple $(\mathsf{pre}_a, \mathsf{eff}_a, c_a)$. The **precondition** pre_a and the **effect** eff_a are partial assignments to V; $c_a \in \mathbb{R}_0^+$ is the action's **cost**. We will sometimes refer to variable-value pairs $v = d$ as **facts**. For a partial assignment p, $\mathcal{V}(p)$ denotes the set of variables instantiated by p. For $V' \subseteq \mathcal{V}(p)$, by $p[V'] := p|_{V'}$ we denote the restriction of p to V'. We say that an action a is **applicable** in a state s if $s[\mathcal{V}(\mathsf{pre}_a)] = \mathsf{pre}_a$. The outcome state $s[\![a]\!]$ is like s except that $s[\![a]\!](v) = \mathsf{eff}_a(v)$ for each $v \in \mathcal{V}(\mathsf{eff}_a)$.

A **transition system** is a tuple $\Theta = (S, L, T, s_0, S_G)$. S is a set of states. L is a set of *labels*. $T \subseteq S \times L \times S$ is a set of **transitions**. $s_0 \in S$ is the **start state** and $S_G \subseteq S$ is the set of **goal states**. A **plan** for a state s is a transition path from s to a state in S_G. The **state space** of Π is the transition system Θ_Π where S is the set of states in Π, $L = A$, $(s, a, s') \in T$ iff a is applicable in s and $s' = s[\![a]\!]$, $s_0 = I$, and $s \in S_G$ if $s[\mathcal{V}(G)] = G$. A plan π for I in Θ_Π is called a **plan** for Π.

The **causal graph** (e. g. (Jonsson and Bäckström 1995; Helmert 2006)) is a digraph with vertices V and an arc (v, v') if $v \neq v'$ and there exists an action $a \in A$ such that $(v, v') \in [\mathcal{V}(\mathsf{eff}(a)) \cup \mathcal{V}(\mathsf{pre}(a))] \times \mathcal{V}(\mathsf{eff}(a))$.

Red-Black Planning

We next give an overview of red-black planning and associated techniques, as needed to understand our contribution.

Definitions

A **red-black planning task**, or **RB task**, is a tuple $\Pi^{\mathsf{RB}} = (V^{\mathsf{B}}, V^{\mathsf{R}}, A, I, G)$ with $V^{\mathsf{B}} \cap V^{\mathsf{R}} = \emptyset$, where $\Pi := (V, A, I, G)$ is an FDR task with $V := V^{\mathsf{B}} \cup V^{\mathsf{R}}$. V^{B} is the set of **black variables**, V^{R} is the set of **red variables**. States are now **RB states** s^{RB}, which map each variable v to a subset of its domain, $s^{\mathsf{RB}}(v) \subseteq D_v$, where $|s^{\mathsf{RB}}(v)| = 1$ for $v \in V^{\mathsf{B}}$. In the **RB initial state** s_0^{RB} each variable v is mapped to $\{I(v)\}$. **RB goal states** are those s^{RB} where $G(v) \in s^{\mathsf{RB}}(v)$ for all $v \in \mathcal{V}(G)$. An action a is applicable in an RB state s^{RB} if $\mathsf{pre}_a(v) \in s^{\mathsf{RB}}(v)$ for all $v \in \mathcal{V}(\mathsf{pre}_a)$. Upon executing a in s^{RB}, $v \in \mathcal{V}(\mathsf{eff}_a) \cap V^{\mathsf{B}}$ is set to $\{\mathsf{eff}_a(v)\}$, and $v \in \mathcal{V}(\mathsf{eff}_a) \cap V^{\mathsf{R}}$ is set to $s^{\mathsf{RB}}(v) \cup \{\mathsf{eff}_a(v)\}$. The outcome state is denoted $s^{\mathsf{RB}}[\![a]\!]$. A plan π^{RB} under this semantics is an **RB plan** for Π^{RB}. We also refer to π^{RB} as an RB plan for Π, viewing Π^{RB} as a **red-black relaxation** of Π, where the choice of V^{B} vs. V^{R} is a **painting** defining the relaxation.

The red-black relaxations of any FDR task Π form a **refinement hierarchy**, with more refined relaxations having larger sets V^{B}. At the extremes, for $V^{\mathsf{B}} = V$ we obtain real planning, and for $V^{\mathsf{B}} = \emptyset$ we obtain fully delete-relaxed planning.

Example 1. *Our example task Π is shown in Figure 1. It has variables $V = \{T, M, A, B\}$ with domains $D_T = \{l_1, l_2\}$, $D_M = \{0, 1, 2\}$, $D_A = \{0, 1\}$, $D_B = \{0, 1\}$. T encodes a traveling agent with two locations l_1 and l_2, initially l_2. The goal is to be at l_2, and to possess each product A and B. Each product is available at l_1 at price 1; A is also available at l_2, but at price 2. M is the available money. The actions have the form $go(l, l')$ and $buy(l, p, m)$. For example, $go(l_1, l_2)$ has precondition $\{T = l_1\}$ and effect $\{T = l_2\}$, and $buy(l_1, A, 2)$ has precondition $\{T = l_1, M = 2\}$ and effect $\{A = 1, M = 1\}$.*

*A fully delete-relaxed plan for this task has two **flaws**: 1) it does not go back from l_1 to l_2; 2) it may choose to buy A at l_2 instead of l_1, over-spending the budget. We can fix 1) by painting T black, and we can fix 2) by painting M black. In the red-black relaxation where $V^{\mathsf{B}} = \{T, M\}$ and $V^{\mathsf{R}} = \{A, B\}$, every RB plan for Π is a real plan for Π.*

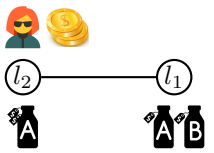


Figure 1: A simple TPP-like task.

Tractable Fragment: ACI

The initial line of work on red-black planning (Domshlak, Hoffmann, and Katz 2015), culminating in the Mercury system's success at IPC'14 (Katz and Hoffmann 2014), generates a heuristic function based on the tractable fragment **ACI**. We simplify some details in what follows, for easier exposition.

ACI requires 1) that the causal graph over the black variables is acyclic, and 2) that every black variable is **invertible**. A variable v is invertible if every value transition can be inverted under the same (or easier) conditions on other variables. An RB plan can then be generated by finding a fully delete-relaxed plan π^+, and running **ACI plan repair** on π^+ to obtain an RB plan π^{RB}. The repair process executes π^+ step-by-step under the red-black semantics; whenever a condition (precondition or goal) g on V^{B} is not satisfied, the process inserts a subsequence π achieving g. The latter is always possible, in time polynomial in the length of π: thanks to 1), V^{B} can be solved in a sequence from clients (variables which can only be modified through actions depending other variables) to servants (the dependent variables); thanks to 2), whenever a servant v must provide a value $d \in D_v$ for a client, v can reach d from its current value.[1]

Example 2. *In Example 1, T is invertible. A relaxed plan is $\pi^+ = \langle buy(l_2, A, 2), go(l_2, l_1), buy(l_1, B, 2)\rangle$. ACI plan repair with $V^{\mathsf{B}} = \{T\}$ finds flaw 1), π^+ does not satisfy the goal $T = l_2$. It inserts $go(l_1, l_2)$ at the end to fix that.*

Given an FDR task Π, the **painting strategies** associated with ACI choose V^{R} so as to guarantee that the resulting relaxed task Π^{RB} is in ACI. A major weakness in practice here

[1] In our implementation, we adapted *red facts following*, the more advanced repair algorithm by Katz and Hoffmann (2013).

is the restriction of V^{B} to invertible variables. In our example, T is the *only* such variable; we cannot paint M black, so we cannot fix flaw 2) pertaining to money consumption.

Intuitively, using ACI instead of full delete relaxation fixes the "moving to-and-fro" issue, for invertible moves now painted black (here: T). But it does not address resource consumption, which involves non-invertible variables (here: M).

Red-Black State Space Search

To enable convergence to real planning in the limit, red-black planning methods are required that can handle arbitrary paintings. Addressing this, Gnad et al. (2016) (*Gnad16*) have introduced **red-black state space search (RBS)**. RBS performs forward search with a relaxed fixed point over the red variables at each transition. At plan extraction time, RBS augments the solution path with a relaxed plan at each transition.

We require some notations. The **red actions** in an RB state s^{RB}, denoted $A^{\mathsf{R}}(s^{\mathsf{RB}})$, are the actions available to the relaxed fixed point at s^{RB}: the actions that comply with the black-variable values. $A^{\mathsf{R}}(s^{\mathsf{RB}}) := \{a^{\mathsf{R}} \mid a \in A, \mathsf{pre}_a[V^{\mathsf{B}}] \subseteq s^{\mathsf{RB}}, \mathsf{eff}_a[V^{\mathsf{B}}] \subseteq s^{\mathsf{RB}}\}$, where a^{R} is the projection of a onto V^{R}.

The relaxed fixed point at s^{RB} is now formalized in terms of a local planning task, namely the RB task $\Pi^+(s^{\mathsf{RB}}) := (\emptyset, V^{\mathsf{R}}, A^{\mathsf{R}}(s^{\mathsf{RB}}), s^{\mathsf{RB}}[V^{\mathsf{R}}], \emptyset)$. The **red completion** of s^{RB} is the RB state $\mathcal{F}^+(s^{\mathsf{RB}})$ where $\mathcal{F}^+(s^{\mathsf{RB}})[V^{\mathsf{B}}] = s^{\mathsf{RB}}[V^{\mathsf{B}}]$, and $\mathcal{F}^+(s^{\mathsf{RB}})[V^{\mathsf{R}}]$ is the set of all facts reachable in $\Pi^+(s^{\mathsf{RB}})$.

Definition 1 (Gnad16)**.** *Let Π^{RB} be an RB planning task. The **RB state space** is the transition system $\Theta^{\mathsf{RB}} = (S^{\mathsf{RB}}, T^{\mathsf{RB}}, A, s_0^{\mathsf{RB}}, S_G^{\mathsf{RB}})$. S^{RB} is the set of RB states. s_0^{RB} is the RB initial state. $S_G^{\mathsf{RB}} = \{s^{\mathsf{RB}} \mid \mathcal{F}^+(s^{\mathsf{RB}})$ is RB goal state$\}$. T^{RB} is the set of transitions $s^{\mathsf{RB}} \xrightarrow{a} t^{\mathsf{RB}}$ where a is applicable to $\mathcal{F}^+(s^{\mathsf{RB}})$, $\mathsf{eff}(a)[V^{\mathsf{B}}] \not\subseteq s^{\mathsf{RB}}[V^{\mathsf{B}}]$, and $t^{\mathsf{RB}} = \mathcal{F}^+(s^{\mathsf{RB}})[\![a]\!]$.*

Example 3. *Setting $V^{\mathsf{B}} = \{M\}$, $\mathcal{F}^+(s_0^{\mathsf{RB}})$ contains $T = l_1$ and $T = l_2$, but neither $A = 1$ nor $B = 1$ as buying a product affects the black variable M. The outgoing transitions of s_0^{RB} are the buy actions. $\langle buy(l_1, A, 2), buy(l_1, B, 1)\rangle$ leads to an RB goal state. For $buy(l_2, A, 2)$, in contrast, the outcome RB state t^{RB} has $t^{\mathsf{RB}}(M) = \{0\}$, so no further actions are applicable here and we detect that this is a dead-end.*

RB plan extraction augments backward solution path extraction with a relaxed plan extraction step at each transition. Assume that $\pi = \langle a_0, \ldots, a_{n-1}\rangle$ is a plan for Θ^{RB}, assume that backward extraction has already extracted an RB plan for the postfix $\pi_k := \langle a_k, \ldots, a_{n-1}\rangle$, and assume that the transition taken by a_{k-1} in π is $s_{k-1}^{\mathsf{RB}} \xrightarrow{a_{k-1}} s_k^{\mathsf{RB}}$. Then the **red goal** for relaxed plan extraction at this transition is $G(s_{k-1}^{\mathsf{RB}}) := \mathsf{Regress}^{\mathsf{R}}(G, a_{k-1} \circ \pi_k) \setminus s_{k-1}^{\mathsf{RB}}[V^{\mathsf{R}}]$, where $\mathsf{Regress}^{\mathsf{R}}$ is regression in the projection onto V^{R}. Intuitively, $G(s_{k-1}^{\mathsf{RB}})$ is the set of red facts that must be achieved before a_{k-1}, and that cannot be achieved further below. Any relaxed plan extraction mechanism can now be used on $\Pi^+(s_{k-1}^{\mathsf{RB}})$ to find a relaxed plan $\pi^+(s_{k-1}^{\mathsf{RB}})$ achieving $G(s_{k-1}^{\mathsf{RB}})$. Then π_k is replaced by $\pi^+(s_{k-1}^{\mathsf{RB}}) \circ a_{k-1} \circ \pi_k$, and we iterate.

Example 4. *In Example 3, denote $\pi = \langle buy(l_1, A, 2), buy(l_1, B, 1)\rangle = \langle a_0, a_1\rangle$. Denote the RB states along π as $s_0^{\mathsf{RB}}, s_1^{\mathsf{RB}}, s_2^{\mathsf{RB}}$. Plan extraction first processes $s_1^{\mathsf{RB}} \xrightarrow{a_1} s_2^{\mathsf{RB}}$. The red goal here is $G(s_1^{\mathsf{RB}}) = \emptyset$, as $\mathsf{Regress}^{\mathsf{R}}(\{A = 1, B = 1\}, buy(l_1, B, 1)) = \{T = l_1, A = 1\}$ and $s_1^{\mathsf{RB}}[V^{\mathsf{R}}] = \{T = l_2, T = l_1, A = 0, A = 1, B = 0\}$. The postfix thus simply is $\pi_1 = \langle buy(l_1, B, 1)\rangle$. In the next step though, at $s_0^{\mathsf{RB}} \xrightarrow{a_0} s_1^{\mathsf{RB}}$, the red goal is $G(s_0^{\mathsf{RB}}) = \{T = l_1\}$, leading to the relaxed plan $\langle go(l_2, l_1)\rangle$ and thus to the overall red-black plan $\pi^{\mathsf{RB}} = \langle go(l_2, l_1), buy(l_1, A, 2), buy(l_1, B, 1)\rangle$.*

Observe that π^{RB} in Example 4 is correct about M, but is flawed regarding T (as π^{RB} does not go back from l_1 to l_2 at the end, leaving the goal $T = l_2$ unsatisfied). This is complementary to the tractable fragment ACI, which can fix T but cannot fix M (cf. Example 2). The first new method we propose here is motivated by this kind of complementarity. We combine RBS with ACI to handle each kind of flaw with the most appropriate method.

Combining RBS with ACI

Any flaw in an RB plan π^{RB} can in principle be fixed by painting the respective variable v black, $V^{\mathsf{B}} := V^{\mathsf{B}} \cup \{v\}$, and re-running RBS. Yet Θ^{RB} grows exponentially in $|V^{\mathsf{B}}|$. Can we avoid the computational cost incurred by painting v black?

As we now show, the answer is yes – if, like for $v = T$ in Example 4, we can handle v by ACI instead. We can use ACI to effectively handle a tractable part of the task at hand (e. g. invertible moves to-and-fro), combined with RBS to handle the remainder (e. g. resource consumption).

The RBS+ACI Framework

Our combined framework, that we baptize **RBS+ACI**, distinguishes black variables of two different kinds, handled by RBS vs. ACI. So a painting is now a partition of V into three subsets $V^{\mathsf{RBS}}, V^{\mathsf{ACI}}, V^{\mathsf{R}}$ where $V^{\mathsf{B}} = V^{\mathsf{RBS}} \cup V^{\mathsf{ACI}}$.

Assume that such a partition is given. We need an RB plan relative to the entire set V^{B} of black variables, i. e. for the RB task $(V^{\mathsf{RBS}} \cup V^{\mathsf{ACI}}, V^{\mathsf{R}}, A, I, G)$. The basic idea is to apply ACI plan repair on the outcome of RBS on the coarser (more relaxed) task $\Pi_+^{\mathsf{RB}} := (V^{\mathsf{RBS}}, V^{\mathsf{R}} \cup V^{\mathsf{ACI}}, A, I, G)$.

ACI plan repair is defined for fully delete-relaxed plans, not RB plans, so we must adapt the repair process. We must make sure that the repair *1) is always possible given the black part V^{RBS} already fixed*, and *2) never affects that fixed part*.

Let π be the plan found by RBS for Π_+^{RB}. Our adapted repair process, **RBS+ACI plan repair**, computes a plan without conflicts on the entire set of black variables $V^{\mathsf{RBS}} \cup V^{\mathsf{ACI}}$, fixing unsatisfied conditions only on V^{ACI} without modifying the conflict-free V^{RBS}.

To ensure 2), an obvious and natural requirement is that there is no $a \in A$ with $\mathcal{V}(\mathsf{eff}_a) \cap V^{\mathsf{ACI}} \neq \emptyset$ and $\mathcal{V}(\mathsf{eff}_a) \cap V^{\mathsf{RBS}} \neq \emptyset$. That is, the repair actions will never affect V^{RBS}.

Ensuring 1) is more tricky. In RBS on Π^{RB}_+, the red completion $\mathcal{F}^+(s^{\mathsf{RB}})$ of any state s^{RB} uses only actions whose precondition is satisfied given the black variable assignment $s^{\mathsf{RB}}[V^{\mathsf{RBS}}]$. So one may think (and we did think at first) that no further restrictions are needed. However, across transitions $s^{\mathsf{RB}} \xrightarrow{a} t^{\mathsf{RB}}$, the fixed repair context changes from $s^{\mathsf{RB}}[V^{\mathsf{RBS}}]$ to $t^{\mathsf{RB}}[V^{\mathsf{RBS}}]$. This causes problems because, during RBS, the values reached for V^{ACI} in $\mathcal{F}^+(s^{\mathsf{RB}})$ are propagated to t^{RB}. But due to the different context $t^{\mathsf{RB}}[V^{\mathsf{RBS}}]$, the repair process at t^{RB} cannot necessarily reach these values.

Similar to Gnad and Hoffmann (2015), we impose that there is no $a \in A$ with $\mathcal{V}(\mathsf{eff}_a) \cap V^{\mathsf{ACI}} \neq \emptyset$ and $\mathcal{V}(\mathsf{pre}_a) \cap V^{\mathsf{RBS}} \neq \emptyset$, i. e., the repair actions do not have preconditions on V^{RBS}. We next show that this restriction is sufficient (the repair will always work). We then show that the restriction is necessary for computational reasons.

The conjunction of our two restrictions is equivalent to the absence of a causal graph arc from V^{RBS} to V^{ACI}. We say in this case that V^{ACI} **does not depend on** V^{RBS}.

Proposition 1. *Given an RB planning task $\Pi^{\mathsf{RB}} = (V^{\mathsf{B}}, V^{\mathsf{R}}, A, I, G)$, and a partition of V^{B} into V^{RBS} and V^{ACI} so that $(V^{\mathsf{ACI}}, V^{\mathsf{R}} \cup V^{\mathsf{RBS}}, A, I, G)$ is in ACI, and V^{ACI} does not depend on V^{RBS}. Let π be an RB plan for $\Pi^{\mathsf{RB}}_+ = (V^{\mathsf{RBS}}, V^{\mathsf{R}} \cup V^{\mathsf{ACI}}, A, I, G)$. Then RBS+ACI plan repair on π succeeds, and its output π^{RB} is an RB plan for Π^{RB}.*

Proof. Any action a that may be inserted by ACI plan repair, and hence by RBS+ACI plan repair, affects a variable in V^{ACI}. Therefore, by prerequisite, 1) a has no effect on V^{RBS}, and 2) a has no precondition on V^{RBS}. So the arguments given by Katz et al. (2013) remain applicable. □

Example 5. *Say we set $V^{\mathsf{RBS}} = \{M\}$ and $V^{\mathsf{ACI}} = \{T\}$. Note that M depends on T: this dependency direction is allowed.*

RBS is run on $\Pi^{\mathsf{RB}}_+ = (\{M\}, \{T, A, B\}, A, I, G)$. The outcome is $\pi = \langle go(l_2, l_1), buy(l_1, A, 2), buy(l_1, B, 1)\rangle$. Running ACI plan repair on π finds the unsatisfied goal condition $g = \{T = l_2\}$ at the end. This is repaired by appending $\langle go(l_1, l_2)\rangle$ to π, yielding a plan for the original task.

Proposition 1 shows that our RBS+ACI framework is sound for RB planning in Π^{RB}. Completeness holds, too:

Proposition 2. *Under the prerequisites of Proposition 1, an RB plan for $\Pi^{\mathsf{RB}} = (V^{\mathsf{RBS}} \cup V^{\mathsf{ACI}}, V^{\mathsf{R}}, A, I, G)$ exists iff an RB plan for $\Pi^{\mathsf{RB}}_+ = (V^{\mathsf{RBS}}, V^{\mathsf{R}} \cup V^{\mathsf{ACI}}, A, I, G)$ exists.*

Proof. The "if" direction holds by Proposition 1. The "only if" direction holds because Π^{RB} is a refinement of Π^{RB}_+. □

So our approach works provided there is no CG arc from V^{RBS} to V^{ACI}. Let us show that this restriction is necessary. Consider the decision problem **RBS-dependent ACI PlanGen**, defined as follows. Given $\Pi^{\mathsf{RB}} = (V^{\mathsf{B}}, V^{\mathsf{R}}, A, I, G)$ and a partition of V^{B} into V^{RBS} and V^{ACI} s.t. $(V^{\mathsf{ACI}}, V^{\mathsf{R}} \cup V^{\mathsf{RBS}}, A, I, G)$ is in ACI, and all CG arcs between V^{RBS} and V^{ACI}, if any, go from V^{RBS} to V^{ACI}. Given an RB plan π for $\Pi^{\mathsf{RB}}_+ = (V^{\mathsf{RBS}}, V^{\mathsf{R}} \cup V^{\mathsf{ACI}}, A, I, G)$. Denote by $\pi|_{V^{\mathsf{RBS}}}$ the subsequence of V^{RBS}-affecting actions in π. Decide whether $\pi|_{V^{\mathsf{RBS}}}$ is a subsequence of an RB plan for Π^{RB}.

Theorem 1. *RBS-dependent ACI PlanGen is **NP**-hard.*

Proof. By a reduction from SAT. Let ϕ be a CNF formula with propositions $p_1, \dots, p_n$ and clauses $c_1, \dots, c_m$. Our planning encoding first chooses values for p_i, then satisfies the clauses c_j. The construction sets V^{RBS} to contain a single "indicator" variable, determining whether we can right now set p_i to 0 or to 1; V^{ACI} represents this choice of values; and V^{R} represents whether or not a clause has been satisfied yet.

In detail, we set $V^{\mathsf{RBS}} = \{v\}$ with domain $\{0, 1\}$, initial value 0, and a single action $a[v01]$ going from 0 to 1. We set $V^{\mathsf{ACI}} = \{v_{p_1}, \dots, v_{p_n}\}$ with domain $\{u, 0, 1\}$, initial value u, actions going from u to 0 with precondition $v = 0$, and actions going from u to 1 with precondition $v = 1$. We set $V^{\mathsf{R}} = \{v_{c_1}, \dots, v_{c_m}\}$ with domain $\{0, 1\}$, initial value 0, goal value 1, and an action $a[v_{c_j}01]$ setting v_{c_j} from 0 to 1 with precondition $\{v = 1, v_{p_i} = x\}$ for each $(p_i = x) \in c_j$.

Observe first that this RB planning task Π^{RB} does satisfy the prerequisites: all $v_{p_i} \in V^{\mathsf{ACI}}$ are invertible, and there are no dependencies across these variables; the dependencies between V^{RBS} and V^{ACI} consist in the CG arcs (v, v_{p_i}).

Consider now $\pi|_{V^{\mathsf{RBS}}} := \langle a[v01]\rangle$. This is a subsequence of an RB plan π for Π^{RB}_+: We can move each v_{p_i} to $v_{p_i} = 0$ before the application of $a[v01]$, and to $v_{p_i} = 1$ after that application. Any formula ϕ can be satisfied that way.

But is $\pi|_{V^{\mathsf{RBS}}}$ a subsequence of an RB plan for Π^{RB}? The answer is "yes" iff ϕ is satisfiable. This is because $\pi|_{V^{\mathsf{RBS}}}$ is (trivially) a subsequence of any RB plan for Π^{RB}, and an RB plan for Π^{RB} exists iff ϕ is satisfiable. The latter is true because, in Π^{RB}, each v_{p_i} can support the clause-satisfying actions $a[v_{c_j}01]$ with only a single truth value. First, $v_{p_i} = 1$ can only be reached after $a[v01]$, at which point $v_{p_i} = 0$ is no longer reachable. Second, we can set $v_{p_i} = 0$ before the application of $a[v01]$. But at that point, $a[v_{c_j}01]$ is not yet applicable due to its precondition $v = 1$. So we must apply $a[v01]$, and afterwards we can no longer reach $v_{p_i} = 1$. □

By Theorem 1, given the fixed solution path $\pi|_{V^{\mathsf{RBS}}}$ found by RBS for Π^{RB}_+, augmenting $\pi|_{V^{\mathsf{RBS}}}$ to an RB plan for Π^{RB} is hard. In our framework, such augmentation is done by red (delete-relaxed) planning in Π^{RB}_+ alongside $\pi|_{V^{\mathsf{RBS}}}$, followed by RBS+ACI plan repair. So one of these steps would need to have worst-case exponential runtime (unless $\mathbf{P} = \mathbf{NP}$). In other words, efficient RBS+ACI plan repair is not possible when allowing CG arcs from V^{RBS} to V^{ACI}.

In practice, i. e., in our overall planning algorithm introduced next, one can ameliorate the situation by *attempting* RBS+ACI plan repair even if V^{ACI} does depend on V^{RBS}. If the repair succeeds, all is fine. We only need to act – remove the problematic variable(s) from V^{ACI} – if the repair fails.

Overall Planning Process: Iterated RBS+ACI

We now know how to solve any RB task Π^{RB} with a painting $V^{\mathsf{RBS}}, V^{\mathsf{ACI}}, V^{\mathsf{R}}$ that qualifies for Proposition 1. But our aim here is to find real plans, for the original FDR input task

Π. So RBS+ACI becomes a tool within an overall planning process.

That process is a loop around RBS+ACI searches with increasingly refined paintings. In a pre-process, we compute an ACI painting $V_0^{\mathsf{B}}, V_0^{\mathsf{R}}$ using the default painting strategy in Mercury, which orders the variables by causal graph level and iteratively paints variables red until the black CG is a DAG (Katz and Hoffmann 2014). We then initialize our painting as $V^{\mathsf{RBS}} := \emptyset, V^{\mathsf{ACI}} := V_0^{\mathsf{B}}, V^{\mathsf{R}} := V_0^{\mathsf{R}}$. We run RBS+ACI on that painting. If an RB plan does not exist, we know that Π is unsolvable and we stop. Otherwise, we now have an RB plan π^{RB}. We check whether π^{RB} is a real plan for Π. If yes, we stop. Otherwise, we refine our painting. Namely, we simulate the execution of π^{RB} under the real planning semantics in Π, and we count the number of flaws associated with each variable $v \in V^{\mathsf{R}}$. We select $v \in V^{\mathsf{R}}$ with a maximal number of flaws (a criterion adapted from Mercury). We set $V^{\mathsf{RBS}} := V^{\mathsf{RBS}} \cup \{v\}$ and $V^{\mathsf{R}} := V^{\mathsf{R}} \setminus \{v\}$, and iterate.

Adding v to V^{RBS} may introduce dependencies of V^{ACI} on V^{RBS}. Therefore, as discussed above, at some point RBS+ACI plan repair may fail. In that case, we move the culprit variable(s) from V^{ACI} to V^{R}, re-establishing the Proposition 1 guarantee that repair will succeed. The red-black relaxation considered is, then, no longer a refinement of the previous one. But convergence to $V^{\mathsf{B}} = V$ remains intact, so that the completeness of the overall planning process is preserved.

Whenever checking whether an intermediate RB plan π^{RB} works under the real planning semantics in Π, a variant is to *commit* to the prefix that works. We will refer to this as **prefix-execution**. The advantage is that the next iteration of RBS+ACI will not have to start from scratch on the initial state. On the downside, of course this loses completeness.

Adaptive Refinement via Realizability

An iterative refinement loop around RBS, as in iterated RBS+ACI, is wasteful in that every iteration of RBS starts from scratch, re-building the entire RB state space. Prefix-execution fixes this, but in a very limited way. Ideally, like other abstraction refinement processes, we ought to refine in an adaptive manner, *only where needed*, and do so incrementally within a single, iteratively refined, relaxed search space.

But how to do this in RBS, and effectively for the purpose of finding real plans? The straightforward approach would be to search until an RB plan π^{RB} is found, execute π^{RB} against the real semantics until the first flaw occurs at RB state s^{RB}, then accordingly refine the painting and re-do the RBS search space below s^{RB}. But there are a number of issues with this. First, it saves us only the work otherwise done above s^{RB} (similarly as the much simpler prefix-execution). Second, with many black variables – as needed to find real plans – finding π^{RB} becomes very expensive so there will be long time intervals between the local refinement steps. Which is especially wasteful as, third, things often go wrong at the root of an RBS sub-tree already. To illustrate the latter, say that the only action applicable at the root s^{RB} has red preconditions p and q, each of which is reached in $\mathcal{F}^+(s^{\mathsf{RB}})$ but which are in conflict so their conjunction is not reachable under the real semantics. Then all search below s^{RB} is wasted.

Given these observations, here we design an eager approach, imposing refinements whenever a transition in Θ^{RB} will not work out in reality. We first show how to do this in RBS, then we discuss the combination with ACI.

Realizability Refinement: X-RBS

Let s^{RB} be any RB state in Θ^{RB}, and let $s^{\mathsf{RB}} \xrightarrow{a} t^{\mathsf{RB}}$ be any outgoing transition of s^{RB}. By construction, we know that $\mathsf{pre}_a[V^{\mathsf{R}}] \subseteq \mathcal{F}^+(s^{\mathsf{RB}})$. That is, the red preconditions of a can be achieved in the delete-relaxed task $\Pi^+(s^{\mathsf{RB}})$ at s^{RB}. Let now π_X^+ be a relaxed plan for the goal $\mathsf{pre}_a[V^{\mathsf{R}}]$ in $\Pi^+(s^{\mathsf{RB}})$, extracted by some relaxed-plan extraction method X. If π_X^+ achieves $\mathsf{pre}_a[V^{\mathsf{R}}]$ under the *real* semantics $V^{\mathsf{B}} = V$, we say that $s^{\mathsf{RB}} \xrightarrow{a} t^{\mathsf{RB}}$ is **realized by** π_X^+ and is **realizable given X**.

Definition 2. *Let Π^{RB} be an RB planning task, and let X be a relaxed-plan extraction method. The **X-RB state space** is the transition system Θ_X^{RB} defined like Θ^{RB} except that:*

(i) transitions $s^{\mathsf{RB}} \xrightarrow{a} t^{\mathsf{RB}}$ not realizable given X are pruned;

(ii) if $s^{\mathsf{RB}} \xrightarrow{a} t^{\mathsf{RB}}$ is realized by π_X^+, then t^{RB} is the outcome state of executing $\pi_X^+ \circ a$ in s^{RB} with $V^{\mathsf{B}} = V$.

Some remarks are in order. First, the rationale behind (i) is that red-black plans will be extracted using X, so if X does not actually achieve pre_a in reality then $s^{\mathsf{RB}} \xrightarrow{a} t^{\mathsf{RB}}$ won't be in a real plan. It is of course a restriction here to commit to X. But there is no systematic alternative: short of a full-scale planning process for pre_a – giving up on the relaxation altogether – if X does not find a real plan, then the best one could do is try another relaxed plan extraction method X'.

Second, that said, Definition 2 is only one half of the story. Whenever a transition $s^{\mathsf{RB}} \xrightarrow{a} t^{\mathsf{RB}}$ is pruned by (i), we spawn a **refinement option**, discussed in detail below. A refinement option is a refined RB planning task at s^{RB}, addressing the reason for non-realizability of $s^{\mathsf{RB}} \xrightarrow{a} t^{\mathsf{RB}}$.

Finally, (ii) has the immediate effect that every reachable state s^{RB} in Θ_X^{RB} is in fact a real state. It turns the red part of the search (the method X) into a fast macro-generator to the next applicable black-variable affecting action. Observe that this is a natural match with our realizability check. What realizability affirms is that, in reality, we can reach pre_a at s^{RB}. In contrast, the over-approximated state transition, without (ii), would pretend that we can reach *the entire set* $\mathcal{F}^+(s^{\mathsf{RB}})$. Intuitively, we can check the validity of $s^{\mathsf{RB}} \xrightarrow{a} t^{\mathsf{RB}}$ only in a limited way, because we don't a-priori know what the red goal might be here at plan extraction time. So we commit to the minimal way of both, checking and using, the transition. (On the side, realizability checks without (ii) would apply the real semantics starting from an RB state, another mismatch.)

Now, that said, (ii) is a choice we made in our work so far. Exploring alternate definitions is a topic for future work.

Let us now turn to refinement options:

Definition 3. *Let $\Pi^{\mathsf{RB}} = (V^{\mathsf{B}}, V^{\mathsf{R}}, A, I, G)$ be an RB planning task. Let $s^{\mathsf{RB}} \xrightarrow{a} t^{\mathsf{RB}}$ be a transition pruned in Θ^{RB}_X, not realized by π^+_X. Let $v \in V^{\mathsf{R}}$ be s.t. π^+_X contains a maximal number of flaws on v. Then $\Pi^{\mathsf{RB}}_{+v}(s^{\mathsf{RB}}) := (V^{\mathsf{B}} \cup \{v\}, V^{\mathsf{R}} \setminus \{v\}, A, s^{\mathsf{RB}}, G)$ is a **refinement option** for $s^{\mathsf{RB}} \xrightarrow{a} t^{\mathsf{RB}}$.*

Whenever a transition $s^{\mathsf{RB}} \xrightarrow{a} t^{\mathsf{RB}}$ is pruned in our exploration of Θ^{RB}_X, we generate a refinement option $\Pi^{\mathsf{RB}}_{+v}(s^{\mathsf{RB}})$. That option is inserted as a search node into the overall (heuristic) search. Thus, the search decides not only which states to explore, but also which refinement is used to explore that state. We will refer to this overall search framework as **X-RBS**.

Observe that the under-approximation (ii) loses completeness, i. e., our overall search space may not contain a plan: below realizable transitions, the commitment to π^+_X may exclude the solutions. As an optional fix, **refinement-explored**, we also spawn refinement options at nodes s^{RB} all of whose descendants have been unsuccessfully explored. In such a case, we do not have a concrete flaw to fix, so we pick a variable $v \in V^{\mathsf{R}}$ to paint black arbitrarily.

Combination with ACI

The number of refinement options can be a major source of computational overhead in X-RBS. One way to ameliorate this is to combine X-RBS with ACI, to **X-RBS+ACI**: replacing delete-relaxed planning with tractable red-black planning will result in fewer flaws, and in more realizable transitions.

The combination is simple in X-RBS as relaxed planning occurs only at individual transitions $s^{\mathsf{RB}} \xrightarrow{a} t^{\mathsf{RB}}$. It 1) generates $\mathcal{F}^+(s^{\mathsf{RB}})$ to test whether pre_a is relaxed-reachable; it 2) extracts a relaxed plan using method X, to check realizability.

Using ACI instead, 1) remains unchanged. For 2), we use ACI plan repair on top of X. This uses separate sets V^{RBS} vs. V^{ACI} of black variables as before, but with no constraint on their dependencies: in a realizability check – against the real semantics – a success guarantee cannot be given anyhow.

Experiments

Our techniques are implemented on top of Gnad16's RBS, which modifies Fast Downward (FD) (Helmert 2006) in a minimally intrusive way, exchanging the state and state transition data structures while preserving all search algorithms. All our configurations run FD's greedy best-first dual-queue search with Gnad16's h^{FF} extension and preferred operators.

We run each of RBS and X-RBS with vs. without ACI. We run RBS with vs. without prefix-execution (PE), and X-RBS with vs. without refinement-explored (RE), yielding eight different configurations. Among these, RBS with neither ACI nor prefix-execution is a baseline easily derived from (though not evaluated by) Gnad16. To represent the state of the art in satisficing planning, we run LAMA (Richter and Westphal 2010) and Mercury (Katz and Hoffmann 2014). We also run the best-performing LPG-plan-repair configuration by Gnad16. This paints 90% of the variables black, uses RBS to find an RB plan π^{RB}, then calls LPG to repair π^{RB} into a real plan.

We run all IPC satisficing STRIPS benchmarks. All experiments were run on a cluster of Intel Xeon E5-2650v3 machines, with runtime (memory) limits of 30 minutes (4 GB).

	RBS				X-RBS						
			+ACI				+ACI		RBS		Mer-
		+PE		+PE		+RE		+RE	+LPG	LAMA	cury
Airport (50)	27	28	27	28	41	43	41	**44**	42	32	32
Barman (40)	0	3	0	3	0	7	0	0	24	39	**40**
Blocks (35)	**35**	**35**	**35**	**35**	**35**	**35**	24	33		**35**	**35**
Childsnack (20)	5	**20**	9	10	0	0	0	0	4	5	0
Depots (22)	15	17	16	18	1	9	14	15	**21**	20	**21**
Driverlog (20)	19	18	**20**	19	2	7	3	9	18	**20**	**20**
Elevat (50)	45	47	**50**	**50**	0	12	**50**	**50**	**50**	**50**	**50**
Floortile (40)	3	3	6	7	0	4	0	0	**9**	8	8
Freecell (80)	71	69	71	69	69	61	69	60	35	79	**80**
GED (20)	10	9	10	10	**20**	**20**	14	0	4	**20**	**20**
Grid (5)	4	4	**5**	4	0	2	4	**5**	4	**5**	**5**
Hiking (20)	**20**	**20**	15	17	18	15	18	**20**	19	18	**20**
Logistics (63)	62	62	**63**	**63**	0	12	**63**	**63**	35	**63**	**63**
Maintenan (20)	**11**	7	**11**	7	0	0	0	0		0	7
Mprime (35)	**35**	34	**35**	**35**	3	18	**35**	34	**35**	**35**	**35**
Mystery (19)	16	13	17	13	1	8	**19**	18	16	**19**	**19**
NoMystery (20)	**19**	**19**	**19**	17	0	4	1	4	**19**	11	14
ParcPrin(50)	49	49	49	49	39	48	36	37	35	49	**50**
Parking (40)	12	13	11	13	0	0	0	0	0	**40**	**40**
Pathways (30)	21	28	21	28	27	26	27	26	21	23	**30**
PegSol (50)	**50**	**50**	**50**	**50**	**50**	**50**	**50**	37	16	**50**	**50**
PipesNoT (50)	35	38	36	38	34	25	25	17	39	43	**44**
PipesTank (50)	31	26	28	30	26	20	34	18	24	**42**	**42**
PSR (50)	**50**	**50**	**50**	**50**	0	49	0	49	**50**	**50**	**50**
Rovers (40)	**40**	**40**	**40**	**40**	2	16	18	20		**40**	**40**
Satellite (36)	**36**	**36**	**36**	**36**	0	5	**36**	**36**		**36**	**36**
Scanaly (50)	42	46	42	**50**	43	42	44	44	46	**50**	**50**
Sokoban (50)	20	15	22	13	44	44	29	9	5	**48**	42
Storage (30)	18	20	18	18	16	17	**28**	**28**	25	19	19
Tetris (20)	0	3	0	2	1	0	3	2	0	13	**19**
Thoughtful (20)	6	11	6	10	15	13	9	5		**16**	13
Tidybot (20)	8	6	7	8	0	2	0	0	13	**17**	15
TPP (30)	**30**	**30**	**30**	**30**	0	10	**30**	27	**30**	**30**	**30**
Transpo (70)	31	33	**70**	**70**	0	20	61	57	45	61	**70**
Trucks (30)	12	12	12	12	4	10	0	8	**20**	15	19
VisitAll (40)	3	4	**40**	**40**	3	3	**40**	**40**	4	**40**	**40**
Woodw (50)	**50**	49	**50**	49	17	16	10	13	47	**50**	**50**
Zenotrav (20)	**20**	**20**	**20**	**20**	1	7	**20**	**20**		**20**	**20**
$\sum$ (1385)	961	987	1047	1061	512	680	855	848	755	1211	**1238**

Table 1: Coverage. Best results **highlighted**. We omit domains where all tested planners have full coverage. RBS+LPG is RBS followed by LPG plan repair (empty entries could not be run, see text).

Coverage

Consider Table 1, and the variants of RBS (leftmost part of the table). Relative to the baseline, our techniques (+ACI and +PE) improve performance substantially. This is clearly visible in overall coverage. Per domain, +PE yields better coverage in 14 domains, +ACI in 12, and the two together in 15. Both techniques also have their drawbacks, as +PE

does not work well if the prefix often leads into dead ends (e.g. in Sokoban). Furthermore, +ACI can sometimes introduce more conflicts into the partially relaxed plan. This happens e.g. in Childsnack, where otherwise the RBS+PE configuration only needs to paint the sandwich objects and tray locations black (22-25% of the total variables) to make the red-black plan a real plan, solving all instances in less than 5 seconds.

For the X-RBS method, in the middle part of Table 1, the results are much worse, in many domains and hence in the overall. A key reason is the overhead from too many refinement options. On average, 74% of the generated transitions are realizable, in some domains much less (15% in Parking, 18% in Tetris). As expected, the combination with ACI ameliorates this significantly. But it remains a question for future work how X-RBS can be made competitive overall. While the +RE option helps in domains where X-RBS fails often, it also increases the overhead of too many refinement options.

Consider now RBS+LPG. The empty entries in Table 1 are domains where that architecture did not run properly, for implementation reasons (Gnad16's results do not include these domains either). Filling in the gaps optimistically – assuming that RBS+LPG can solve *all* instances in the missing domains – overall coverage becomes 934. This still lags behind our RBS methods, even the baseline. On a per-domain level though, the methods are highly complementary: of the 32 domains, RBS beats RBS+LPG in 12 and is inferior in 12; RBS+ACI+RE beats RBS+LPG in 16 and is inferior in 11.

For our X-RBS configurations, the comparison to RBS+LPG is, naturally, less favorable. Complementarity at per-domain level persists though. X-RBS+ACI beats RBS+LPG in 13 domains and is inferior in 14.

Consider finally LAMA and Mercury. All our configurations are far from their performance overall. Our best configuration, RBS+ACI+PE, beats LAMA in 5 domains and is inferior in 20; for Mercury, these numbers are 2 vs. 22.

That said, there are five domains in which at least one of our configurations works exceptionally well. In Airport, our best method gains +12 coverage over the best of LAMA and Mercury; in Childsnack, +15; in Maintenance, +4; in NoMystery, +5; in Storage, +9. So the new methods can potentially contribute in portfolios or per-domain autoconfiguration.

#Black Variables until Solution in RBS

The major motivation behind our +ACI and +PE extensions to RBS is to reduce the size of V^{RBS} required to find a real plan. Figure 2 measures this impact directly.

Both extensions clearly help as intended. Without +ACI, few instances can be solved without search ($|V^{\mathsf{RBS}}| = 0$) as, there, the delete-relaxed plan for the initial state has to be a real plan. The advantage of our extensions remains strong when allowing larger V^{RBS}, until about $|V^{\mathsf{RBS}}|/|V| = 50\%$ where the gap narrows. After that, the difference is mainly due to benchmarks (like Transport) that ACI solves on the initial state but that are beyond reach of RBS search alone.

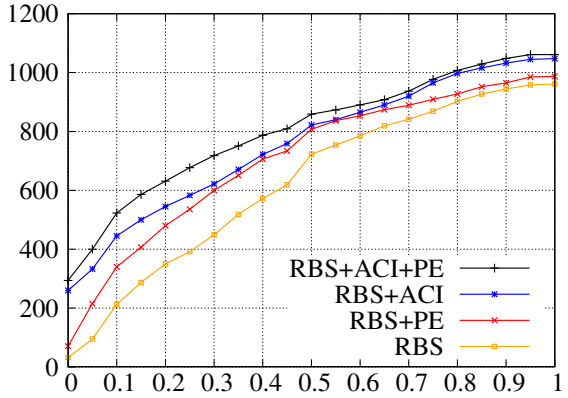


Figure 2: Coverage as a function of the fraction of RBS variables, $|V^{\mathsf{RBS}}|/|V|$, in the first iteration of RBS that finds a real plan.

Conclusion

We have shown that RBS can be synergetically combined with ACI tractable red-black planning, and we have started the exploration of adaptive relaxation refinement within RBS. The results for the former show performance improvements due to the smaller number of black variables that need to be searched over. The results for the latter exhibit promise, but the jury is still out how such adaptive refinement is best done.

Overall, our work contributes another piece in the puzzle how to tap into the power of partial delete relaxation without incurring a prohibitive overhead. This fits into the larger puzzle of how to use informative but costly approximations. We believe that such research is valuable to complement the more prominent focus on fast-but-inaccurate approximations, and we hope that our ideas and insights may be useful for approaches other than red-black planning as well.

Acknowledgments

This work was partially supported by the German Research Foundation (DFG), under grants HO 2169/5-1 ("Critically Constrained Planning via Partial Delete Relaxation") and HO 2169/6-1 ("Star-Topology Decoupled State Space Search").

References

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

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

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

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

Fox, M., and Long, D. 2001. Stan4: A hybrid planning strategy based on subproblem abstraction. *The AI Magazine* 22(3):81–84.

Fox, M.; Gerevini, A. E.; Long, D.; and Serina, I. 2006. Plan stability: Replanning versus plan repair. In Long, D., and Smith, S., eds., *Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS'06)*, 212–221. Ambleside, UK: Morgan Kaufmann.

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

Gnad, D., and Hoffmann, J. 2015. Red-black planning: A new tractability analysis and heuristic function. In Lelis, L., and Stern, R., eds., *Proceedings of the 8th Annual Symposium on Combinatorial Search (SOCS'15)*. AAAI Press.

Gnad, D.; Steinmetz, M.; Jany, M.; Hoffmann, J.; Serina, I.; and Gerevini, A. 2016. Partial delete relaxation, unchained: On intractable red-black planning and its applications. In Baier, J., and Botea, A., eds., *Proceedings of the 9th Annual Symposium on Combinatorial Search (SOCS'16)*. AAAI Press.

Haslum, P. 2012. Incremental lower bounds for additive cost planning problems. In Bonet, B.; McCluskey, L.; Silva, J. R.; and Williams, B., eds., *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*, 74–82. AAAI Press.

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

Helmert, M., and Geffner, H. 2008. Unifying the causal graph and additive heuristics. In Rintanen, J.; Nebel, B.; Beck, J. C.; and Hansen, E., eds., *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS'08)*, 140–147. AAAI Press.

Helmert, M.; Haslum, P.; Hoffmann, J.; and Nissim, R. 2014. Merge & shrink abstraction: A method for generating lower bounds in factored state spaces. *Journal of the Association for Computing Machinery* 61(3).

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

Helmert, M. 2009. Concise finite-domain representations for PDDL planning tasks. *Artificial Intelligence* 173:503–535.

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

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

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

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

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

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

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

Relaxed Decision Diagrams for Cost-Optimal Classical Planning

Margarita P. Castro[†], **Chiara Piacentini**[†], **Andre A. Cire**[‡], and **J. Christopher Beck**[†]
[†]Department of Mechanical and Industrial Engineering, University of Toronto, Toronto, Canada, ON M5S 3G8
[‡]Department of Management, University of Toronto Scarborough, Toronto, Canada, ON M1C 1A4

Abstract

We explore the use of multivalued decision diagrams (MDDs) to represent a relaxation of the state-transition graph for classical planning problems. The relaxation exploits the exact state transitions up to a pre-defined memory limit and uses value-accumulating semantics when the limit is reached. Moreover, it provides admissible heuristic values by means of an efficient shortest-path algorithm, which is applied in an A^* algorithm to find cost-optimal plans. We also consider a variant of A^* that takes advantage of feasible solutions extracted by the MDD to reduce the number of states that need to be evaluated. Our experimental evaluation shows that the MDD-based heuristic, despite being computationally more expensive, can be more informative than some state-of-the-art admissible heuristics.

1 Introduction

We present a new admissible heuristic based on a *relaxed multivalued decision diagram* (MDD). A relaxed MDD is a graph of restricted size that over-approximates the set of feasible solutions to a discrete problem. Relaxed MDDs have been largely applied to mathematical programming and discrete optimization, in particular for obtaining optimization bounds for combinatorial and scheduling problems (Hoda, Van Hoeve, and Hooker 2010; Bergman et al. 2016; Kinable, Cire, and van Hoeve 2017).

This paper defines relaxed MDDs for a classical planning task and uses them to compute a novel admissible heuristic to reach a goal node. We explore the relationship between relaxed MDDs and existing techniques to solve classical planning problems, showing that a relaxed MDD is an abstraction of the transition graph for a planning task and that our heuristic dominates the well-known h^{max} heuristic (Bonet and Geffner 2000).

The MDD-based heuristic is used in a variant of A^* inspired by a branch-and-bound tree search. We enhance the A^* search algorithm with a bounding mechanism that reduces the number of states expanded via bounds on plan cost derived from feasible plans extracted from the MDD. The new algorithm is therefore suitable for finding both feasible and optimal plans.

The paper is organized as follows. Section 2 defines a classical planning task and presents related work. Section 3 defines a relaxed MDD for classical planning and Section 4 presents the construction procedure. Section 5 relates relaxed MDDs to transition graphs and compares them to other heuristics in classical planning. Section 6 explains the implementation and our preliminary results are presented in Section 7. Lastly, Section 8 discusses the approach and possible directions for future research.

2 Background

This section presents a formal definition of a cost-optimal classical planning, introduces the notation used in this paper, and reviews work in the classical planning literature that is related to our relaxed MDD approach.

2.1 Cost-Optimal Classical Planning

We consider cost-optimal classical planning tasks with non-zero cost actions using the STRIPS formalism. A planning *task* is a tuple $\Pi = \langle \mathcal{P}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$, where $\mathcal{P}$ is the set of *propositional* variables, $\mathcal{A}$ is the set of actions, $\mathcal{I} \subseteq \mathcal{P}$ is the initial state, and $\mathcal{G} \subseteq \mathcal{P}$ is the set of goal conditions. A state s is defined as a subset of propositional variables, $s \subseteq \mathcal{P}$.

An action $a \in \mathcal{A}$ is a tuple $\langle pre(a), add(a), del(a), c(a) \rangle$, where $pre(a) \subseteq \mathcal{P}$ is the set of preconditions, $add(a) \subseteq \mathcal{P}$ is the set of add effects, $del(a) \subseteq \mathcal{P}$ is the set of delete effects, and $c(a) > 0$ is the action cost. An action a is applicable to a state s if the preconditions are satisfied in s, i.e., $pre(a) \subseteq s$. The application of an action a to a state s produces a successor state s' given by $s' = \phi(a, s) = (s \setminus del(a)) \cup add(a)$.

A solution of a planning task Π is a *plan*, i.e., a sequence of actions such that each action is applicable in its predecessor state and the last state satisfies the goal conditions. Formally, $\pi = (a_0, \dots, a_n)$ is a plan if for each action a_i in π, $pre(a_i) \subseteq \phi(a_{i-1}, \phi(a_{i-2}, \dots \phi(a_0, \mathcal{I})))$, and $\mathcal{G} \subseteq \phi(a_n, \phi(a_{n-1}, \dots \phi(a_0, \mathcal{I}))) = \phi(\pi, \mathcal{I})$.

The cost of a plan π is the sum of all the actions appearing in π, i.e., $c(\pi) = \sum_{i=0}^{n} c(a_i)$. A cost-optimal plan $\hat{\pi}$ is a plan with minimum cost, i.e, $c(\hat{\pi}) \leq c(\pi)$ for any plan π of Π.

Given a planning task Π, we define a delete-free planning task Π^+ where delete effects are ignored. Formally, the delete-free task is given by $\Pi^+ = \langle \mathcal{P}, \mathcal{A}^+, \mathcal{I}, \mathcal{G} \rangle$, where for each $a \in \mathcal{A}$ there is an action $a' \in \mathcal{A}^+$ such that $pre(a') = pre(a)$, $add(a') = add(a)$ and $del(a') = \emptyset$. A *delete relaxation* of a planning task Π refers to its associated delete-free task Π^+.

2.2 Related Work in Planning

Our work is closely related to heuristics based on graphical structures, such as Graphplan (Blum and Furst 1997), red-black relaxed plans (Katz, Hoffmann, and Domshlak 2013), and abstractions (Edelkamp 2001; Helmert et al. 2007). We also discuss the use of decision diagrams for symbolic A^* search in classical planning (Torralba, Linares López, and Borrajo 2016) and the differences with our approach.

Graphplan (Blum and Furst 1997) is a compact data structure for encoding planning problems. It is a directed and layered graph with alternating propositional and action layers, in which nodes represent propositions and actions, respectively. Edges connect a proposition to an action node if the proposition is a precondition of the action, and an action to a proposition node if the proposition belongs to the add or delete effects of the action. Graphplan derives the admissible heuristic h^G by taking the index of the first layer where the goal conditions appear without any mutex relation (Bonet and Geffner 2000). A relaxed version of Graphplan, called the Relaxed Planning Graph (RPG), represents the delete relaxation of a planning task. Relaxed plans can be extracted from the RPG in polynomial time and yield the non-admissible heuristic h^{FF} (Hoffmann and Nebel 2001).

While the delete relaxation provides several other heuristics, e.g., h^{max}, h^{add} (Bonet and Geffner 2001) and $h^{LM\text{-}cut}$ (Helmert and Domshlak 2009), ignoring the delete effects can result in a poor heuristic estimation. Red-black planning heuristics overcome some of the problems of delete relaxation heuristics by dividing the propositional variables into two groups: one that follows the semantics of the delete relaxation and one that takes into account the delete effects of actions (Domshlak, Hoffmann, and Katz 2015). Our relaxed MDD heuristic follows a similar idea in the sense that we partially ignore delete effects, though our approach to doing so is by considering nodes as the union of plan states.

Abstraction-based heuristics are also related to our work. An abstraction maps the search space into a smaller one in which an optimal path from an abstract initial state to an abstract goal state is an admissible heuristic. Different abstraction mappings result in different heuristics, for example pattern database heuristics (Edelkamp 2001) and *merge-and-shrink* (Helmert et al. 2007; Sievers, Wehrle, and Helmert 2014). Our relaxed MDD representation of a planning task can be viewed as an abstraction, as detailed in Section 5.1.

Binary decision diagrams (BDDs) have been used in planning to succinctly represent sets of states (symbolic states). Using this representation, a symbolic version of the A^* search algorithm achieves state-of-the-art performance in cost-optimal classical planning (Torralba, Linares López, and Borrajo 2016). Several admissible heuristics have been proposed to guide the search over the symbolic state-space, e.g., abstraction-based heuristics (Edelkamp, Kissmann, and Torralba 2012; Torralba, López, and Borrajo 2013). In contrast, our approach uses relaxed MDDs to compute admissible heuristics on a standard A^* search algorithm.

Lastly, the planning literature has used edge-value multi-valued decision diagrams (EVMDD) to represent cost functions of planning problem with state-dependent actions costs (Keller et al. 2016; Geißer, Keller, and Mattmüller 2016).

3 Relaxed MDDs for Planning

In this section, we demonstrate the use of relaxed MDDs as a graphical structure to approximate the state-space transition graph. We first define an MDD for classical planning and then extend the definition to relaxed MDDs.

Consider τ as an upper bound on the number of actions in a cost-optimal plan. An MDD for a classical planning task Π is a graphical structure that, starting from the initial state $\mathcal{I}$, represents the set of reachable states after applying at most τ actions. Specifically, an MDD $\mathcal{M} = (\mathcal{N}, \mathcal{E})$ is a layered directed acyclic graph where $\mathcal{N}$ is the set of nodes and $\mathcal{E}$ is the set of edges. Each node u has a label $\sigma(u)$ that represents a reachable state, i.e., $\sigma(u) \subseteq \mathcal{P}$ is the set of propositions in the state. In particular, the set of nodes is divided into layers $\mathcal{N} = \{\mathcal{N}_0, \mathcal{N}_1, ..., \mathcal{N}_\tau\}$, where layer $\mathcal{N}_0 = \{\mathbf{r}\}$ has a single node, called the root node, and $\sigma(\mathbf{r}) = \mathcal{I}$.

Given an edge $e = (u, v) \in \mathcal{E}$, its tail and head nodes are given by $\rho(e) = u$ and $\kappa(e) = v$, respectively. For a given layer $\mathcal{N}_t$ ($0 \leq t < \tau$), all outgoing edges are directed to a node in layer $\mathcal{N}_{t+1}$, i.e., $\rho(e) \in \mathcal{N}_t$ iff $\kappa(e) \in \mathcal{N}_{t+1}$. Each edge $e \in \mathcal{E}$ has a label $\theta(e)$ that indicates its associated action. Given two nodes $u \in \mathcal{N}_t$ and $v \in \mathcal{N}_{t+1}$ there is an edge $e = (u, v)$ connecting them iff the action associated to the edge, $a = \theta(e)$, is applicable in $\sigma(u)$ and node v represents the successor state, i.e., $pre(a) \subseteq \sigma(u)$ and $\phi(a, \sigma(u)) = \sigma(v)$.

Thus, an MDD for a task Π is a layered state-transition graph. A node $u \in \mathcal{N}_t$ ($0 \leq t \leq \tau$) is associated to a state that can be reached after applying t actions from the initial state $\mathcal{I}$. Specifically, any path $(e_0, ..., e_t)$ in $\mathcal{M}$ from $\mathbf{r}$ to a node $u \in \mathcal{N}_t$ represents a plan $\pi = (\theta(e_0), ..., \theta(e_{t-1}))$ that starts at $\mathcal{I}$ and reaches state $\sigma(u)$.

The construction of such an MDD is, however, impractical. First, the number of reachable states in a planning task Π can grow exponentially with the number of variables. Moreover, the number of actions needed for any cost-optimal plan is unknown, i.e., the minimum number of layers that is required for its construction is also not available in advance.

We define instead *relaxed MDDs*, which are constructed by imposing an additional limit on the number of nodes per layer, i.e., its width $w(\mathcal{M}) := \max\{|\mathcal{N}_t| : 0 \leq t \leq \tau\}$ is bounded by a given parameter $\mathcal{W}$. To enforce this bound, each node in a relaxed MDD represents an approximation of the *union* of one or more states as opposed to a single state. The edges emanating from a node represent all possible actions that can be applied to the union of the states. Two examples of MDDs are depicted in Figure 2 and construction details are presented in Section 4.

3.1 A Relaxed MDD-based Heuristic

Consider a relaxed MDD $\mathcal{M}$ and a node $u \in \mathcal{N}$. Let $\delta^{in}(u)$ and $\delta^{out}(u)$ be the set of edges directed to and emanating from node u, respectively. An edge e is in $\delta^{out}(u)$ if $pre(\theta(e)) \subseteq \sigma(u)$. Then, the proposition label of node u is defined as

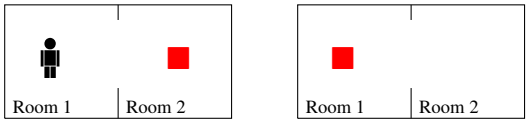
$$\sigma(u) := \bigcup_{e \in \delta^{in}(u)} \phi(\theta(e), \sigma(\rho(e))). \quad (1)$$

Given a planning task Π and a relaxed MDD with width $w(\mathcal{M}) \geq 1$, we can compute the cost to reach each node $u \in \mathcal{N}$ from $\mathbf{r}$, using a shortest path algorithm. Let $\omega^*(u)$ be the minimum cost to reach a node $u \in \mathcal{N}$, with $\omega^*(\mathcal{I}) = 0$. Consider $\mathcal{N}_\mathcal{G} \subseteq \mathcal{N}$ as the set of *goal nodes*, i.e., $u \in \mathcal{N}_\mathcal{G}$ iff $\mathcal{G} \subseteq \sigma(u)$. Then, the relaxed MDD-based heuristic $h^\mathcal{M}$ is given by the minimum cost to reach any goal node:

$$h^\mathcal{M} := \min \{\omega^*(u) : \ u \in \mathcal{N}_\mathcal{G}\}. \tag{2}$$

3.2 Example

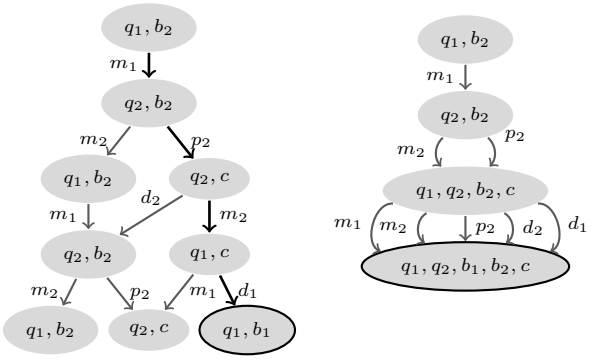
Consider the planning task $\Pi = \langle \mathcal{P}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$ depicted in Figure 1. The set of propositions is $\mathcal{P} = \{q_1, q_2, b_1, b_2, c\}$, where q_i indicates if the robot is in room $i \in \{1, 2\}$, b_i if the block is in room i, and c if the robot is carrying the block. The task has six unit cost actions, $\mathcal{A} = \{m_1, m_2, p_1, p_2, d_1, d_2\}$, where m_1 represents moving the robot from room 1 to room 2, m_2 is the opposite move, and for each $i \in \{1, 2\}$, p_i and d_i correspond to picking-up and dropping the block in room i, respectively. The initial state and goal conditions are illustrated in Figures 1a and 1b, respectively.



(a) Initial State, $\mathcal{I} = \{q_1, b_2\}$. (b) Goals, $\mathcal{G} = \{b_1\}$.

Figure 1: Planning domain description.

Figure 2 shows two MDDs for this planning task, with $\tau = 4$. For each MDD, the edge labels correspond to applicable actions and the nodes denote the set of propositions, as described in equation (1). The first MDD (Figure 2a) has one node per reachable state (i.e., it is an *exact* MDD). The node outlined in black represents a goal node and the bold path corresponds to the shortest path with cost $h^\mathcal{M} = 4$.



(a) Exact MDD, $w = 3$. (b) Relaxed MDD, $w = 1$.

Figure 2: MDDs for the example in Section 3.2.

In the second relaxed MDD (Figure 2b), nodes represent the union of one or more states. In this case, the shortest path in the relaxed MDD reaches a goal node after applying 3 actions, i.e., $h^\mathcal{M} = 3$.

As depicted in Figure 2b, width-one relaxed MDDs have a similar structure to relaxed planning graphs (RPG) (Bonet and Geffner 2000). However, the RPG completely ignores delete effects while our relaxed MDD partially considers them. For example, the second node in the relaxed MDD omits proposition q_1, while the RPG would consider it.

4 Relaxed MDD Construction

We present a top-down algorithm to construct a relaxed MDD for a classical planning task Π. Our construction procedure, presented in Algorithm 1, results in a relaxed MDD with width at most $\mathcal{W}$ and with a finite number of layers.

The top-down procedure is as follows. Starting with a single node in the first layer, $\sigma(\mathbf{r}) = \mathcal{I}$, the procedure iteratively constructs one layer at a time by performing three operations. The first operation, UPDATENODES, updates the nodes in a given layer by computing the set of achievable propositions and calculating the cost to reach the node. Moreover, this step updates the heuristic value if a node is a goal node. The second operation, FINDAPPLICABLEACTIONS, finds the set of actions applicable to the nodes and eliminates any action that does not add any information for the heuristic computation. The procedure creates an edge for each action and directs all edges to a single node in the following layer. Lastly, operation SPLITNODES decides how to partition the incoming edges of the new layer to create at most $\mathcal{W}$ nodes.

Algorithm 1 Relaxed MDD construction

```
1: procedure CONSTRUCTMDD(Input: Π, W)
2:     t = 0, h^M = ∞
3:     repeat
4:         UPDATENODES(N_t, h^M)
5:         FINDAPPLICABLEACTIONS(N_t)
6:         t = t + 1
7:         SPLITNODES(N_t, W)
8:     until TERMINATE(N_t)
9:     return h^M
```

In each iteration, the algorithm checks whether or not we should construct a new layer via the TERMINATE procedure. When the construction is completed, Algorithm 1 returns the heuristic value. The following sections explain each of the procedures presented above.

4.1 Updating Nodes

For a given layer $\mathcal{N}_t$, the procedure updates each node $u \in \mathcal{N}_t$ to represent its set of propositions, $\sigma(u)$, and the minimal cost to reach u. The procedure also updates the heuristic value $h^\mathcal{M}$ if we encounter a goal node.

As described in Section 3, each node $u \in \mathcal{N}$ is associated with a label $\sigma(u)$ that corresponds to the set of propositions that are true in at least one state encoded by u. This label is computed by setting $\sigma(\mathbf{r}) = \mathcal{I}$ and applying recursion (1).

Each node $u \in \mathcal{N}_t$ is also associated with a set of labels that represents the minimum cost to reach u. Inspired by the reachability analysis used in h^{max}, we compute the minimum cost to reach each proposition represented in u.

For each $p \in \sigma(u)$, let $\omega(u,p)$ be the cost to reach proposition p in node u. We associate a cost label $\nu(e,p)$ to each incoming edge e that has proposition p in its resulting state, i.e., $p \in \phi(\theta(e), \rho(e))$. Then, $\omega(u,p)$ is calculated by setting $\omega(\mathbf{r},p) = 0$ to all $p \in \mathcal{I}$, and applying the recursion

$$\omega(u,p) := \min\{\nu(e,p) : \ e \in \delta_p^{in}(u)\}, \tag{3}$$

where $\delta_p^{in}(u)$ represents the set of edges in $\delta^{in}(u)$ that have p in their resulting state.

For a given edge e and a proposition $p \in \phi(\theta(e), \rho(e))$, we calculate $\nu(e,p)$ by considering (i) the cost to apply action $\theta(e)$ and (ii) the cost to have p in the resulting state. Let $\nu(e)$ be the minimum cost of applying action $a = \theta(e)$ in node $v = \rho(e)$. We have that $\nu(e)$ is the cost of action a plus the cost of its most expensive precondition on v, i.e.,

$$\nu(e) := c(a) + \max\{\omega(v,q) : q \in pre(a)\}. \tag{4}$$

Then, for each edge $e \in \delta_p^{in}(u)$, we compute $\nu(e,p)$ as the minimum cost to have p. To do so, we identify two cases. If action $a = \theta(e)$ adds proposition p, then the cost to reach p is given by $\nu(e)$. If a does not add p, the cost of p in the tail node $v = \rho(e)$ might be larger than the cost of any precondition of a in v. In that case, we compute $\nu(e,p)$ by considering the cost of the most expensive associated proposition. The edge cost is hence:

$$\nu(e,p) := \begin{cases} \nu(e) & p \in add(a), \\ \max\{\nu(e), c(a) + \omega(v,p)\} & o.w. \end{cases}$$

To summarize, procedure UPDATENODES iterates over all the nodes $u \in \mathcal{N}_t$ and updates labels $\sigma(u)$ and $\omega(u,p)$ for all $p \in \sigma(u)$.

The procedure will also compute a heuristic estimate whenever a node $u \in \mathcal{N}_t$ is a goal node, i.e., $\mathcal{G} \subseteq \sigma(u)$. Given a goal node u, we compute its minimum cost, $\omega^*(u)$, as the cost to reach its most expensive goal proposition, i.e.,

$$\omega^*(u) := \max\{\omega(u,p) : p \in \mathcal{G}\}. \tag{5}$$

Then, we update the heuristic value $h^{\mathcal{M}}$ as

$$h^{\mathcal{M}} = \min\left\{h^{\mathcal{M}},\ \omega^*(u)\right\}. \tag{6}$$

4.2 Applicable and Essential Actions

For a given layer $\mathcal{N}_t$, the FINDAPPLICABLEACTIONS procedure iterates over each node $u \in \mathcal{N}_t$ to find its applicable actions. The procedure eliminates actions that do not contribute to the computation of the heuristic value and creates an edge for each remaining action.

Given a node $u \in \mathcal{N}_t$, let $A(u)$ be the set of its applicable actions, i.e., $A(u) = \{a \in \mathcal{A} : \ pre(a) \subseteq \sigma(u)\}$. This set can be computed, for instance, by iterating over all actions $a \in \mathcal{A}$ and checking if their preconditions are satisfied.

It is possible to identify if an action $a \in A(u)$ will lead to a state that will be part of the heuristic computation. We denote these actions by $\mathcal{M}$-*essential*.

Definition 4.1. Given a relaxed MDD $\mathcal{M}$ and a node $u \in \mathcal{N}_t$, we say that an action $a \in A(u)$ is $\mathcal{M}$-essential if its successor state $v = \phi(a, \sigma(u))$ satisfies all the following conditions:

(i) State v has not been reached before with less cost, i.e., for each node $u' \in \mathcal{N}_{t'}$ ($t' \leq t$) either $v \not\subseteq \sigma(u')$ or $v \subseteq \sigma(u')$ and $\omega(v,p) \leq \omega(u',p)$ for all $p \in v$.

(ii) State v has a minimum cost less than the current heuristic value, i.e., $c(a) + \max\{\omega(u,p) : p \in pre(a)\} < h^{\mathcal{M}}$.

(iii) State v has a minimum cost less than a given incumbent η^*, i.e., $c(a) + \max\{\omega(u,p) : p \in pre(a)\} < \eta^*$.

We develop a set of rules to identify if an action is $\mathcal{M}$-*nonessential*, i.e., it violates at least one of the conditions in Definition 4.1. Consider a node $u \in \mathcal{N}_t$, an applicable action $a \in A(u)$, and its corresponding edge e. Action a is $\mathcal{M}$-nonessential if any of the following rules hold:

Rule 1. The resulting state adds no new propositions and the cost of each proposition does not decrease, i.e., $\forall p \in add(a) : \ p \in \sigma(u) \ \wedge \ \nu(e,p) \geq \omega(u,p)$.

Rule 2. The minimum cost of the resulting state is higher than the current heuristic value, i.e., $\nu(e) \geq h^{\mathcal{M}}$.

Rule 3. The minimum cost of the resulting state is higher than a given incumbent, i.e., $\nu(e) \geq \eta^*$.

Note that Rules 2 and 3 are direct applications of conditions (ii) and (iii) in Definition 4.1. However, Rule 1 is a necessary, but not sufficient, condition to check if a node has been reached before (i.e., condition (i) in Definition 4.1). The main advantage of these rules, in comparison to the conditions in Definition 4.1, is that we can check them in polynomial time during the construction procedure iterating over each edge only once.

Any action $a \in A(u)$ that satisfies one of the above rules is removed from the set of applicable actions, i.e., $A(u) := A(u) \setminus \{a\}$. After we have checked that each remaining applicable action a in node u is not $\mathcal{M}$-nonessential, we generate a new edge e with label $\theta(e) = a$ that emanates from u and points to node u_0 in the next layer.

4.3 Splitting Nodes

The SPLITNODES procedure is similar to the one used for solving sequencing problems in the literature (Andersen et al. 2007). The procedure, shown in Algorithm 2, splits the nodes in a layer $\mathcal{N}_t$ until it reaches the maximum size $\mathcal{W}$ or there is no more splitting needed.

Algorithm 2 Split states procedure

```
1: procedure SPLITNODES(Input: N_t, W)
2:     if h^M = ∞ and t > 10 and W > 1 then
3:         W = W − 1
4:     Q = {p_1, ..., p_|P|} priority queue
5:     while Q.notEmpty() and |N_t| < W do
6:         p = Q.pop()
7:         for u ∈ N_t do
8:             if δ_p^in(u) = ∅ or δ_p^in(u) = δ^in(u) then
9:                 continue
10:            Create node v and N_t = N_t ∪ {v},
11:            redirect arcs using δ^in(v) = δ_p^in(u) and
12:            δ^in(u) = δ^in(u) \ δ_p^in(u).
13:            if |N_t| = W then break
```

Starting with a layer $\mathcal{N}_t = \{u_0\}$ with a single node, the procedure iteratively splits the nodes such that each resulting node represents fewer states. Specifically, the procedure considers a priority queue of propositions. In each iteration, the procedure chooses a proposition p from the queue (line 6). Then, it iterates over all the nodes $u \in \mathcal{N}_t$ to check if there is any node with incoming edges that can be partitioned such that one partition results in a node with p and the other in a node without p (line 8). In such case, we create a new node v where all edges that represent states where p is true are now directed to v (line 10-12). The procedure ends when there are no more propositions in the queue or we have reached the width limit $\mathcal{W}$.

The priority queue Q divides the propositions into three priority levels. The first level corresponds to goal propositions, i.e., any propositions in $\mathcal{G}$. The second level considers landmark propositions (Hoffmann, Porteous, and Sebastia 2004). We use a simple reachability algorithm to identify propositional landmarks. Finally, the last level corresponds to propositions that are not in the previous levels. Inside each level, we rank the propositions in lexicographical order.

In addition, SPLITNODES checks if the heuristic has been updated (line 2), i.e., if a goal node has been reached in a previous layer. If that is not the case, we reduce the maximum width by 1. The width reduction guarantees the termination of the algorithm (Section 4.4). Our implementation starts the width reduction after 10 layers, which gave the best performance in our testing phase. When $\mathcal{W} = 1$, the construction procedure continues ignoring delete effects.

While there are many ways to split nodes that we intend to investigate in the future, this algorithm has two main advantages. First, this splitting procedure guarantees that there are no two nodes in a layer where one node is a subset of the other. This is due to the fact that we start with a single node and that, in each iteration, we separate the edges according to their propositions. The second advantage is that, if $\mathcal{W}$ is big enough, all nodes will represent a single reachable state.

4.4 Termination Condition

The last component of our top-down construction algorithm, TERMINATE, checks whether we need to create a new layer in our relaxed MDD by observing whether or not procedure FINDAPPLICABLEACTIONS created any new edges.

If the planning task Π is solvable (i.e., a goal state is reachable from $\mathcal{I}$), it is sufficient to check if procedure FINDAPPLICABLEACTIONS created a new edge. Specifically, if the task is solvable, we will eventually reach a goal node, which will make $h^{\mathcal{M}} < \infty$. Since $c(a) > 0$ for all $a \in \mathcal{A}$, Rule 2 (Section 4.2) guarantees that there exists a layer $\mathcal{N}_t$ such that all emanating edges have a cost greater than $h^{\mathcal{M}}$. The same is true if we have an upper bound on the cost-optimal plan ($\eta^* < \infty$) and we use Rule 3 to remove $\mathcal{M}$-nonessential actions.

If the planning task is infeasible, the procedure will still terminates due to the width reduction (Section 4.3). Since our implementation ignores delete effects when we reach $\mathcal{W} = 1$, we can guarantee that there exists a layer $\mathcal{N}_t$ in which $A(u) = \emptyset$ for each $u \in \mathcal{N}_t$.

5 Relationship with Existing Techniques

This section explores the relationship of our relaxed MDD-based heuristic with existing approaches. We start by relating the relaxed MDD structure to a transition graph.

Definition 5.1. (Helmert et al. 2007) A *transition graph* is a 5-tuple $\mathcal{T} = \langle \mathcal{S}, \mathcal{L}, \Sigma, s_{\mathcal{I}}, \mathcal{S}_{\mathcal{G}} \rangle$ where $\mathcal{S}$ is a finite set of states, $\mathcal{L}$ is a finite set of transition labels, Σ is a the set of (labeled) transitions $\Sigma \subseteq \mathcal{S} \times \mathcal{L} \times \mathcal{S}$, $s_{\mathcal{I}}$ is the initial state, and $\mathcal{S}_{\mathcal{G}}$ is the set of goal states $\mathcal{S}_{\mathcal{G}} \subseteq \mathcal{S}$.

Consider a relaxed MDD $\mathcal{M} = (\mathcal{N}, \mathcal{E})$ and a planning task $\Pi = \langle \mathcal{P}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$. Note that a relaxed MDD is in fact a transition graph. Specifically, we can represent a relaxed MDD as a transition graph $\mathcal{T}(\mathcal{M}) = \langle \mathcal{N}, \mathcal{A}, \mathcal{E}, \mathbf{r}, \mathcal{N}_{\mathcal{G}} \rangle$ where the set of states is given by the nodes in $\mathcal{M}$, the set of action corresponds to the labels, the edges define the transitions, and the initial and goal states are given by $\mathbf{r} \in \mathcal{N}$ and $\mathcal{N}_{\mathcal{G}} \subseteq \mathcal{N}$. In particular, each edge $e \in \mathcal{E}$ is associated with the 3-tuple $\langle \rho(e), \theta(e), \kappa(e) \rangle$, which is an element of $\mathcal{N} \times \mathcal{A} \times \mathcal{N}$.

Helmert et al. (2007) define a transition graph induced by a planning task Π as $\mathcal{T}(\Pi) = \langle \mathcal{S}, \mathcal{A}, \Sigma(\Pi), \mathcal{I}, \mathcal{S}_{\mathcal{G}} \rangle$, where $\mathcal{S}$ is the set of states of a planning task, $\Sigma(\Pi)$ represents the set of valid transitions and $\mathcal{S}_{\mathcal{G}}$ is a subset of states such that $s \in \mathcal{S}_{\mathcal{G}}$ iff $\mathcal{G} \subseteq s$. In particular, any transition $\langle s, a, s' \rangle \in \Sigma(\Pi)$ is such that $pre(a) \subseteq s$ and $s' = \phi(a, s)$.

Consider an unbounded (i.e., $\mathcal{W} = \infty$) relaxed MDD $\mathcal{M}^{\infty} = \{\mathcal{N}^{\infty}, \mathcal{E}^{\infty}\}$. The transition graph given by $\mathcal{M}^{\infty}$, $\mathcal{T}(\mathcal{M}^{\infty})$, is in fact a transition graph induced by the planning task Π. Any path from $\mathbf{r}$ to a node $u \in \mathcal{N}_{\mathcal{G}}$ is a valid plan, and the shortest path represents a cost-optimal plan with cost equal to $h^{\mathcal{M}}$.

5.1 Relaxed MDDs and Abstractions

Definition 5.2. (Helmert et al. 2007) An *abstraction* of a transition graph $\mathcal{T}$ is a pair $\langle \mathcal{T}', \alpha \rangle$ where $\mathcal{T}' = \langle \mathcal{S}', \mathcal{L}, \Sigma', s'_{\mathcal{I}}, \mathcal{S}'_{\mathcal{G}} \rangle$ is a transition graph called the *abstract transition graph* and $\alpha : \mathcal{S} \rightarrow \mathcal{S}'$ is a function called the *abstraction mapping*. Specifically, we have that $\langle \alpha(s), a, \alpha(s') \rangle \in \Sigma'$ for all $\langle s, a, s' \rangle \in \Sigma$, $\alpha(s_{\mathcal{I}}) = s'_{\mathcal{I}}$, and $\alpha(s_{\mathcal{G}}) \in \mathcal{S}'_{\mathcal{G}}$ for all $s_{\mathcal{G}} \in \mathcal{S}_{\mathcal{G}}$.

Abstraction-based heuristics are admissible heuristics calculated as shortest paths on an abstract transition graph. Several works have studied different ways to define abstractions (Edelkamp 2001; Helmert et al. 2007) and how to combine them (Katz and Domshlak 2010).

We now show that an MDD relaxation is equivalent to an abstraction of an unbounded MDD. For theoretical purposes, assume that the construction procedure ignores Rule 2 (Section 4.2) and we have an upper bound on the optimal plan cost, η^*. Note that these two requirements do not affect the heuristic computation over a relaxed MDD.

Proposition 5.1. Consider a classical planning task Π. Let $\mathcal{M}^{\infty} = (\mathcal{N}^{\infty}, \mathcal{E}^{\infty})$ and $\mathcal{M} = (\mathcal{N}, \mathcal{E})$ be two relaxed MDDs constructed using Algorithm 1, where $\mathcal{M}^{\infty}$ has an unbounded width, and $\mathcal{M}$ has a maximum width $1 \leq \mathcal{W} < \infty$. For every node $u \in \mathcal{N}^{\infty}_t$ there exists a node $u' \in \mathcal{N}_{t'}$ ($t' \leq t$) such that

$$\sigma(u) \subseteq \sigma(u') \text{ and } \omega(u, p) \geq \omega(u', p) \ \ \forall p \in \sigma(u). \quad (7)$$

Proof. We prove the above statement by induction on the number of layers of $\mathcal{M}^\infty$. Consider the base case where $t = 0$. By construction we have that $\mathcal{N}_0^\infty = \mathcal{N}_0 = \{\mathbf{r}\}$, where $\sigma(\mathbf{r}) = \mathcal{I}$. Thus, condition (7) is satisfied.

Now consider that (7) is valid for all nodes $u \in \mathcal{N}_t^\infty$, for a given $t \geq 0$. Let $v \in \mathcal{N}_{t+1}^\infty$ and $e \in \delta^{in}(v)$ be any edge directed to v. Take $a = \theta(e)$ and $u = \rho(e)$, i.e., $u \in \mathcal{N}_t^\infty$. By hypothesis, there exists a node $u' \in \mathcal{N}_{t'}$ ($t' \leq t$) such that (7) is satisfied for node u. By construction, a is an applicable action on u'. It might be, however, $\mathcal{M}$-nonessential for u'. If $a \notin A(u')$, then Rule 1 (Section 4.2) has to be true and u' satisfies (7) for node v. If $a \in A(u')$, then there exists a node $v' \in \mathcal{N}_{t'+1}$ such that the edge associated to a directs to it. Note that v' satisfies (7) for v due to (4) and (3). □

A direct result of the above proposition is the admissibility of our relaxed MDD based heuristic.

Theorem 5.1. Given a classical planning task Π and a maximum size $\mathcal{W} \geq 1$, Algorithm 1 computes an admissible heuristic $h^{\mathcal{M}}$.

Proof. Consider a relaxed MDD $\mathcal{M} = (\mathcal{N}, \mathcal{E})$ with $1 \leq \mathcal{W} < \infty$ constructed using Algorithm 1 and an unbounded MDD $\mathcal{M}^\infty = (\mathcal{N}^\infty, \mathcal{E}^\infty)$. From Proposition 5.1, for every goal node $u \in \mathcal{N}_\mathcal{G}^\infty$ there exists a goal node $u' \in \mathcal{N}_\mathcal{G}$ such that

$$\omega(u', p) \leq \omega(u', p) \qquad \forall p \in \mathcal{G},$$

and so $\omega^*(u') \leq \omega^*(u)$. Therefore, $h^{\mathcal{M}} \leq h_\infty^{\mathcal{M}} = h^*$, where h^* is the perfect heuristic. □

We now use Proposition 5.1 to create an abstract mapping from an unbounded MDD to a relaxed one, as shown in Proposition 5.2. In other words, we show that the transition graph defined over a relaxed MDD is an abstract transition graph for a planning task Π.

Proposition 5.2. Consider a planning task Π, a relaxed MDD $\mathcal{M} = (\mathcal{N}, \mathcal{E})$ with maximum width $\mathcal{W} \geq 1$, and the transition graph induced by $\mathcal{M}$, $\mathcal{T}(\mathcal{M}) = \langle \mathcal{N}, \mathcal{A}, \mathcal{E}, \mathcal{I}, \mathcal{N}_\mathcal{G} \rangle$. There exists an abstraction mapping α such that $\langle \mathcal{T}(\mathcal{M}), \alpha \rangle$ is an abstraction of $\mathcal{T}(\mathcal{M}^\infty)$, where $\mathcal{M}^\infty = (\mathcal{N}^\infty, \mathcal{E}^\infty)$ is an unbounded MDD for Π.

Proof. We define an abstraction mapping $\alpha : \mathcal{N}^\infty \to \mathcal{N}$ recursively over the layers of $\mathcal{M}^\infty$. We start with $\alpha(\mathbf{r}^\infty) = \mathbf{r}$ and assume that we have defined α for all nodes in layer $\mathcal{N}_t^\infty$. For each node $v \in \mathcal{N}_{t+1}^\infty$ take any incoming edge $e \in \delta^{in}(v)$ and its tail $u = \rho(e)$. Consider $u' = \alpha(u) \in \mathcal{N}$. If there exists an edge $e' \in \delta^{out}(u')$ such that $\theta(e) = \theta(e')$, then $\alpha(v) = \rho(e')$, otherwise $\alpha(v) = u'$.

Due to Proposition 5.1, the abstraction mapping α is such that every goal node $u \in \mathcal{N}_\mathcal{G}^\infty$ is mapped to a goal node $u' \in \mathcal{N}_\mathcal{G}$. Moreover, every transition $\langle u, \theta(e), v \rangle$ defined by an edge $e \in \mathcal{E}^\infty$ has a corresponding transition $\langle \alpha(u), \theta(e), \alpha(v) \rangle$ in $\mathcal{T}(\mathcal{M})$. Note that any transition $\langle \alpha(u), \theta(e), \alpha(v) \rangle$ that defines a self loop (i.e., $\alpha(u) = \alpha(v)$) is not explicitly defined by any edge in $\mathcal{M}$. However, we can extend the set of transitions in $\mathcal{T}(\mathcal{M})$ without impacting the heuristic value. Specifically, we can consider $\mathcal{E}' = \mathcal{E} \cup \mathcal{E}_{loops}$, where every edge in $e \in \mathcal{E}_{loops}$ corresponds to an edge that violates Rule 1 (Section 4.2). □

5.2 $h^{\mathcal{M}}$ vs. h^{max}

We now compare our heuristic with the simplest admissible critical path heuristic, h^{max} (Haslum and Geffner 2000). This heuristic computes the minimum cost to reach each proposition from the initial state. Specifically, consider $h(p)$ as the minimum cost to reach $p \in \mathcal{P}$, and $h(a)$ as the minimum cost to use action a. These values are computed recursively using the formula below and setting $h(p) = 0$ for all $p \in \mathcal{I}$, $h(p) = \infty$ for any $p \notin \mathcal{I}$, and $h(a) = \infty$.

$$h(p) := \min_{a \in \mathcal{A}(p)} \{h(p), h(a)\} \qquad \forall p \in \mathcal{P}$$

$$h(a) := c(a) + \max\{h(q) : q \in pre(a)\} \qquad \forall a \in \mathcal{A}$$

Then, the h^{max} heuristic is define as

$$h^{max} := \max\{h(p) : p \in \mathcal{G}\}$$

Proposition 5.3. Consider a classical planning task Π and a relaxed MDD $\mathcal{M} = (\mathcal{N}, \mathcal{E})$ with a maximum size $\mathcal{W} \geq 1$. Then, $h^{\mathcal{M}} \geq h^{max}$.

Proof. First, consider the following statement:

$$h(p) \leq \omega(u, p) \qquad \forall u \in \mathcal{N},\ p \in \sigma(u) \tag{8}$$

We prove (8) by induction over the layers of $\mathcal{M}$. By construction, (8) holds for $\mathcal{N}_0 = \{\mathbf{r}\}$. Now consider that (8) is true for all nodes $u \in \mathcal{N}_t$ and $p \in \sigma(u)$. Consider a node $v \in \mathcal{N}_{t+1}$ and a proposition $p \in \sigma(v)$. By construction, there exists an edge $e \in \delta^{in}(v)$ such that $\nu(e, p) = \omega(v, p)$. Consider action $a = \theta(e)$ and node $u = \rho(e) \in \mathcal{N}_t$. There are two cases, either $p \in add(a)$ or not. If $p \in add(a)$, then

$$\begin{aligned} \nu(e, p) &= c(a) + \max\{\omega(u, q) : q \in pre(a)\} \\ &\geq c(a) + \max\{h(q) : q \in pre(a)\} \geq h(p) \end{aligned}$$

If $p \notin add(a)$, then $p \in \sigma(u)$. Since $u \in \mathcal{N}_t$, we have $h(p) \leq \omega(u, p)$. Then it follows that

$$h(p) + c(a) \leq \omega(u, p) + c(a) \leq \nu(e, p)$$

Therefore, $h(p) \leq v(e, p)$, which proves (8). Since (8) is true, it follows that $h^{\mathcal{M}} \geq h^{max}$. □

6 Implementation

This section presents how we can exploit the graphical structure given by the relaxed MDD to improve the search procedure. Our approach constructs $\mathcal{M}$ in each state s of the search and uses $h^{\mathcal{M}}$ as an admissible heuristic in a modified A^* search algorithm. Specifically, we add a bounding mechanism to A^*, similar to the branch-and-bound algorithm used in Integer Programming (IP) solvers. To do so, we use $\mathcal{M}$ to find feasible plans while computing $h^{\mathcal{M}}$. The following sections explain how we can find a feasible plan using a relaxed MDD and how the cost of this plan enhances the A^* search algorithm.

6.1 Finding Feasible Plans in a Relaxed MDD

Our implementation considers two different procedures to find a feasible plan using the relaxed MDD graphical structure. The first procedure extracts a relaxed plan, denoted by

π^h, with equal cost to the heuristic value and checks its validity. The second approach selects a subset of nodes from the relaxed MDD that represent single states and uses them to find a valid plan π^b. For both procedures, plan extraction and validation occur after the relaxed MDD construction.

Consider a relaxed MDD $\mathcal{M}$ for a state s with at least one minimum cost goal node $u_{\mathcal{G}}$. We follow the edges of $\mathcal{M}$ backward to find a path from $u_{\mathcal{G}}$ to s. The resulting path is a relaxed plan, π^h, that has the same cost as our heuristic $h^{\mathcal{M}}$. If π^h is a valid plan, we create a plan π that is valid for the planning task. Consider $\pi^{\mathcal{I}}$ as the plan from $\mathcal{I}$ to state s given by the search algorithm. Then, we create a feasible plan π concatenating $\pi^{\mathcal{I}}$ and π^h, i.e., $\pi = (\pi^{\mathcal{I}}, \pi^h)$.

The second method allocates a fixed number of nodes, $\mathcal{W}^e \leq \mathcal{W}$, in each layer of $\mathcal{M}$ to participate in the extraction of a valid plan. For each layer $\mathcal{N}_t$, let $\mathcal{N}_t^e$ be a set of nodes that represent a single state (i.e., *exact* nodes), and $\mathcal{N}_t^r$ be a set of node that represent the union approximation of multiple states (i.e., *relaxed* nodes), where $\mathcal{N}_t = \mathcal{N}_t^e \cup \mathcal{N}_t^r$. Specifically, we modify SPLITNODES such that in each layer $\mathcal{N}_t$ we arbitrarily select $\mathcal{W}^e$ edges emanating from nodes $u^e \in \mathcal{N}_t^e$ to be the exact nodes in $\mathcal{N}_{t+1}^e$. If an exact node u^e is a goal node, then we extract a plan taking any path from $\mathbf{r}$ to u^e. Since all parent nodes of an exact node are exact, we can guarantee that the extracted plan is valid. As previously, we generate a valid plan for the planning task by concatenating the extracted plan π^b with $\pi^{\mathcal{I}}$, i.e., $\pi = (\pi^{\mathcal{I}}, \pi^b)$.

While having more exact nodes increases the chances of finding a feasible plan π^b, the heuristic quality can be negatively affected. Since the maximum width does not change, the union approximation of the relaxed nodes is weaker. Hence, we use the second method only to find a first feasible plan.

Note that whenever we find a valid plan π (created with either π^h or π^b), we can use its cost as an upper bound η^* in the construction procedure. Specifically, for a state s in the search, the value of η^* in Rule 3 (Section 4.2) is set to $c(\pi) - c(\pi^{\mathcal{I}})$, where $\pi^{\mathcal{I}}$ is the plan to reach s from $\mathcal{I}$.

6.2 Exploiting Upper Bounds in A^*

To take advantage of the information represented by the MDD, we propose a modified A^* search algorithm that considers the cost of feasible solutions. In particular, our approach is inspired by the branch-and-bound algorithm implemented in IP solvers. Branch-and-bound uses a linear programming (LP) relaxation as an admissible heuristic to guide the search. Whenever the LP relaxation gives an integer solution, the algorithm prunes any node in the search for which the LP relaxation provides a cost greater than the upper bound. Similarly to the branch-and-bound algorithm, our approach uses feasible extracted plans to create an upper bound and prune states in the search space.

We incorporate this idea in A^*, proposing a variant that we call A^*_{BB}. In every expanded state, A^*_{BB} checks the feasibility of a relaxed plan calculated by a relaxed MDD $\mathcal{M}$. The cost of such a valid plan plus the cost of reaching the state is an upper bound on the cost of the optimal solution.

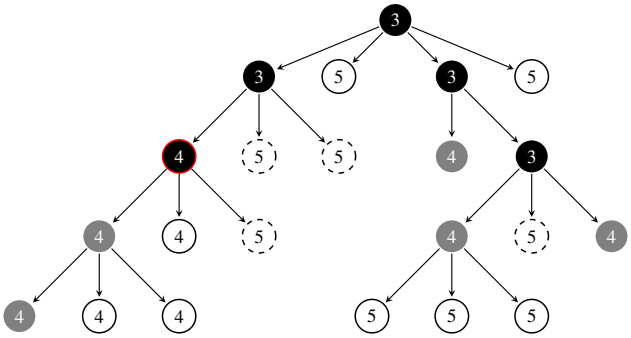


Figure 3: Nodes explored by A^* (in black and gray) and A^*_{BB} (in black). Dashed nodes are states that do not need to be inserted in the search queue by A^*_{BB}. Nodes circled in red are the ones with a feasible plan π^h. Node labels indicate the f-value of a state.

In A^*, states with a f-value[1] strictly greater than the optimal solution are never expanded, thus, the benefit of pruning states with a greater f-value than the upper bound is limited to memory saving during search.

The real advantage of A^*_{BB} arises when $\mathcal{M}$ extracts a feasible plan π^h with the same cost as $h^{\mathcal{M}}$ for a state s. If s is retrieved from the open list, the search terminates and the minimum-cost plan is $\pi = (\pi^{\mathcal{I}}, \pi^h)$. The termination criteria is correct since s has the minimum f-value among the states in the list, i.e., $f = h^{\mathcal{M}} + g$ is a lower bound for the minimum-cost plan. Since $\pi = (\pi^{\mathcal{I}}, \pi^h)$ is a valid plan with cost equal to f, this proves that π is a minimum-cost plan. Notice that A^*_{BB} can be used with any admissible heuristic (consistent or not) that has a plan extraction procedure.

Therefore, A^*_{BB} may avoid expanding states with an f-value equal to the optimal solution, while A^* would need to explore them. Figure 3 shows an example of the difference in states expanded by A^* and A^*_{BB}. Of course, in the worst case, A^*_{BB} will still look at the same number of states as A^*.

7 Preliminary Results

We now present an empirical analysis on the relaxed MDD heuristic using the LPRGP planning system (Coles et al. 2008). We experiment with both A^* and A^*_{BB} algorithms, where ties are broken preferring higher h-values. We consider three variations of relaxed MDDs, where we limit the maximum width to 256, 512 and 1024, respectively. This analysis includes a comparison between our heuristics and h^{max} and the operator counting heuristic h^{oc} (Pommerening et al. 2014). We implemented a STRIPS version of h^{oc} with landmarks and state equation constraints. The LP models are solved using CPLEX v12.7. All experiments are run on a Xeon 3.5GHz processor machine, with a 2 GB memory limit and a 30 minute time limit.

We selected 6 domains with positive action costs, from the last two International Planning Competitions (IPCs): *no-mystery*, *wood-working*, *floortile*, *tetris*, *transport*, and *visit-all*. *No-mystery* and *visit-all* have unary action costs, while

[1]We assume the usual heuristic search notation: $f = h + g$.

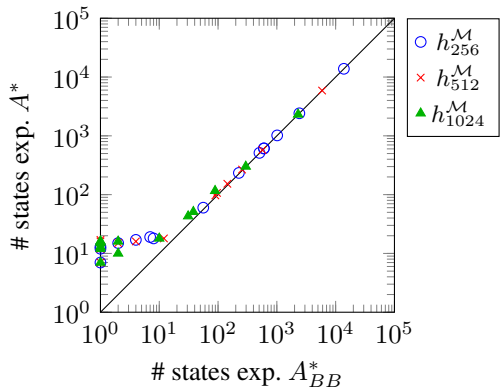


Figure 4: Number of states expanded using A^*_{BB} and A^*.

the other domains feature non-uniform action costs. These domains were chosen due to their range in difficulty and to illustrate the strong and weak aspects of our approach.

7.1 A^* vs. A^*_{BB}

We compare A^* with A^*_{BB} using our relaxed MDD heuristics $h^{\mathcal{M}}_{256}$, $h^{\mathcal{M}}_{512}$ and $h^{\mathcal{M}}_{1024}$ (with $\mathcal{W} \in \{256, 512, 1024\}$, respectively).

Figure 4 shows the number of expanded states (logarithmic scale) for each search algorithm and heuristic. A point in the plot represents an instance and its (x, y) coordinate the number of states expanded by A^*_{BB} and A^*, respectively. Figure 4 shows that A^*_{BB} expands fewer (or equal) number nodes than A^* in all instances, especially for instances that need a small number of expansions to find the cost-optimal plan. In fact, on average A^*_{BB} reduces the number of expanded nodes by 1%, 2% and 6% when using heuristic $h^{\mathcal{M}}_{256}$, $h^{\mathcal{M}}_{512}$ and $h^{\mathcal{M}}_{1024}$, respectively. Similar results are found for number of states evaluated, where A^*_{BB} decreases the number of states evaluated by 1%, 4% and 12% when $h^{\mathcal{M}}_{256}$, $h^{\mathcal{M}}_{512}$ and $h^{\mathcal{M}}_{1024}$ are used, respectively.

As expected, the benefit of using A^*_{BB} is more prominent when using a bigger width. A larger relaxed MDD is more likely to provide valid relaxed plans.

7.2 $h^{\mathcal{M}}$ vs. Existing Techniques

We now compare the performance of our proposed relaxed MDD heuristics against h^{max} and h^{oc}. We use A^*_{BB} as the search algorithm for the $h^{\mathcal{M}}$ heuristics, while we employ A^* for both h^{max} and h^{oc}. Table 1 shows the number of instances that each approach solves to optimality (*# Optimal plans*) and the number of instances for which a relaxed MDD heuristic finds a feasible plan (*# Valid plans*). It should be noted that, due to the nature of A^*, on the problems for which h^{max} and h^{oc} fail to find optimal solutions, they also do not find feasible solutions.

With respect to the number of optimal plans, h^{oc} achieves the best coverage, followed by h^{max} and the $h^{\mathcal{M}}$ heuristics. In particular, $h^{\mathcal{M}}_{256}$ performs best among the relaxed MDD heuristics, finding an optimal plan on 15 instances.

Table 1: Coverage performance.

		# Optimal plans					# Valid plans		
	#	$h^{\mathcal{M}}_{256}$	$h^{\mathcal{M}}_{512}$	$h^{\mathcal{M}}_{1024}$	h^{oc}	h^{max}	$h^{\mathcal{M}}_{256}$	$h^{\mathcal{M}}_{512}$	$h^{\mathcal{M}}_{1024}$
floortile	20	0	0	0	**2**	**2**	19	**20**	**20**
no-mystery	20	8	8	8	**15**	7	10	10	**11**
tetris	20	3	3	3	**13**	5	**15**	14	**15**
transport	20	1	0	0	1	**5**	12	**15**	14
visit-all	20	1	1	1	**6**	0	8	11	**13**
wood-working	20	2	2	2	**5**	2	**19**	**19**	**19**
TOTAL	120	15	14	14	**42**	21	83	89	**92**

To understand these results, Table 2 compares the run time and number of states expanded over the instances that all heuristics solve to optimality. The symbol # indicates the number of instances considered. We can see that the $h^{\mathcal{M}}$ heuristics have the highest average run time. However, we observe an opposite trend in terms of the number of states expanded: all $h^{\mathcal{M}}$ heuristics expand orders of magnitude fewer states than h^{max} and a similar number as h^{oc}. The only exception is *wood-working*, where $h^{\mathcal{M}}$ expands significantly fewer states than h^{oc}.

Table 2: Average run time and states expanded.

		Average run time (sec)				
	#	$h^{\mathcal{M}}_{256}$	$h^{\mathcal{M}}_{512}$	$h^{\mathcal{M}}_{1024}$	h^{oc}	h^{max}
no-mystery	7	20.5	24.8	27.9	**0.6**	49.0
tetris	3	45.3	55.4	69.4	**1.1**	3.1
wood-working	2	307.0	192.1	97.0	**32.2**	223.9
		Average # states expanded				
no-mystery	7	35.6	15.1	**6.4**	45.6	35053.6
tetris	3	360.7	193.7	99.7	**33.0**	6326.0
wood-working	2	553.5	117.0	**20.5**	2238.5	97394.0

While relaxed MDD-based heuristics seem to be highly informative, their computational cost is currently too high to make them competitive with state-of-the-art heuristics.

We also point out the strength of our approach to find valid plans. As shown in Table 1, all relaxed MDD-based heuristics have a high coverage when finding a valid plan. Specifically, our approach has an exceptional performance finding feasible plans in *floortile*, the only domain where none of the $h^{\mathcal{M}}$ heuristics found an optimal solution.

With respect to solution quality, Table 3 shows the mean relative error (MRE) for the best feasible solution found by each MDD-based heuristic. We compute the MRE for instances where all relaxed MDD heuristics found a feasible plan. For a given heuristic and instance, we compute the relative error as $(UB - LB)/UB$, where UB is the best incumbent found the heuristic and LB is the best known lower bound, i.e., either the optimal solution or the best heuristic value in the initial state. The table shows that $h^{\mathcal{M}}_{1024}$, on average, finds the best quality plan. However, on most domains, the feasible plans are still quite far from optimal.

8 Conclusions and Future Works

This work presents a new heuristic to solve cost-optimal classical planning problems based on relaxed multivalued

Table 3: Mean Relative Error for all domains.

Domain	#	$h^{\mathcal{M}}_{256}$	$h^{\mathcal{M}}_{512}$	$h^{\mathcal{M}}_{1024}$
floortile	19	0.61	**0.58**	0.59
no-mystery	10	0.04	0.06	**0.03**
tetris	14	**0.18**	**0.18**	**0.18**
transport	12	0.63	0.59	**0.55**
visit-all	8	**0.31**	0.46	0.46
wood-working	19	0.25	0.25	**0.22**
All instances	82	0.36	0.36	**0.35**

decision diagrams (MDDs), a graphical structure that provides an adjustable approximation of the state-space transition graph. We present an algorithm that constructs relaxed MDDs and calculates an admissible heuristic. Moreover, we show how to exploit the graphical structure to find valid plans and enhance an A^* search algorithm by considering upper bounds. We relate this graphical structure to transition graphs and show that a relaxed MDD is an abstraction of the state transition graph. Moreover, we show that our heuristic is strictly more informative than the h^{max} heuristic.

Preliminary results in a subset of IPC domains show that relaxed MDD heuristics can considerably reduce the number of states expanded during search. However, the effort to compute a relaxed MDD currently makes the approach uncompetitive.

Future directions include an extension of our framework to SAS+ planning and a more in-depth study of the relationship between relaxed MDD and abstractions. In particular, we want to exploit MDDs to represent projections and combine them using a Lagrangian decomposition method (Fisher 2004) similarly to the cost-partition framework (Katz and Domshlak 2010).

References

Andersen, H. R.; Hadzic, T.; Hooker, J. N.; and Tiedemann, P. 2007. A constraint store based on multivalued decision diagrams. In *CP 2007*. Springer. 118–132.

Bergman, D.; Cire, A. A.; van Hoeve, W.-J.; and Hooker, J. N. 2016. *Decision Diagrams for Optimization*. Springer International Publishing, 1 edition.

Blum, A. L., and Furst, M. L. 1997. Fast planning through planning graph analysis. *Artificial intelligence* 90(1):281–300.

Bonet, B., and Geffner, H. 2000. Planning as Heuristic Search: New Results. *Recent Advances in AI Planning* 1809:360–372.

Bonet, B., and Geffner, H. 2001. Planning as Heuristic Search. *Artificial Intelligence* 129(February 2000):5–33.

Coles, A.; Fox, M.; Long, D.; and Smith, A. 2008. A hybrid relaxed planning graph-lp heuristic for numeric planning domains. In *ICAPS 2008*, 52–59.

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

Edelkamp, S.; Kissmann, P.; and Torralba, Á. 2012. Symbolic a* search with pattern databases and the merge-and-shrink abstraction. In *ECAI 2012*, 306–311.

Edelkamp, S. 2001. Planning with pattern databases. In *ECP 2001*, 13–24.

Fisher, M. L. 2004. The lagrangian relaxation method for solving integer programming problems. *Management science* 50(12):1861–1871.

Geißer, F.; Keller, T.; and Mattmüller, R. 2016. Abstractions for planning with state-dependent action costs. In *ICAPS 2016*, 140–148.

Haslum, P., and Geffner, H. 2000. Admissible heuristics for optimal planning. In *AIPS 2000*, 140–149.

Helmert, M., and Domshlak, C. 2009. Landmarks, Critical Paths and Abstractions: What's the Difference Anyway? In *ICAPS 2009*, 162–169.

Helmert, M.; Haslum, P.; Hoffmann, J.; et al. 2007. Flexible abstraction heuristics for optimal sequential planning. In *ICAPS 2007*, 176–183.

Hoda, S.; Van Hoeve, W.-J.; and Hooker, J. N. 2010. A systematic approach to MDD-based constraint programming. In *CP 2010*. Springer. 266–280.

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

Hoffmann, J.; Porteous, J.; and Sebastia, L. 2004. Ordered Landmarks in Planning. *Journal of Artificial Intelligence Research* 22:215–278.

Katz, M., and Domshlak, C. 2010. Optimal admissible composition of abstraction heuristics. *Artificial Intelligence* 174(12-13):767–798.

Katz, M.; Hoffmann, J.; and Domshlak, C. 2013. Red-black relaxed plan heuristics. In *AAAI 2013*, 489–49.

Keller, T.; Pommerening, F.; Seipp, J.; Geißer, F.; and Mattmüller, R. 2016. State-dependent cost partitionings for cartesian abstractions in classical planning. In *IJCAI 2016*, 3161–3169.

Kinable, J.; Cire, A. A.; and van Hoeve, W.-J. 2017. Hybrid optimization methods for time-dependent sequencing problems. *European Journal of Operational Research* 259(3):887 – 897.

Pommerening, F.; Röger, G.; Helmert, M.; and Bonet, B. 2014. LP-based heuristics for cost-optimal planning. In *ICAPS 2014*, 226–234.

Sievers, S.; Wehrle, M.; and Helmert, M. 2014. Generalized Label Reduction for Merge-and-Shrink Heuristics. In *AAAI 20014*, 2358–2366.

Torralba, Á.; Linares López, C.; and Borrajo, D. 2016. Abstraction heuristics for symbolic bidirectional search. In *IJCAI 2016*, 3272–3278.

Torralba, Á.; López, C. L.; and Borrajo, D. 2013. Symbolic merge-and-shrink for cost-optimal planning. In *IJCAI 2013*.

Application of MCTS in Atari Black-box Planning

Alexander Shleyfman
Technion, Haifa, Israel
alesh@campus.technion.ac.il

Alexander Tuisov
Technion, Haifa, Israel
alexandt@campus.technion.ac.il

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

Abstract

Action selection in environments where the problem structure is hidden by an action simulator presents a challenge for domain-independent action planning. Using the Arcade Learning Environment (ALE) that supports Atari 2600 games, recent research on the subject led to several planning algorithms suitable for this challenging settings. The most competitive of this set of algorithms are variants of best-first search with action pruning based on the properties of the states already generated by the simulator. Pushing the envelope of domain-independent planning with simulators, we show that a different family of algorithms, one that plans for a bounded-length trajectories and not only for the next action to apply, allows solving problems that so far were out of our reach. In particular, we present a family of such Monte-Carlo Tree Search algorithms that favorably compete with its state-of-the-art counterparts. Likewise, noticing that the two algorithmic approaches are rather complementary, we examine both a pre-sampling based selection among the two, as well as an alternating composition of the algorithms, and show that they favorably compete with both of their individual components.

Introduction

Popular in the 80s, recently Atari 2600 games once again became a center of attention, but now for an entirely different reason. In 2013, Bellemare *et al.* (2013) introduced the Arcade Learning Environment (ALE) – a convenient platform for domain-independent planners and learners with an accessible interface to numerous Atari video games. In these black-box planning domains, the set of actions, the vector state, and the objective function are fully observable, and all actions have a deterministic effect. At the same time, the action and reward dynamics are inaccessible, and given only via a simulator.

While making the problems closer to many challenges of the real-world applications, these traits of the ALE setup also prevent one from using techniques that have been developed over the years for search in declaratively represented domains, such as planning as heuristic search or planning as satisfiability (Russell and Norvig 2010; Geffner and Bonet 2013). At the high level, this leaves us only with variants of the brute-force search methods such as breadth-first search (BrFS), as well as with sampling-based Monte-Carlo Tree Search (MCTS) algorithms.

The first evaluation of such techniques on ALE was done by Bellemare *et al.* (2013), and it showed that UCT (Kocsis and Szepesvári 2006), one of the most popular MCTS algorithms, substantially outperforms BrFS on a wide set of Atari 2600 games. Recently, however, Lipovetzky *et al.* (2015) and Shleyfman *et al.* (2016) showed that some more sophisticated variants of breadth-first search, namely Iterative Width (IW(i)) and Prioritized Iterative Width (p-IW(i)), respectively, exhibit significantly better performance than UCT. Both these algorithms exploit state pruning that focuses the search only on states which are "novel" with respect to the previously discovered states, with the difference between the two being primarily in the way the state novelty is defined.

The results of Lipovetzky *et al.* (2015) and Shleyfman *et al.* (2016) positioned the breadth-first search algorithms as the tools of choice for problem setups like ALE, and somewhat pushed MCTS into the shadow here. However, a closer look at the empirical results reveal that the superior performance of IW(i) and p-IW(i) over UCT was not uniform across the different Atari 2600 games, and in fact, the two sets of tools could have been seen as complementary in terms of games coverage. Furthermore, several works have shown, both formally and empirically, that UCT is not necessarily the most effective MCTS algorithm available (Coquelin and Munos 2007; Bubeck *et al.* 2009; Feldman and Domshlak 2014; 2013).

Given the above, in this work we examine whether the effectiveness of MCTS techniques in the ALE environment can be pushed substantially further, and provide an affirmative answer to this question, even using relatively straightforward techniques. Specifically, following the path taken by Shleyfman *et al.* (2016), we consider action selection in ALE as a multiarmed bandit style competition between the actions available at the current state. However, in contrast to the work of Shleyfman *et al.*, the competition here is done at the level of action sequences, and thus the planning is done not for the next "best" state, but for some time epoch of a preset length. We show that this approach both dominates UCT, as well as favorably competes with p-IW(1). Furthermore, we show that this approach is complementary to the breadth-first variations, and explore some techniques that successfully combine the two, either by selecting the more appropriate method to the task at hand, or alternating

between the two approaches.

Background

The ALE problem, as it was formalized by Jinnai and Fukunaga (2017), is a tuple $\langle V, A, f, s_0, r \rangle$, where:

- $V = \{v_1, \ldots, v_n\}$ is a finite set of state variables with finite domains $\mathcal{D}(v_i)$, and each state is represented by a complete assignment to these variables (the variable/value pairs are written as $v_i = d$, and sometimes referred to as *facts*);
- A is a finite set of actions, with all actions being applicable in all states;
- f is a deterministic transition function represented by a simulator, with $f(a, s)$ being the state of the game that follows the application of an action a in the state s;
- s_0 is a starting state; and
- r is a real-valued reward function, with $r(s)$ being the reward obtained by applying (any) action in state s.

This setting is somewhat similar to the classical planning in the sense that the current state of the game starting with s_0 and actions A are known to the agent. However, both the transition and reward functions are initially hidden, and are gradually revealed with the search progress through interactions with the simulator: upon simulation of applying action a in state s, the resulting state $f(a, s)$ and the reward $r(f(a, s))$ are being revealed to the planner. Following Shleyfman *et al.* (2016), a cumulative reward $R(s)$ of a state s is recursively defined as $R(s') = R(s) + \gamma^d r(s)$, where s' is the unique parent state of s, γ is a discount factor, and d is the depth of s in the search tree.

P-IW(1)

One of the more prominent algorithms for black-box planning is P-IW(1). It is a regular breadth-first search with a following modifications: When a state s is generated it is assigned a novelty value. A state is declared novel if it has at least one fact $v_i = d$ in s was not previously observed as an element of some state s'' which had a higher cumulative reward than s. If a state is not novel, it is pruned from the search tree. Ties in the BrFS queue are broken by the cumulative reward of the states belonging to the nodes in question. The action a chosen to be applied in s_0 is the action on a path to the leaf node with a state that yields highest cumulative reward. Novelty-based algorithms however often explore the search space in a highly imbalanced manner, often leading to "single tunnel" phenomenon, where the search tends to prune all but one path to the deeper parts of the search tree, and the comparison between different actions close to the root becomes skewed. P-IW(1), although slightly alleviates the problem, is no stranger to this drawback either.

UCT

Another algorithm that had been put to the test in aforementioned setting is UCT, a MCTS-based algorithm originally created for MDP planning by Kocsis and Szepesvári (2006) and adapted for ALE by Bellemare *et al.* (2013). UCT explores the search space by growing a search tree in a manner that treats every search tree node as an independent multi-armed bandit problem (MAB). The algorithm makes use of the UCB1, constructing a statistical confidence intervals for each of the arm in the search node. The UCT algorithm, however, treats each arm optimistically, i.e., evaluating each arm j only by the left part of the interval, resulting in the formula:

$$x_j + \sqrt{2(\ln n)/n_j},$$

where x_j is the the mean payout for arm j, n_j is the number of plays of arm j, and n is the total number of plays from the current node. The strategy of UCT is to pick the arm with the highest upper bound each time. In the *selection* phase, the algorithm moves down on the tree nodes, using the statistics necessary to treat each position as a MAB. This phase lasts until the algorithm reaches a tree leaf. The phase of *expansion* occurs when UCB1 no longer applies. An unvisited child position is randomly chosen, and a new record node is added to the search tree. And after that come the *simulation* and *back-propagation* phases. These are typically done by Monte Carlo simulation (until the algorithm reaches some preset horizon), than averaging on the result of this simulation, correspondingly.

Algorithm stops at a leaf node, and expands it. It then applies a random simulation (rollout) of a certain length to obtain a score estimate for all its successors, and back-propagates the result. It is important to note that the UCT procedure saves in the memory only the search tree built so far, and does not save any simulated rollouts. ALE problems, however, differ from the MDP and game tree problems UCT was originally developed for. ALE environment is deterministic, thus any reward observed is deterministically achievable. This fact shifts our interest from finding best average reward to finding a maximal reward. Moreover, UCB1 formula minimizes the cumulative regret, where ALE design and dynamics suggest one should try and minimize simple regret (the difference between best trajectory found, and best trajectory there is), closer to the setting presented by Schulte and Keller (2014).

Tree Search Algorithms

In this section, we describe the MCTS family of algorithms in general, and approaches we used in particular. Since the MCTS algorithms were not originally developed for the ALE setting, description of adaptations made to fit MCTS to the problem at hand also follows.

MCTS family of algorithms for game trees as presented by Chaslot *et al.* (2008) has the following structure: it consists of four steps, namely selection – traversing the search tree from the root until leaf node using *selection strategy*, expansion – storing some descendants of the chosen node using *expansion strategy*, simulation – evaluation of the leaf node using *simulation strategy*, and back-propagation – updates the value of the nodes based on the results of the simulation using *back-propagation strategy*. These steps are repeated as long as the resources allowed per decision aren't exhausted.

Each of these steps has its own strategy, and to describe an MCTS algorithm it is sufficient to describe these four strategies. It is worth noting that the application of the generalization of the MCTS methods – Trial-Based Heuristic Tree Search (THTS) (Keller and Helmert 2013) had been explored in the context of classical planning (Schulte and Keller 2014), which also features deterministic actions, but differs from our setting.

1. In the deterministic setting back-propagation strategy is pretty straightforward. We propagate back the reward of the best trajectory in every algorithm, where by "trajectory" we mean a sequence of consequently applied actions, and by "best" we mean leading to an end state (determined by a preset horizon) with highest cumulative reward. Since the ALE setting is deterministic, every trajectory can be followed through, and this cumulative reward is achievable at the execution. Therefore, there is no need to consider sub-optimal trajectories after the exploration is finished. This method of back-propagation was discussed in the work of Schulte and Keller (2014) (albeit in a cost rather than a reward setting). Back-propagation strategies will not be discussed further.
2. Selection strategy – deviating from MCTS scheme, we permit selecting non-leaf nodes. Moreover, following the work of Tolpin and Shimony (2012) on MCTS algorithms, in some of the following methods we decouple selection in the root node from selection in the rest of the tree. The intuition for this approach is derived from the game setting, where we apply actions step-by-step, and not fully execute the best trajectory found by the algorithm.
3. Simulation and Expansion strategies – in the deterministic setting of ALE, there is no much point in generating (via simulator) already existing states, thus we combine both these strategies into one. This will result in some nodes in the search tree, where not all the immediate successors had been generated. This approach is not implemented in the UCT algorithm, which "spends" most of the allocated budget on simulating rollouts that will add one tree node each, resulting in a much more shallow search tree.

   Simulating a result here means sampling a trajectory from the trajectory space via Monte-Carlo procedure. Since we would like to compare trajectories of an equal length (otherwise we might create a bias towards longer, but not necessarily better trajectories, or, as in games with negative rewards, shorter but less informed trajectories), we opt to simulate the result only until a certain planning horizon, which is also a maximal trajectory length. In what follows, simulation of trajectory or a trajectory itself may be referred as rollout.

Using this formulation we describe the family of algorithms presented below in terms of the selection and simulation strategies they employ.

UNIFORM

The first, and the most trivial approach, UNIFORM is based upon a uniform selection of the immediate successor of the root. The simulating strategy of applying uniformly selected random actions until it reaches maximal trajectory length l[1]. Given the fact that in the ALE setting, there is a fixed predetermined number of actions in each state (18, to be exact), it is equivalent to picking uniformly at random a trajectory of length l. This approach, however, suffers from some drawbacks. For example, it seems to be "wasting" numerous simulations on creating trajectories from less rewarding (at least so far) children of the root. So, it would seems natural to employ some gradual "candidate rejection" technique.

SEQHALVING

The next approach, SEQHALVING, tries to tackle this problem. SEQHALVING has the same simulation strategy as UNIFORM, but it employs more sophisticated selection strategy. It approaches the selection problem as multi-armed bandit, where the children of the root node are considered bandit arms, and the act of creating a rollout from a node is parallel to sampling a bandit arm. Now one could use an apparatus created for solving MAB problem with fixed budget (since our budget is known in advance to the decision making process), namely SEQUENTIALHALVING technique suggested by Karnin *et al.* (2013). It operates as described in Algorithm 1.

Algorithm 1 SEQHALVING

initialize $T \leftarrow$ total budget
initialize $S_0 \leftarrow$ set of all children of the root node
initialize $n \leftarrow |S_0|$
for $r = 0$ **to** $\lceil log_2 n \rceil - 1$ **do**
    make rollout from every node $i \in S_r$ for

$$\left\lfloor \frac{T}{|S_r| \lceil log_2 n \rceil} \right\rfloor$$

    times
    let S_{r+1} be the set of $\lceil |S_r|/2 \rceil$ successors of the root with the highest maximal rewards
return the subtree rooted at the first action in the best trajectory

On one hand, this technique alleviates the aforementioned problem. On the other hand, the exploration dynamics of the nodes beyond the immediate successors of the root are still unbiased, and contain no elements of exploitation of any previously found high-reward nodes.

ϵ-GREEDY

As discussed before, it may be beneficial to try and shift the dynamics towards exploitation. In the following approach, ϵ-GREEDY selection strategy chooses a node leading to the best trajectory with some probability $0 < \epsilon < 1$, and selects a node uniformly otherwise. Simulation strategy follows a similar principle. The first node of the rollout (if non-leaf)

[1]Note that here there is no difference was the selected node previously was previously generated or not. Previously generated node, however, will not spend the allocated budget, since the state is already part of the search tree.

and encountering a non-leaf node yielding a positive reward are decision points. At any decision point, the simulation follows previously simulated trajectory with probability ϵ until the next decision point (or until the planning horizon if no more decision points are present). With probability $1-\epsilon$, or when in a leaf node, the algorithm chooses the next action uniformly. All children of a node that have not been previously generated are counted as if they were yielding a reward of $-\infty$.

Reasoning

The family of the aforementioned algorithms have at least two reassuring properties. First, given a fixed amount of always applicable actions (as is the case in ALE setting), each trajectory of a preset length can be generated uniformly. Thus, the expectation of the value of the recommended trajectory is monotonic in budget, which is a property novelty-based algorithms such as P-IW(1) don't possess. Second, the probability of recommending the best trajectory for a given planning horizon converges to 1, given a sufficiently large budget. This is also not true for novelty-based algorithms, because of the pruning procedure involved. Note, however, that the optimality of the trajectory is judged with regard to the fixed horizon l, and not for the whole game.

Experimental Results

Out of the presented family of MCTS algorithms we choose to evaluate UNIFORM, SEQHALVING, and ϵ-GREEDY. We are evaluating them against two of the most prominent state-of-the-art algorithms P-IW(1) and UCT. As mentioned before, the testbed for these evaluations will be the games of the ALE setting by Bellemare *et al*. (2013). As in previous works, we exclude two games: SKIING game was already left out in the experiments of Lipovetzky *et al.* (2015) due to certain issues with the reward structure of this game[2], and BOXING since by Shleyfman *et al.* (2016), the game boils down to striking in arbitrary directions since the second player is inactive, thus every algorithm trivially scores the possible maximum. This leaves us with 53 of the 55 different games. We also contemplated to leave out the SPACE INVADERS games, since it may be flawed, and terminates due to some inner bug. However, we decided to leave it, since all the algorithms compete in the same conditions.

We have implemented our algorithms on top of the implementation of Lipovetzky *et al.* (2015), with the addition of P-IW(1) by Shleyfman *et al.* (2016). The implementation of the UCT algorithm was provided by Bellemare *et al.* (2013). Its exploration constant here is set to 0.1. In each decision point, the algorithm normalizes its rewards according to the first reward it has found.

In our experiments, we use the setting of frames reuse proposed by Bellemare *et al.* (2013). In this setting the frames in the sub-tree of the previous lookahead provides the algorithms with "additional" simulations, since there is no point in re-generation of already existing states. In their works Lipovetzky *et al.* (2015) and Shleyfman *et al.* (2016) used the a lookahead budget of 150000 simulated frames (or, equivalently, 30000 search nodes), with time limit of 18000 frames (5 minutes) for each game. This, however, seems unpractical to us, since the duration of full simulation process of a "five-minute-real-time" game may, in practice, take more than three days of computation time. Therefore, we limit our simulation budget to 50000 frames (or, equivalently, 10000 search nodes). It is worth noting that even after this limitation, the evaluation process is still extremely demanding to computational resources, which severely limits our ability to thoroughly check many configurations without compromising on the variance of the results.

The lookahead depth was limited to 1500 frames (300 search nodes, or, equivalently, 25 seconds of game time), and the accumulated rewards were discounted as $R(s') = R(s) + \gamma^d r(s)$ where s a unique parent of s', and d is the depth of s in the search tree. The discount factor was set to $\gamma = 0.95$. To reduce the variance of the result each game was executed 30 times, with seeds $0, \ldots, 29$.

Table 2 shows that UNIFORM, SEQHALVING, and ϵ-GREEDY rather consistently outperform UCT. For example, on the 53 games, UNIFORM achieved higher average scores in 44 games, 1 game ended up with a draw, and UCT achieved higher average scores in 8 games. The situation is almost the same for SEQHALVING and ϵ-GREEDY, with the only difference that both of them draw in one more game, rather than winning it. It's also important to note that this family of MCTS algorithms also outperforms P-IW(1) in a majority of the games, as Table 2 shows us (albeit with a bit less significant margin).

Composite methods

As can be seen from Table 1, the relative performance of the novelty-based and MCTS approaches varies highly depending on the game (take the scores for FROSTBITE and JAMESBOND as an example). These approaches seem to be somewhat complementary indeed, so it seems natural to try and combine them in some way, such that we could reap the benefits of both. In this section we cover a few ways in which a composition between them could be achieved. It is worth mentioning that there was some earlier attempts to use some novelty properties in MCTS by Soemers *et al.* (2016), however, the methods introduced in their work depend heavily on the GVG-AI environment, and cannot be generalized to ALE straight forward. Also, it should be noted that we attribute the effect on the relative scores of P-IW(1) and MCST methods on different games to some innate property of the games themselves (rather than pure chance).

Naïve approach

One can assume that the aforementioned property might be discovered relatively early in the game (the correctness of this assumption will be discussed later). If this is true, one also can invest a relatively negligible amount of simulations in order to discover which algorithm is better suited for this particular game. We propose to do it in the following way:

1. in the first decision point of the game, use some budget of simulations B to run P-IW(1);

[2]The rewards in this game are time based, and it is challenging to extract these rewards in the black-box ALE setting.

Game	P-IW(1)	UCT	UNIFORM	SEQHALVING	$\epsilon = 0.67$
ALIEN	13264	6382	**17240**	14696	16881
AMIDAR	**2041**	47	910	1005	1031
ASSAULT	1552	1625	1831	1805	**1867**
ASTERIX	**347800**	303333	255875	264283	271267
ASTEROIDS	5548	4122	9987	10541	**12490**
ATLANTIS	197450	178753	197547	197133	**200223**
BANK HEIST	592	518	1793	2117	**2498**
BATTLE ZONE	3667	41500	130400	**232200**	103967
BEAM RIDER	4868	5024	14208	15340	**15745**
BERZERK	485	555	**739**	723	684
BOWLING	64	22	**83**	**83**	80
BREAKOUT	856	805	849	849	**864**
CARNIVAL	5605	4787	**6759**	6446	6323
CENTIPEDE	**186964**	106260	136776	138767	144143
CHOPPER COMMAND	2140	18243	36480	42500	**52240**
CRAZY CLIMBER	**140833**	135563	97134	84525	91753
DEMON ATTACK	**38898**	24128	30512	30700	31641
DOUBLE DUNK	-16	**24**	**24**	**24**	**24**
ELEVATOR ACTION	23077	14427	26137	25790	**26390**
ENDURO	0	279	405	**428**	426
FISHING DERBY	18	34	36	**39**	26
FREEWAY	**33**	0	8	8	7
FROSTBITE	**7667**	272	293	295	293
GOPHER	28618	8215	29285	29544	**30099**
GRAVITAR	1177	2888	5767	**5862**	5235
HERO	5702	10100	13933	**14478**	13946
ICE HOCKEY	15	39	**56**	**56**	**56**
JAMESBOND	40	385	13813	**15068**	14927
JOURNEY ESCAPE	2507	1320	**86120**	84420	77830
KANGAROO	**4337**	2048	2027	2047	2107
KRULL	**11443**	8742	5241	5744	6023
KUNG FU MASTER	**79717**	50347	63000	63690	63667
MONTEZUMA REVENGE	0	0	50	**293**	193
MS PACMAN	23584	17502	30893	30958	**33548**
NAME THIS GAME	**16713**	14927	13992	13779	14062
PONG	**21**	**21**	**21**	**21**	**21**
POOYAN	18943	14655	20358	**21182**	20802
PRIVATE EYE	920	100	1040	9	**1076**
QBERT	21727	17598	38274	**43233**	36553
RIVERRAID	**11702**	6316	9607	9915	9994
ROAD RUNNER	**62650**	39043	59650	60343	55283
ROBOTANK	6	45	**58**	**58**	**58**
SEAQUEST	590	543	499	490	**609**
SPACE INVADERS	2448	2482	2603	**2665**	2417
STAR GUNNER	1373	**1467**	1233	1217	1200
TENNIS	**24**	3	**24**	**24**	22
TIME PILOT	54137	52640	53060	53052	**63386**
TUTANKHAM	135	229	**271**	259	257
UP N DOWN	73325	63272	99037	**100593**	99639
VENTURE	**33**	0	10	17	0
VIDEO PINBALL	**592102**	323700	340109	328939	334709
WIZARD OF WOR	115347	98327	127667	129713	**133525**
ZAXXON	21553	24540	45047	**52217**	46123
Best in	16	3	9	15	16

Table 1: Performance results over the 53 Atari 2600 games. The algorithms P-IW(1), UCT, UNIFORM, SEQHALVING, $\epsilon = 0.67$, (ϵ-GREEDY) are evaluated over 30 episodes for each game. The look ahead of every algorithm is limited to a budget of 50000 simulated frames. The maximum episode duration is 18000 frames. Numbers in bold show best performer in terms of average score. The **Best in** row shows on how many games an algorithm scored the maximum.

	P-IW(1)	UCT	UNIFORM	SEQHALVING	$\epsilon = 0.67$
$p - IW(1)$	0	34	20	21	19
UCT	17	0	8	8	8
$Uniform$	33	44	0	17	21
$SeqHalving$	31	43	34	0	25
$\epsilon = 0.67$	34	43	31	27	0

Table 2: presents on how many instances the algorithm in row X outperformed the algorithm in column Y. For example, UNIFORM was strictly better than UCT on 44 out of 53 instances.

2. delete the search tree;
3. use budget B again to run an UNIFORM algorithm described in Section ;
4. delete the search tree;
5. for the rest of the decision points in the game, use the algorithm that yielded the highest cumulative reward.

It is important to emphasize that this decision is being made only once per game, thus B can be relatively large compared to the budget allocated per decision point.

ALTERNATION

One of the techniques for combining different approaches for the same problem is alternating between them. In classical planning this approach was introduced by Röger and Helmert (2010).

The version of alternation presented here builds a search tree as dictated by the UNIFORM algorithm at every even decision point, and as dictated by P-IW(1) at any odd decision point (thus alternating between the two). This type of combination, however, presents an ambiguity when we reuse already generated frames. In order to get rid of this ambiguity, we apply these rules:

1. odd step: run the P-IW(1) algorithm, do not prune the nodes that are already present in the search tree (including those added in UNIFORM step), and ignore them in the novelty calculation, so more nodes can escape pruning;
2. after recommending the "best" action chosen by the algorithm, add the nodes rooted in the node created by the recommended action to the search tree as they are;
3. even step: run the UNIFORM algorithm;
4. once again, pass the subtree of the winning action to the next iteration;
5. in each step, recommend the action, that leads to the trajectory with the highest commutative reward.

In contrast to the MCTS family of algorithms described in Section , this algorithm often picks actions that lead to the rooted subtrees of different sizes and depths. This may result both in positive and negative outcomes, as will be demonstrated in the next Section.

Empirical Evaluation of the Composite Methods

We evaluate both composition methods mentioned in the previous Section on the setting described in Section [3]. We compare both amalgamation methods to the base line presented by the algorithms P-IW(1) and UNIFORM (which was chosen as a representative of the MCTS family as the most basic method). Two numbers in the *Naïve* approach represent the score that the algorithm that is based upon running the algorithm chosen by the classification, and the accuracy of the algorithm (the percentage where the classification was correct). If the score on both P-IW(1) and UNIFORM pre-test is equal (this mostly happens when none of the algorithms could find any reward from the initial state), we choose randomly between the two.

The budget B allocated to each of the two iterations is 150000 (30000 nodes), this is tree times more than the budget allocated to each decision point.

[3] Some games were evaluated over 15 episodes because of the lack of time

Table 3 shows that the *Naïve* approach is a weak binary classifier (its accuracy is 62.8%), which means that the underlying assumption mentioned earlier is not entirely correct. The results of the classifier may be boosted however, by applying the tests to the game in question with different random seeds , and pick the algorithm that achieved most points in the majority of the runs (accuracy of this approach is 75.8%).

The experiments show that the ALTERNATION technique typically results in a score closer to the maximum between P-IW(1) and UNIFORM. In some games, however, it fails to achieve even a minimum score between them. These games can be divided into two groups. First – adversarial games, e.g.,PONG, TENNIS, or FISHING DERBY. The common denominator of this games is the presence of the negative rewards (in adversarial games negative rewards appear if the opponent is getting some positive rewards). In this setting, the disadvantage comes from the frame reuse of the algorithm. The subtrees produced by the P-IW(1) part and the UNIFORM part are mixed in one search tree. Abundance of the negative rewards makes longer trajectories more likely to yield negative score overall, which makes the algorithm choose an action starting the shorter trajectories, thus gravitating towards myopic decisions (and those aren't likely to be optimal). On the other hand, there are games like CRAZY CLIMBER and VIDEO PINBALL, in which the rewards are very sparse. That once again leads to a bias, but now towards longer rollouts of the UNIFORM algorithm. This leads us to conclusion, that in order to get a better version of this algorithm, there is a need in more balancing between the P-IW(1) and UNIFORM parts of the algorithm.

To summarize the experimental result on the ALTERNATION presented in Table 3. In most games the scores of the algorithm lie in-between the scores of P-IW(1) and UNIFORM with a slight bias towards the UNIFORM algorithm. However, on 15 games the algorithm scores more than the maximum between the two components, and on 44 games it scores more than minimum. An important finding is that in 6 games the algorithm scores more than 150% of the maximum of its components.

Summary

Black-box planning still presents a challenging task, since unlike in the classical planning domains, a black-box planner cannot rely on the off-the-shelf techniques that employ any kind of reasoning over propositional encoding of actions and goals. Previous works show that BRFS-like algorithms with pruning, such as IW(1) and P-IW(1) are setting the state-of-the-art performance, with the key to success being structural, similarity-based approximation of duplicate pruning (Lipovetzky *et al.* 2015). However, we have demonstrated that methods based on the Monte-Carlo simulations can be competitive in the majority of the Atari games, given some reasonable adaptations to the setting. The empirical results show that all 4 algorithms proposed in this work significantly outperform UCT and are competitive with the state-of-the-art P-IW(1).

Empirical evaluation also shows us that the blind search and MCTS approaches excel in different games, and a com-

Game	P-IW(1)	UNIFORM	*Naïve*		ALTERNATION
ALIEN	13264	17240	13264	0.37	**31939**
AMIDAR	2041	932	2041	0.90	**2216**
ASSAULT	1551	**1831**	1774	0.73	1825
ASTERIX	**349041**	255875	349041	1.00	273283
ASTEROIDS	5548	**9986**	9053	0.77	8222
ATLANTIS	197450	**197546**	197163	0.40	196870
BANK HEIST	591	1792	1247	0.77	**2782**
BATTLE ZONE	3666	130400	130400	1.00	**259333**
BEAM RIDER	4868	14208	14208	0.97	**14922**
BERZERK	485	739	707	0.80	**1044**
BOWLING	63	**83**	81	0.93	78
BREAKOUT	**855**	832	832	0.79	849
CARNIVAL	5605	**6759**	6598	0.80	6682
CENTIPEDE	**186964**	136775	155046	0.37	149865
CHOPPER COMMAND	2140	**36480**	6283	0.03	26986
CRAZY CLIMBER	**140388**	90418	91614	0.04	76948
DEMON ATTACK	**38897**	30511	35572	0.60	33252
DOUBLE DUNK	-16	**24**	24	1.00	**24**
ELEVATOR ACTION	22503	**26136**	25918	0.97	25556
ENDURO	0	**405**	254	0.63	358
FISHING DERBY	18	**36**	29	0.63	-1
FREEWAY	**32**	7	32	1.00	31
FROSTBITE	**7667**	293	3572	0.50	994
GOPHER	28618	**29284**	29284	0.67	27812
GRAVITAR	1176	**5766**	5766	1.00	5448
HERO	5702	**13932**	9765	0.47	13858
ICE HOCKEY	14	**55**	39	0.60	**55**
JAMESBOND	40	13813	12149	0.87	**14883**
JOURNEY ESCAPE	2506	**86120**	5161	0.07	69900
KANGAROO	**4336**	2026	2026	0.00	3300
KRULL	**11443**	5240	8200	0.50	10193
KUNG FU MASTER	**79716**	63000	65735	0.17	64446
MONTEZUMA REVENGE	0	50	25	1.00	**536**
MS PACMAN	23583	30893	26245	0.47	**33229**
NAME THIS GAME	**16713**	13991	14391	0.23	14086
PONG	**21**	**21**	21	0.77	2
POOYAN	18926	**20358**	20235	0.93	20283
PRIVATE EYE	919	**1040**	1040	0.50	731
QBERT	21726	**38274**	29782	0.73	36189
RIVERRAID	**11702**	9607	11702	0.77	10045
ROAD RUNNER	62655	58982	62655	0.69	**127715**
ROBOTANK	6	**58**	54	0.93	**58**
SEAQUEST	**589**	498	621	0.63	350
SPACE INVADERS	2447	2602	2535	0.60	**2620**
STAR GUNNER	**1373**	1233	1320	0.33	1300
TENNIS	**24**	**24**	24	0.57	19
TIME PILOT	**54136**	53060	50800	0.43	53606
TUTANKHAM	134	**271**	157	0.23	252
UP N DOWN	73325	99037	76000	0.13	**99730**
VENTURE	33	10	21	0.73	**96**
VIDEO PINBALL	**592101**	340108	439210	0.33	323869
WIZARD OF WOR	115346	127666	124746	0.47	**132540**
ZAXXON	21553	**45046**	39385	0.77	38800
Best in	18	22		0.63	17

Table 3: Performance results over the 53 Atari 2600 games. The algorithms P-IW(1), UNIFORM, *Naïve*, and ALTERNATION are evaluated over 30 episodes for each game. The look ahead of every algorithm is limited to a budget of 50000 simulated frames. The maximum episode duration is 18000 frames. Numbers in bold show best performer in terms of average score. Average scores were rounded to a nearest integer

position of the two may yield a more consistent result on a previously unseen task. Further experiments with different types of composition tend to support this claim. We explore two types of composition: pre-sampling based selection among the complementary methods, and an alternation between them. Both composite methods perform with mixed success. It is possible that a smarter classification of games can give an insight what algorithm should be used on the problem at hand.

References

M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The Arcade Learning Environment: An evaluation platform for general agents. *JAIR*, 47:253–279, 2013.

Sébastien Bubeck, Rémi Munos, and Gilles Stoltz. Pure exploration in multi-armed bandits problems. In *Algorithmic Learning Theory, 20th International Conference, ALT 2009, Porto, Portugal, October 3-5, 2009. Proceedings*, pages 23–37, 2009.

Guillaume M JB Chaslot, Mark HM Winands, H JAAP VAN DEN Herik, Jos WHM Uiterwijk, and Bruno Bouzy. Progressive strategies for monte-carlo tree search. *New Mathematics and Natural Computation*, 4(03):343–357, 2008.

Pierre-Arnaud Coquelin and Rémi Munos. Bandit algorithms for tree search. In *UAI 2007, Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence, Vancouver, BC, Canada, July 19-22, 2007*, pages 67–74, 2007.

Zohar Feldman and Carmel Domshlak. Monte-carlo planning: Theoretically fast convergence meets practical efficiency. In *Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence, UAI 2013, Bellevue, WA, USA, August 11-15, 2013*, 2013.

Zohar Feldman and Carmel Domshlak. On mabs and separation of concerns in monte-carlo planning for mdps. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2014, Portsmouth, New Hampshire, USA, June 21-26, 2014*, 2014.

Hector Geffner and Blai Bonet. *A Concise Introduction to Models and Methods for Automated Planning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2013.

Yuu Jinnai and Alex S. Fukunaga. Learning to prune dominated action sequences in online black-box planning. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA.*, pages 839–845, 2017.

Zohar Shay Karnin, Tomer Koren, and Oren Somekh. Almost optimal exploration in multi-armed bandits. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, pages 1238–1246, 2013.

Thomas Keller and Malte Helmert. Trial-based heuristic tree search for finite horizon mdps. In *ICAPS*, 2013.

L. Kocsis and C. Szepesvári. Bandit based Monte-Carlo planning. In *ECML*, pages 282–293, 2006.

N. Lipovetzky, M. Ramírez, and H. Geffner. Classical planning with simulators: Results on the Atari video games. pages 1610–1616, 2015.

Gabriele Röger and Malte Helmert. The more, the merrier: Combining heuristic estimators for satisficing planning. *Alternation*, 10(100s):1000s, 2010.

Stuart J. Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach (3. internat. ed.)*. Pearson Education, 2010.

Tim Schulte and Thomas Keller. Balancing exploration and exploitation in classical planning. In *Proceedings of the Seventh Annual Symposium on Combinatorial Search, SOCS 2014, Prague, Czech Republic, 15-17 August 2014.*, 2014.

Alexander Shleyfman, Alexander Tuisov, and Carmel Domshlak. Blind search for atari-like online planning revisited. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 3251–3257, 2016.

Dennis JNJ Soemers, Chiara F Sironi, Torsten Schuster, and Mark HM Winands. Enhancements for real-time monte-carlo tree search in general video game playing. In *Computational Intelligence and Games (CIG), 2016 IEEE Conference on*, pages 1–8. IEEE, 2016.

David Tolpin and Solomon Eyal Shimony. MCTS based on simple regret. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada.*, 2012.

On Computational Complexity of Automorphism Groups in Classical Planning

Alexander Shleyfman
Technion, Haifa, Israel
alesh@campus.technion.ac.il

Abstract

Symmetry-based pruning is a family of state-of-the art methods that are used to reduce the search effort. Applying these methods requires first to establish an automorphism group that is used later on during the main search procedure. Although this group can be applied in various contexts, one of the prominent ways is to use it for pruning symmetric states. Despite the increasing popularity of these techniques, nothing have been said about the computational complexity of the automorphism group of a general planning task. Herein, we show a reduction that proves that the aforementioned problem of computing the symmetry group of planning task is GI-hard. Furthermore, we discuss the presentation of these symmetry groups and list some of their drawbacks.

Introduction

Symmetry breaking is a method for search reduction, that was well-explored across several areas in Computer Science, including, but not limited to classical planning (e. g. (Starke 1991; Emerson and Sistla 1996; Fox and Long 1999; Rintanen 2003; Pochter *et al.* 2011; Domshlak *et al.* 2012)). Symmetry pruning divides the states in the search space into orbit-based equivalence classes, which in turn, allows to explore only one representative state per such class. Application of this technique to the forward search partially prunes the exponential growth of the search space in the presence of objects with symmetric behavior. This method, however, requires to compute first some subgroup of automorphisms of the state transition graph.

The first notion of symmetries for classical planing was proposed by Pochter *et al.* (2011), and then refined by Domshlak *et al.* (2012). The definitions presented in these works, practical however they are, were based on the notion of colored graphs, and thus are quite cumbersome to reason about. Later on, Shleyfman *et al.* (2015) came up with the notion of structural symmetries that captures previously proposed concepts, and which can be derived from the syntax of a planning task in a simple declarative manner.

With this in mind, it is quite surprising, that not much has been said about the complexity of computing an automorphism group for a given planning task. In this work, we present two reductions that not only provide us with a pessimistic result that computing automorphism group for a specific planning task is as hard as for any undirected graph, but also show the non trivial connection between the automorphism groups of a planning task and its causal graph.

Background

To define a *planning task* we use the finite-domain representation formalism (FDR) presented by (Bäckström and Nebel 1995; Helmert 2006). Each task is given by a tuple $\Pi = \langle \mathcal{V}, \mathcal{A}, I, G \rangle$, where $\mathcal{V}$ is a set of multivalued *variables*, each associated with a finite domain $\mathcal{D}(v)$. The sets of variable/value pairs are written as $\langle var, val \rangle$, and sometimes referred as *facts*. A *state* s is a full variable assignments which maps each variable $v \in \mathcal{V}$ to some value in its domain, i.e. $s(v) \in \mathcal{D}(v)$. For $V \subseteq \mathcal{V}$, $s[V]$ denotes the partial assignment (also referred as a *partial state*) of s over V. *Initial state* I is a state. The *goal* G is a partial assignment. Let p be a partial assignment. We denote by $vars(p) \subseteq \mathcal{V}$ the subset of variables on which p is defined. For two partial assignments p and q, we say that p *satisfies* q, if $vars(q) \subseteq vars(p)$, and $p[v] = q[v]$ for all $v \in vars(q)$, this is denoted by $p \models q$. $\mathcal{A}$ is a finite set of *actions*, each represented by a triple $\langle \mathsf{pre}(a), \mathsf{eff}(a), \mathsf{cost}(a) \rangle$ of *precondition*, *effect*, and *cost*, where $\mathsf{pre}(a)$ and $\mathsf{eff}(a)$ are partial assignments to $\mathcal{V}$, and $\mathsf{cost}(a) \in \mathbb{R}^{0+}$. In this work we assume all actions are of unit-cost, unless otherwise stated. An action a is *applicable* in a state s if $s \models \mathsf{pre}(a)$. Applying a in s changes the value of all $v \in vars(\mathsf{eff}(a))$ to $\mathsf{eff}(a)[v]$, and leaves s unchanged elsewhere. The outcome state is denoted by $s[\![a]\!]$.

S denotes the set of all states of Π. We say that action sequence π is a *plan*, if it begins in I ends in in s_G s.t. $s_G \models G$, and each action in π is iteratively applicable, i.e. for each $a_i \in \pi$ holds that $s_{i-1} \models \mathsf{pre}(a_i)$ and $s_{i-1}[\![a_i]\!] = s_i$. The cost of a plan defined as $\mathsf{cost}(\pi) = \sum_{a_i \in \pi} \mathsf{cost}(a_i)$. An *optimal* plan, is a plan of a minimal cost. In a unit-cost domain, an optimal plan is a plan of the shortest length. The *state space* of Π is denoted $\mathcal{T}_\Pi$.

A *directed graph* is pair $\langle N, E \rangle$ where N is the set of *vertices*, and $E \subseteq N^2$ is a set of *edges*. An *undirected graph* is a pair $\langle N, E \rangle$ where N, once again, is the set of *vertices*, and $E \subseteq \{e \subseteq N \mid |e| = 2\}$ is the set of edges.

Let $\mathcal{G} = \langle N, E \rangle$ be a (un-)directed graph, and let σ be a permutation over the vertices N, we say that σ is a graph *automorphism* (or just an automorphism, if this is clear from the context) when $(n, n') \in E$ iff $(\sigma(n), \sigma(n')) \in E$. The

automorphisms[1] of a graph $\mathcal{G}$ are closed under composition, and for every automorphism there exists an inverse permutation which is also an automorphism. Thus, automorphisms of a graph form a group. We call this group an *automorphism group*, and denote it by $Aut(\mathcal{G})$. The identity element e in this group will be denoted by $id_{\mathcal{G}}$.

Causal Graph

One way of capturing the structural complexity of a planning task are causal graphs. The idea mention in numerous papers, e.g. (Knoblock 1994; Bacchus and Yang 1994; Domshlak and Brafman 2002), but here we will follow the definition given by Helmert (2010).

The *causal graph* of a planning task $\Pi = \langle \mathcal{V}, \mathcal{A}, I, G \rangle$ is a directed graph $CG(\Pi) = \langle \mathcal{V}, \mathcal{E} \rangle$, where $(u, v) \in \mathcal{E}$ if $u \neq v$ and there exists $a \in \mathcal{A}$, s.t. $u \in vars(\mathsf{pre}(a)) \cup vars(\mathsf{eff}(a))$ and $v \in vars(\mathsf{eff}(a))$.

In a nutshell, the causal graph contains an edge from a source variable to a target variable if changing the value of the target variable may depend on the value of the source variable, even if it's only a co-depending effects.

Structural Symmetries

The second ingredient we need was recently introduced by Shleyfman *et al.* (2015). This subsection defines the notion of *structural symmetries*, which capture previously proposed concepts of symmetries in classical planning. In short, structural symmetries are relabellings of the FDR representation of a given planning task Π. Variables are mapped to variables, values to values (preserving the $\langle var, val \rangle$ structure), and actions are mapped to actions. In this work, we follow the definition of structural symmetries for FDR planning tasks as defined by Wehrle *et al.* (2015). For a planning task $\Pi = \langle \mathcal{V}, \mathcal{A}, I, G \rangle$, let P be the set of Π's facts, and let $P_{\mathcal{V}} := \{\{\langle v, d \rangle \mid d \in \mathcal{D}(v)\} \mid v \in \mathcal{V}\}$ be the set of sets of facts attributed to each variable in $\mathcal{V}$. We say that a permutation $\sigma : P \cup A \to P \cup \mathcal{A}$ is a *structural symmetry* if the following holds:

1. $\sigma(P_{\mathcal{V}}) = P_{\mathcal{V}}$,
2. $\sigma(\mathcal{A}) = \mathcal{A}$, and, for all $a \in \mathcal{A}$, $\sigma(\mathsf{pre}(a)) = \mathsf{pre}(\sigma(a))$, $\sigma(\mathsf{eff}(a)) = \mathsf{eff}(\sigma(a))$, and $\mathsf{cost}(\sigma(a)) = \mathsf{cost}(a)$.
3. $\sigma(G) = G$.

Note, that that the definition $\sigma(X) := \{\sigma(x) \mid x \in X\}$, where X is a set, can be applied recursively. For example, let s be a partial state, since s can be represented a set of facts, applying σ to s will result in a partial state s', s.t. for all facts $\langle v, d \rangle \in s$ it holds that $\sigma(\langle v, d \rangle) = \langle v', d' \rangle \in s'$ and $s'[v'] = d'$.

A set of structural symmetries Σ for a planning task Π induce a subgroup Γ of the automorphism group $Aut(\mathcal{T}_\Pi)$, which in turn defines an equivalence relation over the states S of Π. Namely, we say that s is *symmetric* to s' iff there exists an automorphism $\sigma \in \Gamma$ such that $\sigma(s) = s'$.

[1]The definition of a *graph automorphism* for an undirected graph is almost the same with the only exception of edges, that are sets of a size 2 and not directed pairs.

Symmetries from Problem Description Graphs

The last creature we want to introduce in this section is the problem description graph (PDG), that was introduced by Pochter *et al.* (2011), and later on reformulated for different purposes by Domshlak *et al.* (2012), and Shleyfman *et al.* (2015). In this work we will use the definition of PDG for the FDR planning tasks. It is important to point out, that this structure has no direct use in the work below. However, since we want to illustrate some of our claims by graphic examples, PDG becomes quite helpful, since in contrast to structural symmetries PDG is a graph, and hence can be presented in a picture.

Definition 1. *Let Π be a FDR planning task. The **problem description graph** (PDG) of Π is the colored directed graph $\langle N, E \rangle$ with nodes*

$$N = N_{\mathcal{V}} \cup \bigcup_{v \in \mathcal{V}} N_{\mathcal{D}(v)} \cup N_{\mathcal{A}}$$

where $N_{\mathcal{V}} = \{n_v \mid v \in \mathcal{V}\}$, $N_{\mathcal{D}(v)} = \{n_{\langle v,d \rangle} \mid d \in \mathcal{D}v\}$, and $N_{\mathcal{A}} = \{n_a \mid a \in \mathcal{A}\}$; node colors

$$col(n) = \begin{cases} 0 & \text{if } n \in N_{\mathcal{V}} \\ 1 & \text{if } n \in \bigcup_{v \in \mathcal{V}} N_{\mathcal{D}(v)} \text{ and } \langle v, d \rangle \in G \\ 2 & \text{if } n \in \bigcup_{v \in \mathcal{V}} N_{\mathcal{D}(v)} \text{ and } \langle v, d \rangle \notin G \\ 3 + \mathsf{cost}(a) & \text{if } n_a \in N_{\mathcal{A}} \end{cases}$$

and edges

$$E = \bigcup_{v \in \mathcal{V}} E^v \cup \bigcup_{a \in \mathcal{A}} E_a^{\mathsf{pre}} \cup E_a^{\mathsf{eff}}$$

where $E^v = \{(n_v, n_{\langle v,d \rangle}) \mid d \in \mathcal{D}(v)\}$, $E_a^{\mathsf{pre}} = \{(n_a, n_{\langle v,d \rangle}) \mid \langle v, d \rangle \in \mathsf{pre}(a)\}$, and $E_a^{\mathsf{eff}} = \{(n_{\langle v,d \rangle}, n_a) \mid \langle v, d \rangle \in \mathsf{eff}(a)\}$.

In their work, Pochter *et al.* (2011) observed, that *PDG symmetry* is a symmetry of $\mathcal{T}_\Pi$ that is induced by a graph automorphism of the PDG of Π. Shleyfman *et al.* (2015), in turn, showed that every structural symmetry of Π corresponds to a PDG symmetry of Π in the sense that they induce the same transition graph symmetry. In what follows, we will denote by $Aut(\Pi)$ the automorphism group of PDG of a task Π. The illustrative examples of planning tasks will also be presented via PDGs.

Complexity

In this section we aim to prove that for each undirected graph one may construct a planning task with a similar automorphism group. Surprisingly, not much have been said about complexity of computation of such a group. We show a simple reduction that will prove that this computation is at least GI-hard.

The graph isomorphism problem (GI) is a well-known problem, that gave its name to a whole complexity class. This problem is a decision problem of determining whether two finite graphs are isomorphic. Other well know problem is the graph automorphism problem, that is a problem of testing whether a graph has a nontrivial automorphism. And this

is at least as hard as solving the decision problem of either automorphism group of a given graph is trivial or not. The graph automorphism problem is polynomial-time many-one reducible to the graph isomorphism problem (Mathon 1979) (the converse reduction is unknown). Thus, given the reduction, we can say that computing the automorphism group of a given planning task is at least GI-hard. The latest result by Babai (2015) claims (most probably rightfully) that GI can be solved in quasipolynomial time, i.e. in $\exp((\log n)^{O(1)})$. The best previous bound stood on $O(\exp(\sqrt{n \log n}))$ (Babai and Luks 1983).

Reduction to bounded variable domains

While discussing relations between the automorphism groups of different structures, we first should introduce the notation of mapping and comparing these groups. In this section we will rely mostly on the basic definitions of the group theory taken from the book "Topics in algebra" (Herstein 1975). Let us start with some useful mappings:

Definition 2. *Let G and G' be groups:*

1. *and let $f : G \mapsto G'$ be a mapping that satisfies $f(ab) = f(a)f(b)$ for all $a, b \in G$, then f is a **homomorphism** of G to G'.*
2. *If f is a bijection, it is called an **isomorphism**, this is denoted by $G \cong G'$, or simply $G = G'$.*
3. *If f is a injection, then there exist a subgroup $H \leq G$, s. t. $H \cong G'$. In this case we will write simply $G \leq G'$.*

As we mentioned before, Pochter *et al.* (2011) introduced a method for deduction of the automorphism group for an FDR representation of a planning task using PDG. While this representation is easy to visualize and understand, it is a bit inconvenient as an algebraic model of representation. On the other hand, structural symmetries while having a much simpler definition, lack the graphical appeal. In the proof which follows, we want to establish a connection between the vertexes of a given undirected graph and the variables of the planning task constructed by our reduction. Thus, as the middle ground, we chose to conduct our proof using causal graphs, since they are both simple in representation and, once again, have an allure of being graphs. Unfortunately, as the next statement shows, there is no straightforward subgroup relation between the automorphism group of the planning task and the automorphism group of its causal graph.

Observation 1. *There exist a planning task Π, s. t. $Aut(\Pi) \not\leq Aut(CG(\Pi))$ and $Aut(CG(\Pi)) \not\leq Aut(\Pi)$.*

Proof. Let Π be a planning task with variables $\mathcal{V}$ and actions $\mathcal{A}$. Consider a set of variables $\mathcal{V} = \{v, u\}$, where $\mathcal{D}(v) = \{v_1, v_2, v_3, v_4\}$ and $\mathcal{D}(u) = \{u_1, u_2\}$. Let $\mathcal{A}$ be a set of actions with single precondition and single effect. To ease the writing and reading of this example, we will use the following notation $a_{x_i \to y_j} := \langle\{\langle x, x_i\rangle\}, \{\langle y, y_j\rangle\}\rangle$. Thus, for example the action $a_{v_1 \to v_2} = \langle\{\langle v, v_1\rangle\}, \{\langle v, v_2\rangle\}\rangle$. Now, let us define a set of actions of Π, $\mathcal{A} = \{a_{v_1 \to v_2}, a_{v_2 \to v_3}, a_{v_3 \to v_1}, a_{v_1 \to v_4}, a_{v_2 \to v_4}, a_{v_3 \to v_4}, a_{v_4 \to u_1}, a_{u_1 \to u_2}, a_{u_2 \to v_4}\}$. Since we don't want our planning task to be redundant we will set $G = \{\langle v, v_4\rangle\}$. It is easy to check that $Aut(\Pi)$ is generated by the cycle $(\langle v, v_1\rangle, \langle v, v_2\rangle, \langle v, v_3\rangle)$, i. e. for some $\sigma \in Aut(\Pi)$ holds that $\sigma(\langle v, v_1\rangle) = \langle v, v_2\rangle$, $\sigma(\langle v, v_2\rangle) = \langle v, v_3\rangle$, and $\sigma^3 = id_\Pi$ to complete the cycle, and that σ is fixed on all other facts. Thus, $Aut(\Pi) \cong \mathbb{Z}_3$, and cyclic group of order 3. The causal graph and PDG of Π are depicted in Figure 1.

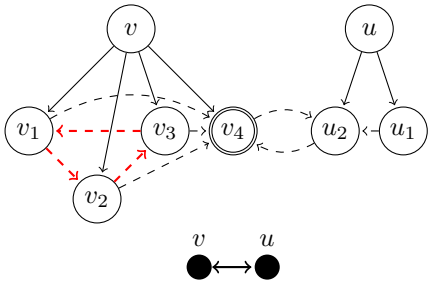


Figure 1: Illustration for Observation 1: The graph with the white nodes represents the PDG of the task described in the Observation in question. Since preconditions and effects of each action are single-valued, we annotated them via dashed arrows. The goal fact is denoted by double circle. Here it is easy to see, that the automorphism group of the PGD is generated by the cycle (v_1, v_2, v_3) (red, dashed arrows). The filled dots represent the causal graph of the same task.

On the other hand, the causal graph of task Π is $\langle N = \{v, u\}, \mathcal{E} = \{(v, u), (u, v)\}\rangle$, and has the automorphism group that is isomorphic to $\mathbb{Z}_2$. Hence, since both $\mathbb{Z}_3, \mathbb{Z}_2$ have no non-trivial subgroups the claim holds. □

Since we still want to embed $Aut(\Pi)$ into $Aut(CG(\Pi))$, we will need the following Definitions and Theorem (once again taken from the "Topics in algebra" (Herstein 1975)).

Definition 3. *Let H be a subgroup of G, and let x be an element in G.*

1. *The set of elements $Hx = \{hx \mid h \in H\}$ is called a **right coset** of H. A **left coset** defined similarly.*
2. *If for every $x \in G$ holds that $Hx = xH$, H is called a **normal subgroup**.*
3. *Let H be a normal subgroup of a G. The set ${}^{G}/_{H} := \{xH \mid x \in G\}$ of all left cosets forms a **quotient group** of G modulo H.*

Below we give a short reminder on the definition of group homomorphism.

Definition 4. *Let H and G be two groups. We say that the map $\phi : G \to H$ is a **group homomorphism** (later on referred as homomorphism), if for all $g_1, g_2 \in G$ it holds that $\phi(g_1)\phi(g_2) = \phi(g_1 g_2)$.*

*The **kernel** of the map ϕ, denoted by $ker(\phi)$, is the set $\phi^{-1}(id_H)$.*

*We say that ϕ is an **isomorphism**, if in addition to the mentioned above it is also a bijection.*

Note, that it is easy to prove that $ker(\phi) \leq G$ and $\phi(G) \leq H$ are both subgroups. To establish additional relationships between homomorphisms, quotients, and subgroups we will need the following theorem by Noether (1927).

Theorem 1 (First isomorphism theorem)**.** *Let G and H be groups, and let $\phi : G \to H$ be a homomorphism. Then:*

1. *The kernel of ϕ is a normal subgroup of G,*
2. *The image of ϕ is a subgroup of H, and*
3. *The image of ϕ is isomorphic to the quotient group ${}^{G}/_{ker(\phi)}$.*

In particular, if ϕ is surjective then H is isomorphic to ${}^{G}/_{ker(\phi)}$.

Intuitively, the next Lemma shows that if we strip from $Aut(\Pi)$ all the automorphism that do not affect the tasks variables, the resulted subgroup can be embedded into the automorphism group of $CG(\Pi)$.

Lemma 1. *Let $\phi : Aut(\Pi) \mapsto Aut(CG(\Pi))$ be a map s.t. for each $\sigma \in Aut(\Pi) : \phi(\sigma) = \sigma_{\mathcal{V}}$, where $\sigma_{\mathcal{V}}$ is σ restricted to $\mathcal{V}$.*

Then, ϕ is a homomorphism, and ${}^{Aut(\Pi)}/_{ker(\phi)} \leq Aut(CG(\Pi))$.

Proof. Once again, let Π be a planning task with variables $\mathcal{V}$ and actions $\mathcal{A}$. First, let us prove that $\sigma_{\mathcal{V}}$ is an automorphism. Let $(u, v) \in \mathcal{E}$ be an edge in $CG(\Pi)$. Hence, exists $a \in \mathcal{A}$, s.t. $u \in vars(\mathsf{pre}(a)) \cup vars(\mathsf{eff}(a))$ and $v \in vars(\mathsf{eff}(a))$. And therefore, for each $\sigma \in Aut(\Pi)$ it holds that $\sigma(u) \in vars(\mathsf{pre}(\sigma(a))) \cup vars(\mathsf{eff}(\sigma(a)))$ and $\sigma(v) \in vars(\mathsf{eff}(\sigma(a)))$. From which follows that $(\sigma_{\mathcal{V}}(u), \sigma_{\mathcal{V}}(v)) \in \mathcal{E}$. The converse is true, since each σ^{-1} is also an automorphism.

Second, ϕ is a homeomorphism, since for each $\sigma, \sigma' \in Aut(\Pi)$, it holds that $\phi(\sigma)\phi(\sigma') = \sigma_{\mathcal{V}}\sigma'_{\mathcal{V}} = (\sigma\sigma')_{\mathcal{V}} = \phi(\sigma\sigma')$, given that ϕ is a restriction to variables.

Now, $ker(\phi) = \{\sigma \in Aut(\Pi) \mid \sigma = id_{\mathcal{V}}\}$, and by the first isomorphism theorem it holds that ${}^{Aut(\Pi)}/_{ker(\phi)} = \phi(Aut(\Pi)) \leq Aut(CG(\Pi))$. □

Following the intuition of Lemma 1, in the Theorem below we construct a planning task that has no "inner" automorphism. The automorphism group of such task should be isomorphic to the automorphism groups of its causal graph. The theorem is the main result of this section.

Theorem 2. *Let $\mathcal{G}$ be a directed graph without loops, then there exists a planning task Π, s.t. $\mathcal{G} = CG(\Pi)$, $Aut(\mathcal{G}) = Aut(\Pi)$.*

Proof. In this proof, given a directed graph $\mathcal{G} = \langle N, E\rangle$, we should construct a planning task Π that satisfies the conditions of the Theorem. First, it's clear that vertex $x \in N$ should correspond a variable $v \in \mathcal{V}$. Now, since we would like to use Lemma 1, the kernel of the homomorphism ϕ should be trivial. Thus, we set $\mathcal{D}v = \{T, F\}$, and add an action $a_{v:F\to v:T} := \langle\{\langle v, F\rangle\}, \{\langle v, T\rangle\}\rangle$, s. t. for each $\sigma \in Aut(\Pi)$ holds $\sigma(\langle v, F\rangle) \neq \langle v, T\rangle$. For each $(x, y) \in E$, let v and u be the corresponding variables in $\mathcal{V}$. To ensure that $\mathcal{G} = CG(\Pi)$, we add a unique action $a_{u:F\to v:F} := \langle\{\langle u, F\rangle\}, \{\langle v, F\rangle\}\rangle$, which, in turn, assures that if $\sigma(a_{u:F\to v:F}) \neq a_{u:F\to v:F}$, then either $\sigma(v) \neq v$ or $\sigma(u) \neq u$. In addition, we need to specify an initial state, and a goal description. Let those two be full assignments $I := \{\langle v, F\rangle \mid v \in \mathcal{V}\}$ and $G := \{\langle v, T\rangle \mid v \in \mathcal{V}\}$. Since, by construction of Π, σ never maps T to F, this leaves the automorphism group $Aut(\Pi)$ unchanged. To summarize, the constructed planning task $\Pi = \langle\mathcal{V}, \mathcal{A}, I, G\rangle$ looks as follows:

1. $\mathcal{V} = \{v \mid v \in N\}$, with $\mathcal{D}v = \{T, F\}$ for each v,
2. $\mathcal{A} = \{a_{v:F\to v:T} \mid v \in V\} \cup \{a_{v:F\to u:F} \mid (v, u) \in E\}$,
3. $I = \{\langle v, F\rangle \mid v \in \mathcal{V}\}$, and
4. $G = \{\langle v, T\rangle \mid v \in \mathcal{V}\}$.

Since the algebraic mapping can be hard to imagine, the PDG structure of edge (u, v) is depicted in Figure 2.

Now let ϕ be a homomorphism as defined is Lemma 1. By construction of Π, ϕ is surjective and $ker(\phi) = id_{\mathcal{G}}$, thus $Aut(\mathcal{G}) = Aut(\Pi)$. □

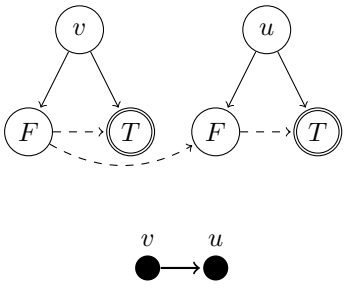


Figure 2: Illustration of a mapping of a single edge in a graph for Theorem 2: Once again, the graph with the white nodes represents the PDG of and edge (v, u) (depicted by filled nodes). Here it easy to see that there no "inner" symmetries, and the planning variable $(v, \mathcal{D}(v))$ can be mapped into a planning variable $(u, \mathcal{D}(u))$ exactly in one way.

Now what is left to show, is that there is an automorphism group preserving reduction from a undirected graphs to directed graphs.

Proposition 1. *Let $\mathcal{G}$ be a undirected graph, then there exists directed graph $\mathbb{G}$, s.t. $Aut(\mathcal{G}) = Aut(\mathbb{G})$.*

The proof of this statement is not new, but we will use it later on to show that even special cases of planning tasks are difficult to solve, in the sense of finding the automorphism group.

Proof. Let $\mathcal{G} = \langle N, E\rangle$ be an undirected graph. Let us define $\mathbb{G} = \langle\mathcal{V}, \mathcal{E}\rangle$ as follows:

1. $\mathcal{V} := N \cup E$, and
2. $\mathcal{E} := \{(e, x), (e, y) \mid e = \{x, y\} \in E\}$.

Note, that for the vertices in $\mathcal{V}$ for $x \in N$ and $e \in E$, $deg_{out}(x) = deg_{in}(e) = 0$. The graphic example of this construction can be seen if Figure 3. Hence, for each $\sigma \in Aut(\mathbb{G})$ holds that $\sigma(N) = N$ and $\sigma(E) = E$, where N and E are both sets of vertices in $\mathbb{G}$. Moreover, for each edge $e = \{x, y\}$ in E correspond to edges $(e, x), (e, y)$ in $\mathcal{G}$. Thus, for $\sigma \in Aut(\mathbb{G})$, $e = \{x, y\} \in \mathcal{E}$ iff $\sigma(e) = \{\sigma(x), \sigma(y)\} \in \mathcal{E}$ which corresponds to $(e, x), (e, y) \in \mathcal{E}$

iff $(\sigma(e), \sigma(x)), (\sigma(e), \sigma(y)) \in \mathcal{E}$. Using this, we will define $\phi : Aut(\mathcal{G}) \to Aut(\mathbb{G})$:

$$\phi(\sigma) = \begin{cases} \sigma(x) & \text{if } x \in N, \\ \sigma(e) = \{\sigma(x), \sigma(y)\} & \text{if } e = \{x, y\} \in E. \end{cases}$$

Now, to prove that ϕ is an isomorphism we need to prove that ϕ is surjection, and that $ker(\phi) = \{id_{\mathcal{G}}\}$. First, for each $\tau \in Aut(\mathbb{G})$, $\phi^{-1}(\tau) = \tau|_N \in Aut(\mathcal{G})$. Second, $\phi^{-1}(id_{\mathbb{G}}) = id_{\mathbb{G}}|_N = id_{\mathcal{G}}$. Thus, by the first isomorphism theorem it holds that ${}^{Aut(\mathcal{G})}/_{\{id_{\mathcal{G}}\}} = Aut(\mathcal{G}) = Aut(\mathbb{G})$. □

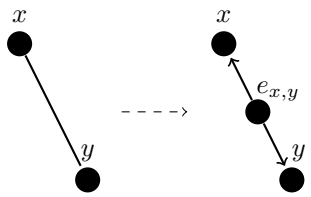


Figure 3: Illustration of a mapping of a single edge in a graph for Proposition 1: edge $e_{x,y} = \{x, y\} \in E$ is mapped to a vertex $e_{x,y} \in \mathcal{V}$ and two edges $(e, x), (e, y) \in \mathcal{E}$.

The next Corollary is the immediate consequence of Proposition 1 and Theorem 2.

Corollary 1. *Given a planing task Π, computing $Aut(\Pi)$ is as equivalent to computing $Aut(\mathcal{G})$ for some undirected graph $\mathcal{G}$.*

Proof of Proposition 1 also show thats even planning tasks that have a bipartite one-way directed causal graphs (fork decomposition by Katz and Domshlak (2008)) may have an arbitrary finite automorphism group, which follows from the next theorem proven by Frucht (1949)

Theorem 3 (Frucht's theorem)**.** *Every finite group is the automorphism group of a finite undirected graph.*

Reduction to singe variable domain

As for now, we have seen that given an undirected graph $\mathcal{G}$ we can construct a planning task Π with the same automorphism group, and where each vertex in the graph $\mathcal{G}$ corresponds with a bounded variable in a task Π. In this section we show, that if we remove the bounded domains condition, only one variable for such a task will suffice.

Zemlyachenko *et al.* (1985) showed that finding an isomorphism of a connected graph is a GI-complete problem. Therefore, to prevent the task from being reducible via standard preprocessing we will take the graph $\mathcal{G}$ to be a connected undirected graph.

Proposition 2. *Let $\mathcal{G}$ be a connected undirected graph, then there exists planning task $\Pi = \langle \mathcal{V}, \mathcal{A}, I, G \rangle$, s.t. $Aut(\mathcal{G}) = Aut(\Pi)$ and $|\mathcal{V}| = 1$.*

Proof. Let $\langle N, E \rangle$ be the vertices and edges of $\mathcal{G}$, correspondingly, and let $\mathcal{V} = v$ be single variable in the task Π. We will define the domain on v to be $\mathcal{D}(v) := \{v_x \mid x \in N\} \cup \{v_g\}$, where v_g is the goal value of v ($G := \{\langle v, v_g \rangle\}$). Now, since the structural symmetries ignore the initial state, all is left to do is to define the actions of this task. Since we have only one variable we will use the following notation $a_{v_x \to v_y} := \langle \{\langle v, v_x \rangle\}, \{\langle v, v_y \rangle\} \rangle$. The actions $\mathcal{A}$ of our task will be divided into two sets:

1. $\mathcal{A}_E := \{a_{v_x \to v_y}, a_{v_y \to v_x} \mid e = \{x, y\} \in E\}$, and
2. $\mathcal{A}_g := \{a_{v_x \to v_g} \mid x \in N\}$.

Now, let us look at the map $\psi : N \to \{\langle v, d \rangle \mid d \in \mathcal{D}(v)\}$, by construction ψ is injective. Therefore the map $\phi : Aut(\mathcal{G}) \to Aut(\Pi)$:

$$\phi(\sigma) = \begin{cases} \sigma(\psi(x)) & \text{if } x \in N, \\ \langle v, v_g \rangle & \text{otherwise} \end{cases}$$

is also injective, since it's easy to see that ψ preserves the relation on the edges, $\psi : E \to \mathcal{A}_E$ set-wise. The injection follows from the fact that $\langle v, v_g \rangle$ is a unique goal fact that can be mapped by σ only upon itself, and $\psi : N \to \mathcal{A}_g$ is a bijection. Thus we get the desired $Aut(\mathcal{G}) = Aut(\Pi)$. □

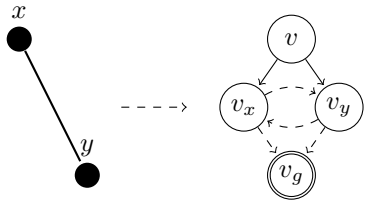


Figure 4: Illustration of a mapping of a single edge in a graph for Proposition 2: vertices $x, y \in N$,and an edge $e_{x,y} = \{x, y\} \in E$ are mapped to value vertices v_x, v_y and two dashed action edges $(v_x, v_y), (v_y, v_x)$, correspondingly. in addition, to preserve the PDG structure, each PDG graph will have a single variable vertex v, a single goal value v_g, and an edge (v_x, v_g) for each $x \in N$.

Group Presentation

In this section we will discuss the presentation of symmetry groups, and show that for some planning domains this task may be quite difficult.

Most of the tools for computing automorphism groups, such as *Bliss* (Junttila and Kaski 2007), *nauty* (McKay and Piperno 2014), and *saucy* (Darga *et al.* 2008), report the set of generators required to produce the $Aut(G)$ automorphism group of a given graph G. Thus, in some works, the authors (Sievers *et al.* 2015; 2017) chose to report this numbers for each group, or even for each planning domain in the experimental benchmarks. In this subsection we will show the faults in this approach. To do so, we will need some standard definitions:

Definition 5. *Let G be a group. We say that G has a **presentation** $\langle S \mid R \rangle$, where S is a set of **generators** so that every element of the group can be written as a product of powers of some of these generators, and R is a set of **relations** among those generators.*

*Let $F(S)$ be a **free group** on S (all finite words of S with the relation $suu^{-1}t = st$). The set of relations R is the subset of $F(S)$.*

The group G is said to have the above presentation if it is isomorphic to the quotient of $F(S)$ by the minimal normal subgroup that contains the set R.

*We say that presentation $\langle S \mid R\rangle$ of group G is **irreducible** if for no $S' \subsetneq S$ holds that G isomorphic to $\langle S' \mid R|_{S'}\rangle$.*

From the First isomorphism theorem follows that every finite group has a presentation. And as corollary of this statement, easily obtained, that every finite group is finitely generated, since S can be taken to be G itself. To get a better grip on this definition we will present an couple of examples, that will be use further in this section.

Example 1. *The **cyclic group** is a group generated by a single element. The group $\mathcal{C}_k$ can be presented as $\langle S \mid R\rangle$ where:*

- $S := \{\sigma\}$;
- $R := \{\sigma^k\}$.

It is easy to see that for a given $k \in \mathbb{N}$, it holds that $|\mathcal{C}_k| = k$, and only one generator. Example 1 provides us the fact that the number of generators does not ensure the upper bound on the size of the group. To calculate the lower bound we will prove the following lemma[2]:

Lemma 2. *Let G be a group with presentation $\langle S \mid R\rangle$, and let be $S = \{g_1, \dots, g_n\}$ set of n irreducible generators. Then, $|G| \geq 2^n$.*

Proof. Let G^m be a subgroup of G that has a set of generators $\{g_1, \dots g_m\}$, for $1 \leq m < n$. Since the set S is irreducible with respect to G, every subset of S is also irreducible with respect to the subgroup of G it generates, thus $g_{m+1} \notin G^m$. Therefore, G^{m+1} has at least two cosets $e\,G^m = G^m$ and $g_{m+1}G^m$. By definition of cosets it holds that $G^m \cap g_{m+1}G^m = \emptyset$. Which leads to $|G^{m+1}| \geq 2|G^m|$. Thus, by induction on m we have that $G^n \geq 2^n$. □

The equality for the Lemma above is achieved on the group $\mathcal{C}_2^n \cong \mathbb{Z}_2^n = \prod_{i=1}^n \mathbb{Z}_2$. Giving us that the amount of elements (*order*) of a finite group, is at least exponential in the size of group generators.

To show that group structure is not defined by the number of generators we will need at least one another group, that is not cyclic. For that we will define the symmetric group. Note, that in literature symmetry group is often used as a synonym to the automorphism group of some mathematical object, symmetric group, on the other hand, is a group of all permutations of some n identical objects.

Example 2. *The **symmetric group** $\mathcal{S}_n$ on a finite set of n symbols is the group whose elements are all the permutations on n distinct symbols. The group $\mathcal{S}_n$ can be written as $\langle S \mid R\rangle$ where:*

- $S := \{\sigma_i | i \in [n-1]\}$;
- $R := \{\sigma_i^2 | i \in [n-1]\} \cup \{\sigma_i\sigma_j\sigma_i^{-1}\sigma_j^{-1} | i \neq j \pm 1\} \cup \{(\sigma_i\sigma_j)^3 | i, j \in [n-1]\}$

[2]This result is well known in group theory, but unfortunately we haven't found citing source.

It's important to point out (and easy to check) that $\mathcal{S}_n$ has $n!$ elements. Using the cyclic notation, each element in the presentation can be written as $\sigma_i = (i, i+1)$, which means that σ_i maps the element i to element $i+1$, element $i+1$ is mapped to i, and other elements are mapped to itself. This presentation is irreducible (Alperin and Bell 1995), meaning that non of the permutations can be excluded from the set S, where $|S| = n - 1$.

As we showed a bit earlier in this section the group $\mathcal{C}_2^n$ also has n generators, each of order two[3]. However, it is obvious that $\mathcal{C}_2^n \ncong \mathcal{S}_{n+1}$, since $2^n \neq (n+1)!$, for $n > 1$.

By this example, reporting the number of generator per domain, or even for a specific group (even with order of these generators) is not very informative, since this numbers reports us almost nothing on the size and structure of the group. Note that the group $\langle a, b \,|\, a^2, b^2\rangle$ is of an infinite size (as an intuition, consider the following elements $a, ab, aba, abab, \dots$).

Another way to write the cyclic group $\mathcal{S}_n$, may be given with only two cyclic generators $(1, 2)$ and $(2, \dots, n)$, which is also irreducible (Alperin and Bell 1995). Note that in the first cyclic presentation of $\mathcal{S}_n$ each generator has an order of 2, and in the second the first cycle is of order 2, and the second one is of order $n - 1$. As we can see, each group may have more than one presentation. And calculating the minimal number of these generators per group is known to be at most $O(\log^2 n)$ space (Arvind and Torán 2006).

Conclusion

An automorphism group of a planning task can be seen as permutation on objects involved in this task, and thus it constitutes a subgroup of some symmetric group. In this sense, our results coincide with the famous Cayley's theorem (Herstein 1975).

Theorem 4 (Cayley's theorem)**.** *Every group G is isomorphic to a subgroup of the symmetric group acting on G.*

The obvious corollary of this theorem is that, every finite group is isomorphic to a subgroup of the symmetric group.

One may have an intuition, that since planning tasks are constructed from objects, the symmetry groups of such tasks can be subjects to a special treatment. This is apparently false in the general case. Thus, we want to conclude this unoptimistic paper by a quote from the book "Groups and representations" by Alperin and Bell (1995): "in general the fact that finite groups are embedded in symmetric groups has not influenced the methods used to study finite groups". Unfortunately, by reduction we showed in the previous Section, this statement also holds for the automorphism groups of the tasks in classical planning.

References

J.L. Alperin and R.B. Bell. *Groups and representations*. Graduate texts in mathematics. Springer, 1995.

Vikraman Arvind and Jacobo Torán. The complexity of quasigroup isomorphism and the minimum generating set problem. In *ISAAC*, 2006.

[3]Order of an element g is the minimal number m s.t. $g^m = e$.

László Babai and Eugene M. Luks. Canonical labeling of graphs. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 25-27 April, 1983, Boston, Massachusetts, USA*, pages 171–183, 1983.

László Babai. Graph isomorphism in quasipolynomial time. *CoRR*, abs/1512.03547, 2015.

Fahiem Bacchus and Qiang Yang. Downward refinement and the efficiency of hierarchical problem solving. *AIJ*, 71(1):43–100, 1994.

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

Paul T. Darga, Karem A. Sakallah, and Igor L. Markov. Faster symmetry discovery using sparsity of symmetries. In *Proceedings of the 45th Annual Design Automation Conference*, DAC '08, pages 149–154, New York, NY, USA, 2008. ACM.

Carmel Domshlak and Ronen I. Brafman. Structure and complexity in planning with unary operators. In Malik Ghallab, Joachim Hertzberg, and Paolo Traverso, editors, *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems, April 23-27, 2002, Toulouse, France*, pages 34–43. AAAI Press, 2002.

Carmel Domshlak, Michael Katz, and Alexander Shleyfman. Enhanced symmetry breaking in cost-optimal planning as forward search. In Blai Bonet, Lee McCluskey, José Reinaldo Silva, and Brian Williams, editors, *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*. AAAI Press, 2012.

E. Allen Emerson and A. Prasad Sistla. Symmetry and model-checking. 9(1/2):105–131, 1996.

Maria Fox and Derek Long. The detection and exploitation of symmetry in planning problems. In Thomas Dean, editor, *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI 1999)*, pages 956–961. Morgan Kaufmann, 1999.

Robert Frucht. Graphs of degree three with a given abstract group. 1(??):365–378, 1949.

Malte Helmert. The Fast Downward planning system. *JAIR*, 26:191–246, 2006.

Malte Helmert. Landmark heuristics for the pancake problem. In Ariel Felner and Nathan Sturtevant, editors, *Proceedings of the Third Annual Symposium on Combinatorial Search (SoCS 2010)*, pages 109–110. AAAI Press, 2010.

I.N. Herstein. *Topics in algebra*. Xerox College Pub., 1975.

Tommi Junttila and Petteri Kaski. Engineering an efficient canonical labeling tool for large and sparse graphs. In David Applegate, Gerth Stølting Brodal, Daniel Panario, and Robert Sedgewick, editors, *Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments and the Fourth Workshop on Analytic Algorithms and Combinatorics*, pages 135–149. SIAM, 2007.

Michael Katz and Carmel Domshlak. Structural patterns heuristics via fork decomposition. In Jussi Rintanen, Bernhard Nebel, J. Christopher Beck, and Eric Hansen, editors, *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS 2008)*, pages 182–189. AAAI Press, 2008.

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

R. Mathon. A note on the graph isomorphism counting problem. *Information Processing Letters*, 8:131–132, 1979.

Brendan D. McKay and Adolfo Piperno. Practical graph isomorphism, {II}. *Journal of Symbolic Computation*, 60(0):94 – 112, 2014.

E. Noether. Abstrakter aufbau der idealtheorie in algebraischen zahl- und funktionenkörpern. *Mathematische Annalen*, 96:26–61, 1927.

Nir Pochter, Aviv Zohar, and Jeffrey S. Rosenschein. Exploiting problem symmetries in state-based planners. In Wolfram Burgard and Dan Roth, editors, *Proceedings of the 25th National Conference of the American Association for Artificial Intelligence (AAAI'11)*, San Francisco, CA, USA, July 2011. AAAI Press.

J. Rintanen. Symmetry reduction for SAT representations of transition systems. In Enrico Giunchiglia, Nicola Muscettola, and Dana Nau, editors, *Proceedings of the 13th International Conference on Automated Planning and Scheduling (ICAPS'03)*, pages 32–41, Trento, Italy, 2003.

Alexander Shleyfman, Michael Katz, Malte Helmert, Silvan Sievers, and Martin Wehrle. Heuristics and symmetries in classical planning. In Blai Bonet and Sven Koenig, editors, *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI'15)*, pages 3371–3377. AAAI Press, January 2015.

Silvan Sievers, Martin Wehrle, Malte Helmert, and Michael Katz. An empirical case study on symmetry handling in cost-optimal planning as heuristic search. In Steffen Hölldobler, Markus Krötzsch, Rafael Peñaloza, and Sebastian Rudolph, editors, *KI 2015: Advances in Artificial Intelligence - 38th Annual German Conference on AI, Dresden, Germany, September 21-25, 2015, Proceedings*, volume 9324 of *Lecture Notes in Computer Science*, pages 166–180. Springer, 2015.

Silvan Sievers, Gabriele Röger, Martin Wehrle, and Michael Katz. Structural symmetries of the lifted representation of classical planning tasks. In *HSDIP 2017*, 2017.

Peter Starke. Reachability analysis of petri nets using symmetries. *Journal of Mathematical Modelling and Simulation in Systems Analysis*, 8(4/5):293–304, 1991.

Martin Wehrle, Malte Helmert, Alexander Shleyfman, and Michael Katz. Integrating partial order reduction and symmetry elimination for cost-optimal classical planning. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 1712–1718, 2015.

V. N. Zemlyachenko, N. M. Korneenko, and R. I. Tyshkevich. Graph isomorphism problem. *Journal of Soviet Mathematics*, 29(4):1426–1481, May 1985.

Representing General Numeric Uncertainty in Non-Deterministic Forwards Planning

Liana Marinescu and **Andrew Coles**
Department of Informatics,
King's College London, UK
firstname.lastname@kcl.ac.uk

Abstract

Many interesting applications of planning exhibit numeric uncertainty. Prior work in forwards planning approximates uncertain values as Gaussian distributions, but this is not always accurate. We explore a novel way to represent numeric uncertainty more generally. Our approach allows us to sample non-deterministic action effects from any probability distribution without sacrificing computational time. We integrate our approach into an existing policy-building setting, and use it to improve how well the states expanded by search reflect reality. This is part of a work in progress, and will provide new insights into the amount of detail about uncertainty necessary to obtain robust plans.

1 Introduction

1.1 Context

Planning under uncertainty is a compelling research area due to its role in broadening the range of problems that automated planners can tackle. Common situations where uncertainty arises include noisy sensors, unpredictable environments, and limited domain knowledge. For example, after each stretch of driving on rough terrain, a car may or may not have suffered a flat tyre - this is a case of propositional uncertainty. Or, after navigating through more or less favourable currents, a submarine may have used a non-deterministic amount of fuel - this is a case of numeric uncertainty. In this paper we focus on the latter, and in particular on the representation of non-deterministic numeric effects as probability distributions.

There is no question that plan robustness benefits from taking uncertainty into account. While it is possible to ignore uncertainty and assume all non-deterministic numeric effects take the median value every time, this simplification can have serious consequences for plan success. For example, a mission-critical action might cost 10 units of fuel; the planner might use the median value of all effects so far to calculate that the fuel remaining is exactly 10; it will then consider it safe to take the mission-critical action. In reality though, there might be 8.5 (or 9, or 10.7) fuel remaining, so there is a risk that the mission-critical action will fail. Planning under uncertainty aims to address this problem.

1.2 Prior Work

There is a rich body of prior work which tackles uncertainty from several angles.

Providing an excellent starting point for our contributions, work by (Beaudry, Kabanza, and Michaud 2010) uses a Bayesian network to model resources and time based on continuous random variables. They introduce the idea of querying the Bayesian network to check the likelihood of the variables remaining in a valid state. We base our planning kernel on their approach, as described further in this paper.

(Coles 2012) adapted the work of (Beaudry, Kabanza, and Michaud 2010) to assuming for heuristic purposes that variables take their median value. They proposed a method to first generate plans that are conservative about resource usage, and then to create branches that can exploit situations where resource usage is less than pessimistically expected. This approach to branching inspired part of our work as well.

Building on top of (Coles 2012) is the paper by (Marinescu and Coles 2016a), which employs the median in the calculation of the heuristic, and additionally introduces the concept of an offset – a safety margin by which preconditions must be met some given percentage of the time. They compute this margin based on Gaussian uncertainty in the problem model, and enable the planner to consider actions that reduce uncertainty.

For propositional uncertainty (where actions have many discrete outcomes), work by (Muise, McIlraith, and Beck 2012) and (Muise, Belle, and McIlraith 2014) on the planner PRP builds a policy by making repeated calls to a deterministic planning kernel. This kernel finds weak plans, which assume the non-deterministic action outcome can be chosen. They incrementally build a policy to cover the outcomes that were not chosen, and recurse. Their approach scales remarkably well due to the use of regression to keep only the relevant parts of a state, leading to a compact policy representation.

Building on the work of (Muise, McIlraith, and Beck 2012) on propositional uncertainty, (Marinescu and Coles 2016b) extend the policy-building process to numeric uncertainty. They achieve this by defining the process of regression through non-deterministic numeric effects. This offers them the additional benefit of generalising numeric deadends in order to prune them more efficiently. The limitation

here is that the approach only works if the effects are represented as Gaussian probability distributions. We aim to address this limitation with our current work.

There are many other compelling approaches to planning with numeric uncertainty. For example, work by (Beck and Wilson 2007) solves the job shop scheduling problem in the case where durations are drawn from Gaussian distributions. (Babaki, Guns, and Raedt 2017) integrate a probabilistic engine with constraint programming in order to acommodate uncertain demand or processing times in decision-making problems. Optimal planning in stochastic domains with resource constraints is addressed with a novel algorithm by (Meuleau et al. 2009). POMDPs are used to model the problem of maximising performance while bounding risk with a safety threshold by (Santana, Thiebaux, and Williams 2016).

1.3 Our work

We aim to tackle one of the main drawbacks of prior work on policy-bulding for non-deterministic numeric planning – the fact that it is limited in scope when it comes to the types of probability distributions it can accommodate. General distributions are acknowledged but they are not focused on, with approaches so far concentrating only on Gaussian distributions.

In (Marinescu and Coles 2016b) the outcome of a non-deterministic action can be chosen when starting to build the policy. For example, if an action has three modes (lucky, normal, and unlucky) the weak plan would always choose the lucky outcome and leave the other outcomes to be filled in later.

In our approach however, the user should not be telling the planner control knowledge, so we don't require the outcome modes to be specified in the domain file. In fact we expend no computational effort on discovering what the modes of a certain action might be – especially since the most suitable modes to branch on may differ at different points in the plan for the same action. We instead rely on branching as necessary in order to meet the certainty requirements, as we explain further below.

One novel element in the context of general probability distributions is enabling the planner to be proactive about improving on its initial solution. It can start by finding a policy which meets a given certainty threshold, and continue by incrementally refining the solution in order to nudge up the certainty.

An obvious question to pre-empt would be the following – won't we sacrifice computational time in order to accommodate any probability distributions in the policy-building process? We explore this concern later in the paper through the use of parallel computation in order to speed up our extended regression algorithm.

In the following, we first define the non-deterministic planning formalism we use, and the policy-building process on top of which we build our work. We then present our contribution and discuss our implementation and our extension to prior work.

As this is a work in progress, we can only provide our best estimates regarding experimental performance, and we put forth our ideas on both ways our approach could perform – faster or slower than the Gaussian-only approach.

2 Background

2.1 Planning with Numeric Uncertainty

The formalism we use in this work is based on that of (Beaudry, Kabanza, and Michaud 2010), adapted so that actions can have multiple outcomes, to support the policy-building mechanics we will detail in Section 2.2. A planning problem is thus a tuple $\langle F, \mathbf{v}, I, G, A, \theta \rangle$ where:

- F is a set of propositional facts.
- $\mathbf{v}$ is a vector of numeric variables.
- I is the initial state: a subset of F and values of variables in $\mathbf{v}$.
- Conditions are conjunctions of facts from F and Linear Normal Form constraints on $\mathbf{v}$, each written: $(\mathbf{w}.\mathbf{v} \geq c)$, where $c \in \Re$, and $\mathbf{w}$ is a vector of real values.
- G describes the goals: a set of conditions.
- A is a set of actions, with each $a \in A$ having:
  - $Pre(a)$: a (pre)condition on its execution;
  - $\mathit{Eff}(a)$: a list of outcomes. Each $o \in \mathit{Eff}(a)$ is a tuple $\langle \mathit{Eff}^+, \mathit{Eff}^-, \mathit{Eff}^{num} \rangle$ where:
    - Eff^+, Eff^-: a set of facts added (deleted) by that outcome;
    - Eff^{num}: a set of numeric variable updates triggered by that outcome, each of the form $\langle v \ op \ D(\mathbf{v}) \rangle$ where $op \in \{+=, =\}$ and D is a (possibly deterministic) probability distribution that governs the range of the numeric effect. For instance, $\langle battery \mathrel{+}= \mathcal{N}(-10, 2^2) \rangle$ means 'decrease $battery$ by an amount with mean 10 and standard deviation 2'.
- $\theta \in [0.5, 1)$ is a confidence level that $Pre(a)$ must meet to be considered true (this is necessary due to the uncertainty in effects).

Because there is uncertainty on numeric variables (due to the distributions D in Eff^{num}), it is not possible to be absolutely certain that numeric conditions are satisfied. Thus, (Beaudry, Kabanza, and Michaud 2010) uses a Bayesian Network (BN) to model this uncertainty, and check that numeric conditions are satisfied with the prescribed confidence level θ. When each action has only a single outcome, the task of planning is to find a sequence of steps $[a_0, .., a_n]$, giving a state trajectory $[I, S_0, .., S_n]$; with the BN ensuring that, with confidence θ, each action's preconditions are true and S_n satisfies the goals G.

Work by (Marinescu and Coles 2016a) looks at Gaussian vs non-Gaussian distributions in the context of heuristics. In particular, they intoduce a heuristic which is admissible for monotonically worsening uncertainty, based on the difference between the median and the θ'th percentile of a distribution. Using this difference (offset) they can evaluate whether numeric preconditions are true. In the case where an effect would have influence the uncertainty of a variable non-monotonically (e.g. assigns it a value rather than

Algorithm 1: Generating a Strong-Cyclic Plan in PRP (Muise *et al.* 2012)

Data: A planning task, with initial state I and goals G
Result: A policy P

1 $P \leftarrow \{\}$; $Open \leftarrow [I]$; $Seen \leftarrow \{\}$;
2 **while** *Open is not empty* **do**
3 $S \leftarrow Open$.pop();
4 **if** $(S \in Seen) \vee (S \vDash G)$ **then continue**;
5 $Seen \leftarrow Seen \cup S$;
6 **if** $\exists \langle ps, a \rangle \in P$ *such that* $S \vDash ps$ **then**
7 **for** $S' \in apply\_outcomes(S, a)$ **do**
8 $Open$.push(S');
9 **else**
10 $(weak\_plan, G') \leftarrow$ run planning kernel from S;
11 **if** *planning kernel could not solve problem* **then**
12 $ps\_dead \leftarrow generalise\_dead\_end(S)$;
13 generate forbidden state–action pairs from ps_{dead};
14 $P \leftarrow \{\}$; $Open \leftarrow [I]$; $Seen \leftarrow \{\}$;
15 **else**
16 $PS \leftarrow$ regress G' through $weak\_plan$ to generate partial-state–action pairs;
17 $P \leftarrow P \cup PS$;
18 **for** $S' \in apply\_outcomes(S, weak\_plan_0)$ **do**
19 $Open$.push(S');
20 **return** P

increases it by a value), then then in the heuristic the offset is reset back to zero.

2.2 Policy-Building for Uncertainty

As noted in the formalism above, actions can have multiple outcomes, and each outcome has a set of associated effects. A solution to problems containing such actions can be represented by using a *policy* – a set of rules that dictates what should be done in each state. For our policies, we assume states are fully observable, i.e. we know which action outcome occurred at any point.

In the presence of multiple outcomes, a *weak plan* corresponds to a single trajectory of actions that leads from the initial state to a goal state, assuming it is possible to choose which action outcome occurs at each point (i.e. to be optimistic). In the propositional case, weak plans can be found using a deterministic planner which is given as input the *all outcomes determinisation*. This means that each action with preconditions $Pre(a)$ and effects $\mathit{Eff}(a)$ is replaced by several actions, one for each $o \in \mathit{Eff}(a)$, whose preconditions are $Pre(a)$ and whose effects are just those corresponding to o.

(Muise, McIlraith, and Beck 2012) present an approach where, by repeatedly invoking a deterministic planner to find weak plans, it is possible to incrementally build a policy. The core of this approach is set out in Algorithm 1. Key to the success of their approach is exploiting *relevance* – by regressing the goal through a weak plan step-by-step, they determine which facts at each point are relevant to plan success.

Regression takes as input a partial state ps – here, a set of literals. It then applies an action 'backwards' to it, yielding a new partial state ps' that has to be satisfied prior to the action being applied. That is, applying the action in ps' returns us to ps.

The process begins from the goals, i.e. initially $ps = G$. Regressing ps through a step a of a weak plan, with preconditions $Pre(a)$ and a single outcome with add effects $\mathit{Eff}^+(a)$ yields a partial state ps' where:

$$ps' = (ps \setminus \mathit{Eff}^+(a)) \cup Pre(a)$$

Each of these pairs $\langle ps', a \rangle$ is added to the policy (line 17). If policy-building reaches a state S that matches some known partial-state–action pair (line 6), then all the outcomes of the corresponding action are applied. Otherwise, if there is no such match, the planning kernel is invoked from S. Ideally, this produces a weak plan, either to the goals G or to some other partial state which is already in the policy (G'), in which case the partial-state–action pairs from this weak plan are added to the policy.

Policy-building terminates when the open list is empty, hence $\exists \langle ps, a \rangle$ such that $S \vDash ps$ for all states S reachable from the initial state via the policy. Alternatively, if no strong cyclic plan exists, all actions that could be applied in the initial state are forbidden, and planning terminates with failure.

The numeric extension to this algorithm has been proposed by (Marinescu and Coles 2016b), where regression is performed through successive Gaussian effects by taking advantage of these distributions' analytical form.

3 Approach

3.1 Focus of the Paper

Our goal is to introduce a general representation of uncertain numeric effects in non-deterministic planning. We aim to allow effects to be sampled from any probability distribution, without incurring a high computational cost. Our representation allows the planner to find robust solutions which meet a given certainty threshold.

Below we present how the policy-building process described in Section 2.2 can be extended to incorporate a wider range of uncertain action effects. Specifically, our novel representation of uncertainty allows regression to be done through a sequence of actions with non-Gaussian effects.

The propositional elements of regression stay the same as in prior work by (Muise, McIlraith, and Beck 2012). We leverage, among other things, their approach to propositional uncertainty, for its notable speed when triaging possible matches to a given partial state.

The numeric elements of regression are the focus of our work. We show that it is computationally feasible to perform regression through non-Gaussian effects, and provide an efficient implementation to do so.

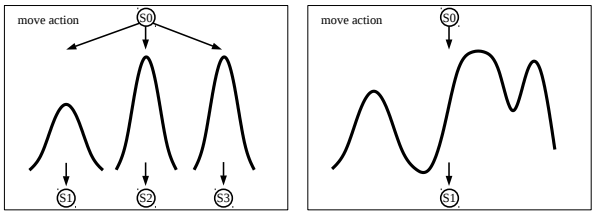


Figure 1: Multiple modes (left) vs single mode (right).

3.2 Multiple Modes vs Single Mode

One of the factors contributing to the success of prior work by (Marinescu and Coles 2016b) was the existence of explicit modes in non-deterministic action effects. For example, the `move` action had three different outcomes hard-coded in the PDDL domain file. One lucky mode (using less energy than nominally), one nominal mode, and one unlucky mode (using more energy than nominally). The existence of these modes allowed the planner to do two things. First, to perform an all-outcomes determinisation on the non-deterministic effects (cf. (Muise, McIlraith, and Beck 2012)), which resulted in a favourable weak plan (as it could choose all the lucky action outcomes). Second, to recursively branch off from the weak plan at those lucky points in order to solve for all the unfavourable outcomes as well.

However, there is a problem with these hard-coded modes – the user needs to specify them in the beginning. This is often impractical, as it forces the user to guess how an uncertain environment might react. It also implies giving the planner additional control knowledge, essentially offering it hints without being certain these hints are correct. We expect better performance when allowing the planner to decide by itself when branching an outcome into several modes is necessary.

The question then becomes, if we don't want to rely on the user to specify modes, how can we still leverage the prior work and its fast policy-building process? We need to infer the modes automatically, without extra information from the user. We also need to infer them efficiently – it would be inefficient for example to always branch 3 ways (or some other arbitrarily chosen number); it might not always be necessary to branch, as we explain below.

We propose to dynamically generate branches as needed. We do this by successively bisecting the probability distribution of the non-deterministic effect. Whether we generate a branch or not will depend on the success or failure of the weak plan from that branch outcome to the goal. This success or failure is dictated by θ as described in Section 2.

3.3 Representing the Single Mode

The core motivation of our work is to allow the representation of any probability distribution – not just a set of Gaussians – in non-deterministic action effects. We demonstrate that our ambition is computationally feasible and effective by using it to improve the policy-building process of prior work. Thus, we face the question – how do we represent a single, general probability distribution such that policy-building still works as efficiently as it did with Gaussian distributions?

We introduce the concept of a Bayesian Plan Network (BPN) as a representation of uncertainty at any point in the reachable search space. Its purpose is to check whether an action precondition holds given the uncertainty at the point of application.

The answer to the computation done by the BPN (a boolean – precondition holds or does not hold) is used to better inform state expansion about the uncertain environmental conditions. This is the case both in search and in RPG building. As part of this computation we use the certainty cutoff value θ explained in Section 2. We use the value $\theta = 0.9$ throughout the following to illustrate our approach. This is for demonstration purposes – the concepts we introduce work the same regardless of the value of θ.

3.4 Building a Bayesian Plan Network

The BPN can be described as a directed graph of nodes, where each node represents either a probability distribution d or a variable v. A BPN that corresponds to a given plan contains all the variables affected by that plan as they go through sequential changes (actions affecting them), together with their corresponding distributions (if an effect is not uncertain, its distribution is degenerate). The graph is weighted – coefficients from action effects are used to indicate how a variable depends on previous variables multiplied by constants.

The distribution nodes can be described as source nodes – they have no parent nods, as their value does not depend on other variables in the plan. They are akin to buckets of samples (whether described analytically, like the shape of a Gamma function, or empirically, like a collection of sensor measurements).

The variable nodes are essentially addition nodes. They have at least one parent node (which can be a distribution, or another variable). If a variable node is queried to obtain a sample of its value, it will in turn query its parents – this operation recursively all nodes in the PN once.

The steps for building a BPN are as follows:

1. Input a problem description and a weak plan found by the planning kernel.
2. For each variable set in the initial state, create one variable node and one distribution node for each one. Each distribution node is the parent of its corresponding variable node, and represents a degenerate distribution (only contains one sample, the value set by the initial state).
3. For each action in the weak plan, loop through its effects.
4. For each effect, create one new variable node for the affected variable. Then create parent links between the new node and all the variable nodes whose values are used by that effect (with their respective weights assigned to the parent link). Note that we keep track of the latest variable nodes at all times, to ensure the sequential changes to the variables are accurate.
5. If the effect above is non-deterministic, then create one

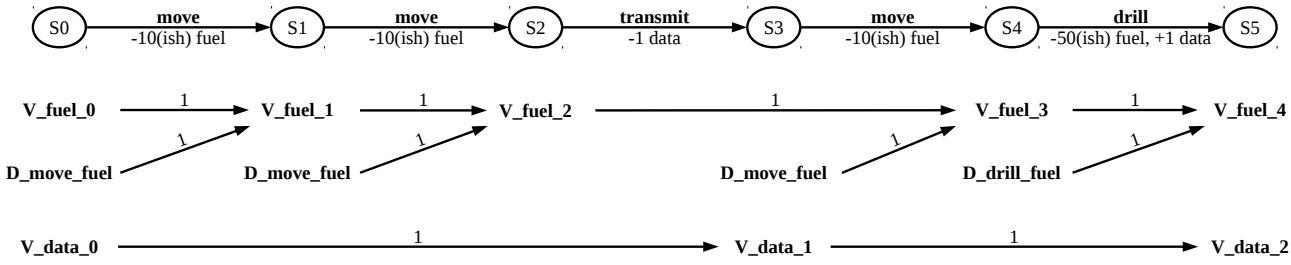


Figure 2: Weak plan and its corresponding BPN.

new distribution node containing the samples corresponding to that effect.

6. If the effect above uses a constant, then create one new distribution node containing a degenerate distribution (only contains one sample, the constant itself).
7. Each time a new node is created, sample its value a given N number of times and compute the median value. This median will be necessary when defining the regression operation in Section 3.5.
8. Each time an action with two modes (outcomes) is encountered, apply the operations above for each mode, then create one new variable node whose parents are the variable nodes of each mode, weighted equally at 0.5 each. This situation can arise when our algorithm cannot find a solution by using a single mode, and bisects the probability distribution at its median.

3.5 Integrating a Bayesian Plan Network with Prior Work

As the purpose of our work here is to allow the prior policy-building work to deal with general probability distributions, we present below how this can be achieved. Specifically, we introduce a novel way to perform regression.

The core idea of our contribution is to build a BPN and sample it to check whether an action's preconditions hold at any given stage.

The steps for building a policy by using the BPN to include general probability distributions are as follows:

1. Use the planning kernel to find a weak plan.
2. Use the weak plan to build a BPN as described in Section 3.4.
3. Use the BPN in the regression stepo to build a policy. This is where we use the BPN to check if preconditions hold, not with a Gaussian offset as in prior work, but with an offset obtained through repeated sampling of the BPN. We will expand on this step below.
4. Use the policy to decide which actions the planner should take.

Instead of the (partial state, action) pairs described in Section 2.2, the policy will now contain (partial state, list of actions) pairs. We obtain this list by storing the steps in the weak plan created when the partial state is expanded.

When checking if the policy knows what to do in a particular state, we first do the propositional triage. We look for all the partial states in the policy which match the facts in our particular state, and thus obtain a list of candidate matches.

To choose a candidate match, we need to recreate the functionality of regression, as we no longer have the option of computing regression analytically based on all uncertainty being Gaussian. We build a BPN from the actions in the plan-so-far up to our particular state, concatenated with the list of actions from our particular state to the goal (as mentioned above, the list of actions is found in the candidate pair). We then sample the BPN a certain amount of times – in our preliminary experiments, an amount of 1000 was suitable. By sampling the BPN we refer to sampling the goal state in the BPN, which will propagate backwards and eventually sample all nodes in the network. After each sampling, we loop through the plan that generated the BPN and check if at each step the preconditions hold, keeping track of each precondition's status with a counter. At the end of the 1000 sampling steps, we check if all the preconditions are satisfied with the required degree of certainty θ, e.g. if they are satisfied at least 90% of the time during those 1000 samples, based on our counters.

To improve the process of choosing a candidate match, we also order the list of candidates by heuristic value (Metric-RPG).

3.6 Representing a Plan Network Efficiently

The graphical representation of the BPN described in the previous section is useful to intuitively understand how the network functions. However, the sequential computations based on this representation slow down our approach and make it less competitive with prior work.

We thus propose a matrix representation in order to efficiently compute the answer to the central question in the section above – are all the preconditions in a given plan satisfied with certainty θ? Our method will allow each sample run to happen concurrently rather than sequentially, significantly reducing the time taken to compute the final answer.

The structure of the matrix stems from the topological order traversal of the BPN. Each row is a node, and each column is a sample run. For each distribution node we have a value of 1 in the column that corresponds to a sample from that distribution, and a value of 0 everywhere else. For each

variable node we have non-zero values in the columns that define that node's value in relation to the edges coming into it.

Then, to check if that BPN's corresponding plan is satisfied with certainty θ, we multiply our matrix representation with a matrix containing all the sampled values for all uncertain variables. We then use the result to count the number of sample runs in which all the preconditions were satisfied, as in Section 3.5.

3.7 Example

Consider a simple robot-waiter domain with two actions:

- The robot can move from the customer to the kitchen. This has deterministic propositional effects.
- The robot can move from kitchen to the customer, and pass the butter. The amount of butter passed is non-deterministic, according to some distribution, due to the accuracy of the robot's actuators.

From a modelling point of view, there is a single outcome mode for second of these actions: that some amount of butter is passed. In prior work (Marinescu and Coles 2016b) one would represent that as a single Gaussian-distributed outcome – there is no reason per se to use multiple Gaussian outcomes in the effect list of passing butter.

To ensure that enough butter is passed, with sufficient confidence, a strong plan could then be [move, pass-butter, move, pass-butter]. In the absence of multiple outcomes on passing butter (because there is no need to hand-prescribe multiple outcomes, from a user point-of-view), there would be no branching. As such, regardless of how much butter was passed, the expected solution length is four actions. A more efficient outcome would be to branch on the outcome of butter passing within the planner, rather than expecting this in the model. For instance, if with P(0.5) enough butter is passed, then a branch to execute the second round of moving and passing butter would reduce the expected solution length to $(0.5 \times 2) + (0.5 \times 4) = 3$ actions.

4 Evaluation

4.1 Evaluation Plan

As our work is still in progress, we will confine this section to presenting our evaluation plan.

First, we will compare our planner with the one used by (Marinescu and Coles 2016a). We will use the same numeric planning domains (such as `rovers`) for both planners, while taking into account that information about uncertainty is conveyed differently to these two planners. In prior work, the PDDL domain file contains the parameters of the Gaussian distribution associated with each non-deterministic effect. In our work, an additional data file contains a set of samples taken from a Gaussian distribution in a preprocessing phase.

This first comparison will establish whether our planner performs as well as prior work or better in problems where the uncertainty is genuinely a Gaussian (rather than poorly approximated as one). We expect these tests to confirm the computational feasibility of our work – even facing off against the fast analytical mathematics that are possible with Gaussians.

Second, we will take the prior work from (Marinescu and Coles 2016b) and compare it with our work in order to measure the impact of multi-mode versus single-mode action outcomes. As before, information about uncertainty is conveyed differently. The prior work once again hard-codes both the Gaussian parameters and the outcome modes into the PDDL domain file. For our work, we take in a data file containing samples from all the outcome modes (assuming that, for N modes, each mode has a $1/N$ likelihood to occur).

We make this second comparison in order to check how the removal of the hard-coded clues (the modes) impacts planner performance and solution certainty. One interesting metric to look at will be the expected probabilistic cost of the plan, as mentioned in Section 3.7.

Another useful indicator of performance will be the computational cost of planning. We are interested in comparing the process of finding a plan for the Gaussian case versus finding a plan for the general case. This will indicate whether the extra detail present in the general probability distribution impacts planner running time. We expect this not to be the case, due to the parallel computation approach we described in Section 3.6.

4.2 Potential Domains

We initially aim to test our planner on the same domains as prior work, namely:

1. `Rovers` from (Coles 2012), where the `move` action exhibits Gaussian uncertainty due to soil characteristics and potentially incomplete data on obstacles.
2. `AUV` from (Coles 2012) (modified to no longer be an over-subscription problem), where the `move` action outcome is drawn from a general probability distribution reflecting the influence of stochastic ocean currents.
3. `TPP` from (Gerevini et al. 2009), where the `purchase` action is drawn from a general probability distribution simulating the honesty of the merchants.

We are also actively seeking out domains from the planning applications community, in order to best illustrate the types of problems our approach is suitable for. Of particular interest are situations where probability distributions are skewed to either side of the median line, or exhibit modes (distinct areas on the graph) that are not easily distinguishable when modelling the problem. Intuitively, in these cases Gaussian approximations are not adequate.

4.3 Potential Outcomes

Extrapolating from the improvements obtained in (Marinescu and Coles 2016a) by adding approximate information about uncertainty into the heuristic, we expect that adding more accurate information about uncertainty into the heuristic will enhance the already-existing benefits. For example, we expect a more informed heuristic to discover dead ends faster (as it is not likely to lead search down risky paths). We also expect that, if placed side-by-side in a simulator,

our work would find more reliable solutions than (Marinescu and Coles 2016a) when measured by how frequently the solution obeyed the certainty threshold θ in the simulator.

Compared to the work in (Marinescu and Coles 2016b), on top of more accurate information about uncertainty, we expect our single-mode outcomes to lower the expected probabilistic cost of solution plans. We believe this is the case due to the planner only branching as-needed depending on the certainty threshold, rather than always branching on multiple pre-specified modes.

If the improvements outlined above do not occur, then our work provides evidence that the Gaussian approach taken in (Marinescu and Coles 2016b) is a good enough approximation of uncertainty – with the drawback of being reliant on explicit information on modes from the user.

Additionally, we expect the computational cost to be lower due to the efficient matrix implementation of the Plan Network outlined in Section 3.6.

If instead this cost turns out to be higher, it would mean that our implementation, in spite of parallel computing, cannot surpass the advantages offered by using analytic Gaussian mathematics.

5 Conclusions

5.1 Summary

In this paper, we introduced a novel way to represent numeric uncertainty at any point in the reachable search space. Our representation allows non-deterministic numeric effects to be drawn from any probability distribution, specified either in analytic form or as a collection of data samples. We described an efficient way to implement this representation that uses parallel computation and can make the most of GPU hardware. We integrated our approach with prior work on policy-building for non-deterministic planning, defining the regression operation through non-Gaussian effects.

While this is a work in progress and experimental evaluation is still pending, our research is an excellent opportunity to examine the precision / speed trade-off when it comes to generality. Accommodating general probability distributions requires less effort in terms of domain modelling and user input. Our work is able to take the most accurate probability distribution available (perhaps obtained by sampling data from previous runs), and make the most of that information, efficiently creating branches in non-deterministic outcomes as necessary to meet the certainty requirements.

According to our research, if information about uncertain probability distributions is available when writing the domain model, the planner should use it in its entirety rather than abstract it into a Gaussian distribution, or to a single median. With our implementation, it'll be tractable to do so, taking advantage of all available information.

If on the other hand information is not available to begin with, our approach is able to start out with a uniform or degenerate probability distribution, and refine it with time as more information is obtained (typically at plan running time).

5.2 Future Work

While at present the main application of our contribution is the policy-building process outlined in Section 3.5, our work can be used in the future to generate strong plans, in a similar vein to (Coles 2012) and (Marinescu and Coles 2016a).

In addition, our architecture allows the planner to not only generate a plan with θ certainty, but also to bump up θ to the highest value it can take under the given uncertain numeric effects. This is fairly straightforward to achieve – the query to the Plan Network that indicates success or failure can be modified to instead return the number of successful sample runs out of the total ones attempted.

The opposite of the above is also achievable in a similar fashion. If a solution with certainty θ is not found, our planner can be modified to return an alternative, lower value of θ for which a solution is found.

Another highly promising avenue for future work is learning probability distributions during execution. We can start out with a rough idea of a probability distribution – perhaps a Gaussian or a degenerate one if we lack any insight about the problem. We can then refine it to a more accurate distribution in a live feedback loop during plan execution. Our architecture allows changing distribution samples and parameters easily, so we expect the learning aspect to be a major selling point of our future work.

Acknowledgements

We would like to thank Amanda Coles for her insight into branching in the presence of uncertainty, and also for her critical analysis of our work.

Liana Marinescu's research is funded by a scholarship awarded by the Department of Informatics at King's College London.

References

Babaki, B.; Guns, T.; and Raedt, L. D. 2017. Stochastic constraint programming with and-or branch-and-bound. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*.

Beaudry, E.; Kabanza, F.; and Michaud, F. 2010. Planning with concurrency under resources and time uncertainty. In *Proceedings of the Nineteenth European Conference on Artificial Intelligence*.

Beck, J. C., and Wilson, N. 2007. Proactive algorithms for job shop scheduling with probabilistic durations. *Journal of Artificial Intelligence Research*.

Coles, A. J. 2012. Opportunistic branched plans to maximise utility in the presence of resource uncertainty. In *Proceedings of the Twentieth European Conference on Artificial Intelligence*.

Gerevini, A.; Long, D.; Haslum, P.; Saetti, A.; and Dimopoulos, Y. 2009. Deterministic planning in the fifth International Planning Competition: PDDL3 and experimental evaluation of the planners. *Artificial Intelligence*.

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

Hoffmann, J. 2003. The Metric-FF planning system: Translating ignoring delete lists to numeric state variables. *Journal of Artificial Intelligence Research*.

Marinescu, L., and Coles, A. I. 2016a. Heuristic guidance for forward-chaining planning with numeric uncertainty. In *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling*.

Marinescu, L., and Coles, A. I. 2016b. Non-deterministic planning with numeric uncertainty. In *Proceedings of the Twenty-Second European Conference on Artificial Intelligence*.

Marinescu, L., and Coles, A. I. 2016c. Non-deterministic planning with numeric uncertainty. Technical report, King's College London.

Meuleau, N.; Benazera, E.; Brafman, R. I.; Hansen, E. A.; and Mausam, M. 2009. A heuristic search approach to planning with continuous resources in stochastic domains. *Journal of Artificial Intelligence Research*.

Muise, C. J.; Belle, V.; and McIlraith, S. A. 2014. Computing contingent plans via fully observable non-deterministic planning. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*.

Muise, C. J.; McIlraith, S. A.; and Beck, C. J. 2012. Improved non-deterministic planning by exploiting state relevance. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling*.

Santana, P.; Thiebaux, S.; and Williams, B. 2016. RAO*: an algorithm for chance constrained POMDPs. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*.

Reformulating Oversubscription Planning Tasks

Michael Katz
IBM Research
Yorktown Heights, NY, USA
michael.katz1@ibm.com

Vitaly Mirkis
Amazon Research
Haifa, Israel*
vitamin@amazon.com

Florian Pommerening
University of Basel
Basel, Switzerland
florian.pommerening@unibas.ch

Dominik Winterer
Unaffiliated
Germany†
dominik_winterer@gmx.de

Abstract

Most modern heuristics for classical planning are specified in terms of minimizing the summed operator costs. Heuristics for oversubscription planning (OSP), on the other hand, maximize the utility on states. In this work we aim to provide the grounds for the adaptation of existing heuristics for classical planning to the OSP setting. To this end, we reformulate the OSP task to a classical planning task extended with an additional operator costs function, reflecting the utility information fully. We exemplify how existing heuristics from classical planning can be adapted to such a setting with a merge-and-shrink heuristic and empirically validate the feasibility of our approach.

Introduction

The field of automated planning is concerned with the problem of finding a course of action satisfying certain predefined goals. While the classical planning problem requires achieving all goals, partial satisfaction planning relaxes this restriction, allowing to achieve a subset of the goals. As a result, even an empty plan is a trivial valid solution, and therefore the aim of partial satisfaction planning is to obtain solutions of best possible quality. In *net-benefit* planning (van den Briel *et al.* 2004), a subfield of partial satisfaction planning, the assumption is that the solution cost and state values are comparable. As a consequence, the solution quality is measured as the net difference between the value of the obtained end state and the solution cost. In *oversubscription planning* (Smith 2004), on the other hand, the solution cost and state values are assumed to be incomparable. Thus, to take the cost into account, a bound on the cost or a *budget* is introduced (Smith 2004), and the objective is to maximize the value of the obtained end state, while constraining the solution cost.

Heuristic search is among the best performing approaches to both classical and net-benefit planning, with many search guiding heuristics developed over the years. These heuristics are typically classified into four families: *abstractions*, (e.g., Culberson and Schaeffer 1998; Edelkamp 2001; Helmert *et al.* 2014; Katz and Domshlak 2010a), *delete relaxations*, (e.g., Bonet and Geffner 2001; Hoffmann and Nebel 2001; Keyder and Geffner 2008; Domshlak *et al.* 2015), *critical paths* (Haslum and Geffner 2000), and *landmarks*, (e.g., Richter *et al.* 2008; Karpas and Domshlak 2009; Helmert and Domshlak 2009; Keyder *et al.* 2010). The basic principle behind all these heuristics is the same – relaxing the task at hand to fit some tractable fragment of the planning problem. In net-benefit planning, these heuristics are often applied not directly to the net-benefit task, but to a reformulation into classical planning (Keyder and Geffner 2009).

In optimal oversubscription planning, however, not much work was focused on heuristic search, and the progress was somewhat slower. A significant performance improvement was first reported by Mirkis and Domshlak (2013). They exploited *explicit abstractions* (Edelkamp 2001), which are tractable due to their small size. The abstract oversubscription planning problems were additively composed into informative admissible estimates which are then used to prune states in a branch-and-bound search. The approach turned out to work well in practice: in some cases the search space was reduced by three orders of magnitude compared to the baseline algorithm. Later, Mirkis and Domshlak (2014) exploit the notion of *landmarks* for task reformulation, enriching the task with reachability information. Katz and Mirkis (2016) characterize tractable fragments of oversubscription planning tasks according to causal graph structure and variable domain sizes, and derive admissible estimates from these fragments. Unfortunately, even the simplest fragment under this characterization was found to be not solvable in polynomial time. Thus, additional restrictions are required to achieve tractability, similarly to the ones that were previously exploited in deriving heuristics for classical planning.

Our aim in this work is to lay grounds for adapting many existing and future heuristics for classical planning to oversubscription planning. In order to do that, similarly in spirit to what was done for net-benefit planning by Keyder and Geffner (2009), we suggest a reformulation of an oversubscription planning task to a classical planning task with two cost functions on operators. The first one corresponds to the original operator costs and is intended for restricting the set of feasible solutions. The second one corresponds to the net difference in state values. We then search for an optimal

*The participation in this work was done prior current position.

†The contribution to this work was done in a Master thesis at Universities of Basel and Freiburg.

feasible solution of the reformulated planning task according to the second cost function. Using merge-and-shrink abstraction heuristics (Helmert *et al.* 2014) as an example, we show how this reformulation can exploit existing heuristics. Another contribution of our work is the first attempt at standardizing the benchmark set for oversubscription planning. For that, we introduce additional sections to PDDL intended to specify state-additive utility functions and a cost budget. Further, we adapt the Fast Downward translator (Helmert 2006) to parse these sections, and we create a collection of oversubscription planning benchmarks from the classical STRIPS domains used in International Planning Competitions.

Background

In line with the SAS formalism[1] for deterministic planning (Bäckström and Klein 1991), a *planning task structure* is given by a pair $\langle V, O\rangle$, where V is a set of *state variables*, and O is a finite set of *operators*. Each state variable $v \in V$ has a finite domain $dom(v)$. A pair $\langle v, \vartheta\rangle$ with $v \in V$ and $\vartheta \in dom(v)$ is called a *fact*. A partial assignment to V is called a *partial state*. The subset of variables instantiated by a partial state p is denoted by $\mathcal{V}(p) \subseteq V$. Often it is convenient to view partial state p as a set of facts with $\langle v, \vartheta\rangle \in p$ iff $p[v] = \vartheta$. We say a partial state s is a *state* iff $\mathcal{V}(p) = V$. Partial state p is *consistent* with state s if s and p agree on all variables in $\mathcal{V}(p)$. We denote the set of states of a planning task structure $\langle V, O\rangle$ by S.

Each *operator* o is a pair $\langle \mathsf{pre}(o), \mathsf{eff}(o)\rangle$ of partial states called *preconditions* and *effects*. We assume that all operators are in SAS format i.e. $\mathcal{V}(\mathsf{eff}(o)) \subseteq \mathcal{V}(\mathsf{pre}(o))$ for all $o \in O$. An *operator cost* function is a mapping $\mathcal{C} : O \to \mathbb{R}$. While in classical planning the operator cost functions $\mathcal{C}$ are typically assumed to be non-negative, we emphasize that in general cost functions $\mathcal{C}$ can take negative values as well.

An operator o is applicable in a state $s \in S$ iff $s[v] = \mathsf{pre}(o)[v]$ for all $v \in \mathcal{V}(\mathsf{pre}(o))$. Applying o changes the value of each $v \in \mathcal{V}(\mathsf{eff}(o))$ to $\mathsf{eff}(o)[v]$. The resulting state is denoted by $s[\![o]\!]$. An operator sequence $\pi = \langle o_1, \ldots, o_k\rangle$ is applicable in s if there exist states $s_0, \cdots, s_k$ such that (i) $s_0 = s$, and (ii) for each $1 \leqslant i \leqslant k$, o_i is applicable in s_{i-1} and $s_i = s_{i-1}[\![o_i]\!]$. We denote the state s_k by $s[\![\pi]\!]$ and call it the end state of π.

Oversubscription Planning An *oversubscription planning (OSP) task* $\Pi_{\text{OSP}} = \langle V, O, s_I, \mathcal{C}, \mathsf{u}, \mathsf{B}\rangle$ extends a planning task structure $\langle V, O\rangle$ with an *initial state* $s_I \in S$, a non-negative operator cost function $\mathcal{C}$ and a *utility* function $\mathsf{u} : S \to \mathbb{R}^{0+}$, and a *cost bound* $\mathsf{B} \in \mathbb{R}^{0+}$.

An operator sequence π is called an s-plan for Π_{OSP} if it is applicable in s_I, and $\sum_{o\in\pi} \mathcal{C}(o) \leqslant \mathsf{B}$. We call an s_I-plan a *plan* for Π_{OSP}. By the value $\hat{\mathsf{u}}(\pi)$ of a plan we refer to the value of the end-state of π, that is, $\hat{\mathsf{u}}(\pi) = \mathsf{u}(s_I[\![\pi]\!])$. A plan π for Π_{OSP} is *optimal* if $\hat{\mathsf{u}}(\pi)$ is maximal among all the plans. While an empty operator sequence is a plan for every OSP task, the objective in oversubscription planning is to find a plan achieving a state of high utility and *optimal oversubscription planning* is devoted to searching for optimal plans only. In what follows, we restrict our attention to *additive utility function*, computed as a sum over the state facts. Such value functions have the form $\mathsf{u}(s) = \sum_{f\in s} \mathsf{u}'(f)$, where u' is a function mapping facts to non-negative real values. Slightly abusing the notation, we denote u' by u in the following.

A heuristic for the OSP task $\Pi_{\text{OSP}} = \langle V, O, s_I, \mathcal{C}, \mathsf{u}, \mathsf{B}\rangle$ over states S is a mapping $h : S \times \mathbb{R}^{0+} \mapsto \mathbb{R}^{0+} \cup \{\infty\}$ from state-budget pairs to a non-negative real value or infinity. The *perfect heuristic* h^* maps each state $s \in S$ and bound $\mathsf{b} \in \mathbb{R}^{0+}$ to the utility $\hat{\mathsf{u}}(\pi^*)$ of an optimal plan π^* for the OSP task $\langle V, O, s, \mathcal{C}, \mathsf{u}, \mathsf{b}\rangle$ or to $-\infty$ if no such plan exists. A heuristic h is admissible if $h \geqslant h^*$. Note that admissible heuristics *overestimate* the optimal utility instead of underestimating the optimal plan cost as in classical planning.

Multiple Cost Function Planning

We now present an extended classical planning formalism that limits the set of feasible solutions with secondary cost functions and can have negative values in the primary cost function.

Definition 1. *A multiple cost function (MCF) planning task is a tuple* $\Pi_{\text{MCF}} = \langle V, O, s_I, G, \mathcal{C}_0, \mathscr{C}\rangle$*, where* $\langle V, O\rangle$ *is a planning task structure and*

- s_I *is a state, called* initial state
- G *is a partial state, called* goal state
- $\mathcal{C}_0$ *is a cost function*
- $\mathscr{C} = \{\langle \mathcal{C}_i, \mathsf{B}_i\rangle \mid 1 \leqslant i \leqslant n\}$ *where* $\mathcal{C}_i$ *is a non-negative cost function and* $\mathsf{B}_i \in \mathbb{R} \cup \{\infty\}$.

We call the cost function $\mathcal{C}_0$ the primary cost function and each cost function $\mathcal{C}_i$ with $1 \leqslant i \leqslant n$ a secondary cost function. An operator sequence π is a *plan* for Π_{MCF} if G is consistent with $s[\![\pi]\!]$ and $\sum_{o\in\pi} \mathcal{C}_i(o) \leqslant \mathsf{B}_i$ for $1 \leqslant i \leqslant n$. A plan is optimal if it has minimal primary cost among all plans of Π_{MCF}. A *heuristic* for MCF planning task $\Pi_{\text{MCF}} = \langle V, O, s_I, G, \mathcal{C}_0, \mathscr{C}\rangle$ with states S is a mapping $h : S \times \mathbb{R}^{|\mathscr{C}|} \mapsto \mathbb{R} \cup \{-\infty, \infty\}$. The *perfect heuristic* h^* maps a state s and a vector of bounds $\mathbf{b}$ to the primary cost $\mathcal{C}_0(\pi^*)$ of an optimal plan π^* for the MCF planning task $\langle V, O, s, G, \mathcal{C}_0, \mathscr{C}'\rangle$, with $\mathscr{C}' = \{\langle \mathcal{C}_i, \mathbf{b}_i\rangle \mid \langle \mathcal{C}_i, \mathsf{B}_i\rangle \in \mathscr{C}\}$ or to ∞ if no such plan exists. A heuristic h is *admissible* if $h \leqslant h^*$.

A *classical planning task* is an MCF planning task $\Pi = \langle V, O, s_I, G, \mathcal{C}_0, \varnothing\rangle$ with $\mathcal{C}_0$ being non-negative. As the set of secondary cost functions only constrains the set of plans, every plan for an MCF task $\Pi_{\text{MCF}} = \langle V, O, s_I, G, \mathcal{C}_0, \mathscr{C}\rangle$ is also a plan for the classical planning task $\Pi = \langle V, O, s_I, G, \mathcal{C}_0, \varnothing\rangle$.

In classical planning, abstractions can be obtained by e.g. projecting the problem on a subset of its variables (Edelkamp 2001), or through a merge-and-shrink process (Helmert *et al.* 2007; 2014). One of the strengths of abstraction heuristics in classical planning is their low per-node computation time during search. For explicit abstractions, such as projections and merge-and-shrink, the computation

[1] Not to be confused with the more commonly used SAS$^+$ formalism (Bäckström and Nebel 1995).

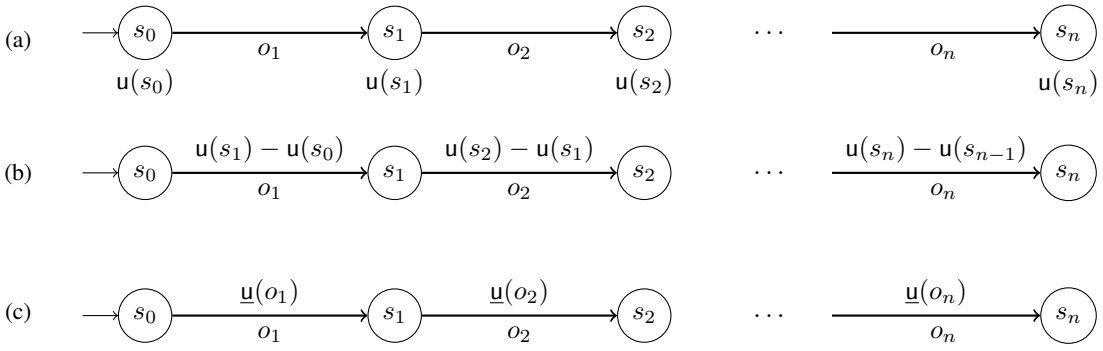


Figure 1: The figures show the idea behind reformulating an operator sequences with a state dependent utility function (a) into an operator sequences where a cost function reflects the utility difference between two successive states (b). The additive utility function allows for a state-independent cost function (c).

is basically a linear-time lookup. For implicit abstractions (Katz and Domshlak 2010a), the computation is more complicated, but is still of low polynomial time.

Abstractions for MCF planning generalize the definition for classical planning (Helmert *et al.* 2007) by additionally requiring reachable abstract state distances under the secondary cost functions to be below their respecive bounds. Formally, a *(labeled) transition system* (with multiple cost functions) is a tuple $\Theta = \langle S, L, \mathbf{c}, T, s_0, S_* \rangle$ where S is a finite set of states, L is a finite set of labels, $\mathbf{c} = \langle c_0, \cdots, c_n \rangle$ are functions $c_i : L \mapsto \mathbb{R}$ $(1 \leqslant i \leqslant n)$, $T \subseteq S \times L \times S$ a set of labeled transitions, s_0 the initial state and S_* the goal states.

The induced transition of an MCF task $\Pi_{\text{MCF}} = \langle V, O, s_I, G, \mathcal{C}_0, \mathscr{C} \rangle$ is the transition system $\Theta_{\Pi_{\text{MCF}}} = \langle S', L', \mathbf{c}', T', s'_0, S'_* \rangle$ where S' are the states of Π_{MCF}, $L' = O$, $\mathbf{c}_i(o) = \mathcal{C}_i(o)$, $(s, o, t) \in T'$ iff s is consistent with $\mathsf{pre}(o)$ and t is consistent with $\mathsf{eff}(o)$, s'_0 is the initial state of the planning task and S'_* are the goal states of the planning task. An abstraction is a mapping $\alpha : S' \mapsto S^\alpha$ where S^α are the states of the transition system $\Theta^\alpha = \langle S^\alpha, L, \mathbf{c}, T^\alpha, s_0^\alpha, S_*^\alpha \rangle$ with $T^\alpha = \{\langle \alpha(s), o, \alpha(t) \rangle \mid (s, o, t) \in T\}$, $s_0^\alpha = \alpha(s_0)$ and $S_*^\alpha = \{\alpha(s) \mid s \in S\}$. Θ^α is called the abstract transition system.

For this paper, we assume MCF tasks with at most one secondary cost function, i.e., having $|\mathscr{C}| \leqslant 1$.

Reformulation

We now show how to reformulate an OSP task into an MCF task. The key idea here is to compile the (additive) utility function into the primary cost function of an MCF planning task. We start by noting that for an additive state value function u, there is an easily computable finite upper bound

$$M := \sum_{v \in V} \max_{\vartheta \in dom(v)} \mathsf{u}(\langle v, \vartheta \rangle).$$

This upper bound allows us to switch from maximization to minimization of the utility value. Thus, our first step in the formulation is to switch to a new state value function $\underline{\mathsf{u}} : S \to \mathbb{R}^{0+}$ defined by $\underline{\mathsf{u}}(s) = M - \mathsf{u}(s)$, and the objective of the new task is to find a plan π *minimizing* the value $\hat{\mathsf{u}}(\pi)$. The idea behind our reformulation, illustrated in Figure 1, is to compute by how much each operator changes the utility of a state, if applied. In other words, for a state s and an operator o applicable in s, we compute the value $\underline{\mathsf{u}}(s, o) := \underline{\mathsf{u}}(s[\![o]\!]) - \underline{\mathsf{u}}(s)$.

Theorem 1. *The value $\underline{\mathsf{u}}(s, o)$ is independent of the state s.*

Proof. By definition of SAS, $\mathcal{V}(\mathsf{eff}(o)) \subseteq \mathcal{V}(\mathsf{pre}(o))$ for every operator $o \in O$. For a variable $v \in V \backslash \mathcal{V}(\mathsf{eff}(o))$, we have $s[v] = s[\![o]\!][v]$ and hence $\mathsf{u}(\langle v, s[v] \rangle) - \mathsf{u}(\langle v, s[\![o]\!][v] \rangle) = 0$. Therefore, it suffices to consider variables $v \in \mathcal{V}(\mathsf{eff}(o))$:

$$\begin{aligned}
\underline{\mathsf{u}}(s, o) &= (M - \mathsf{u}(s[\![o]\!])) - (M - \mathsf{u}(s)) \\
&= \sum_{v \in V} \mathsf{u}(\langle v, s[v] \rangle) - \mathsf{u}(\langle v, s[\![o]\!][v] \rangle) \\
&= \sum_{v \in \mathcal{V}(\mathsf{eff}(o))} \mathsf{u}(\langle v, s[v] \rangle) - \mathsf{u}(\langle v, s[\![o]\!][v] \rangle) \\
&= \sum_{v \in \mathcal{V}(\mathsf{eff}(o))} \mathsf{u}(\langle v, \mathsf{pre}(o)[v] \rangle) - \mathsf{u}(\langle v, \mathsf{eff}(o)[v] \rangle).
\end{aligned}$$

□

Thus, we can define a (state-independent) cost function over operators $\underline{\mathsf{u}} : O \to \mathbb{R}$ as

$$\underline{\mathsf{u}}(o) = \sum_{v \in \mathcal{V}(\mathsf{eff}(o))} \mathsf{u}(\langle v, \mathsf{pre}(o)[v] \rangle) - \mathsf{u}(\langle v, \mathsf{eff}(o)[v] \rangle).$$

Note that the cost function $\underline{\mathsf{u}}$ may have negative values. We say that an operator o *achieves utility* if $\underline{\mathsf{u}}(o) < 0$ and o *destroys utility* if $\underline{\mathsf{u}}(o) > 0$.

Theorem 2. *For a sequence of operators π applicable in state s, we have $\underline{\mathsf{u}}(s) + \sum_{o \in \pi} \underline{\mathsf{u}}(o) = \underline{\mathsf{u}}(s[\![\pi]\!])$.*

The proof is straightforward from the definition of $\underline{\mathsf{u}}$ on operators. Thus, finding a sequence of operators leading to a state with the minimal value $\underline{\mathsf{u}}$ corresponds exactly to finding a sequence of operators of a minimal summed cost $\underline{\mathsf{u}}$. We can thus solve the OSP task as a classical with multiple cost functions and an empty goal.

Definition 2. *Let* $\Pi_{\text{OSP}} = \langle V, O, s_I, \mathcal{C}, \mathsf{u}, \mathsf{B}\rangle$ *be an oversubscription planning task. The* multiple cost function reformulation $\Pi^{\mathscr{R}}_{\text{MCF}} = \langle V, O, s_I, G, \mathcal{C}_0, \{\langle \mathcal{C}, \mathsf{B}\rangle\}\rangle$ *of* Π_{OSP} *is the* MCF planning task*, where*

- $G = \varnothing$*, and*
- $\mathcal{C}_0(o) = \sum_{v \in \mathcal{V}(\mathsf{eff}(o))} \mathsf{u}(\mathsf{pre}(o)[v]) - \mathsf{u}(\mathsf{eff}(o)[v])$*, for* $o \in O$*.*

Theorem 3. *Let* Π_{OSP} *be an oversubscription planning task and* Π_{MCF} *its multiple cost function reformulation. If* π *is a plan of* Π_{OSP} *with utility* $\hat{\mathsf{u}}(\pi)$ *then* π *is a plan of* Π_{MCF} *with cost* $\mathcal{C}_0(\pi) = \mathsf{u}(s_I) - \hat{\mathsf{u}}(\pi)$ *and vice versa.*

Proof. Operator applicability is defined in the same way for Π_{OSP} and Π_{MCF}, so if π is a plan in one task, it is certainly applicable in the other task and ends in the same state, i.e. $s_I[\![\pi]\!]$ is well-defined and the same state in both tasks.

The operator sequence π respects the bounds of Π_{OSP} iff $\sum_{o\in\pi} \mathcal{C}(o) \leqslant \mathsf{B}$ iff π respects the bounds of the (only) secondary cost function of Π_{MCF}. Therefore, and because all states are goal states in Π_{MCF}, π is a plan in Π_{OSP} iff it is a plan in Π_{MCF}.

The primary cost of π is $\mathcal{C}_0(\pi) = \sum_{o\in\pi} \underline{\mathsf{u}}(o)$, which is equal to $\underline{\mathsf{u}}(s_I[\![\pi]\!]) - \underline{\mathsf{u}}(s_I) = \mathsf{u}(s_I) - \hat{\mathsf{u}}(\pi)$ according to Theorem 2. □

As $\mathsf{u}(s_I)$ is constant, a plan π maximizes $\hat{\mathsf{u}}(\pi)$ iff it minimizes $\mathcal{C}_0(\pi)$ and the following result directly follows:

Corollary 1. *An oversubscription planning task and its multiple cost function reformulation have the same optimal plans.*

Heuristics for OSP via Reformulation

Having proposed the OSP reformulation, we now turn our attention to devising heuristics for MCF planning. We start by clarifying how heuristics from MCF planning can be integrated into an OSP approach.

Definition 3. *Let* Π_{OSP} *be an OSP task,* Π_{MCF} *its multiple cost function reformulation, and* S *the states of* Π_{OSP}*. Let* $h_{\text{MCF}} : S \times \mathbb{R} \mapsto \mathbb{R} \cup \{-\infty, \infty\}$ *be a heuristic for* Π_{MCF}*. The* multiple cost function reformulation heuristic *of* h_{MCF}*, denoted by* $h^{\mathscr{R}}_{\text{MCF}}$ *is defined by* $h^{\mathscr{R}}_{\text{MCF}}(s, \mathsf{b}_s) = \mathsf{u}(s) - h_{\text{MCF}}(s, \mathsf{b}_s)$.

Multiple cost function reformulation heuristics are heuristics for OSP tasks. The following lemma establishes the connection between the informativeness of heuristics for MCF planning tasks and their multiple cost function reformulation heuristics.

Lemma 1. *For an OSP task* Π_{OSP} *and* $\Pi_{\text{MCF}} = \Pi^{\mathscr{R}}_{\text{MCF}}$*, we have* $h^*_{\Pi_{\text{OSP}}} = (h^*_{\Pi_{\text{MCF}}})^{\mathscr{R}}$.

The lemma is a direct outcome from Theorem 3. We use it in order to show the following main result.

Theorem 4. *Let* $\Pi_{\text{OSP}} = \langle V, O, s_I, \mathcal{C}, \mathsf{u}, \mathsf{B}\rangle$ *be an OSP task,* Π_{MCF} *its multiple cost function reformulation, and* h_{MCF} *an admissible heuristic for* Π_{MCF}*. Then* $h^{\mathscr{R}}_{\text{MCF}}$ *is an admissible heuristic for* Π_{OSP}.

Proof. Let h^*_{MCF} be the perfect heuristic for Π_{MCF} and h^*_{OSP} the perfect heuristic for Π_{OSP}. With Definition 3, we can rewrite $h^*_{\text{MCF}}(s, \mathsf{b}_s)$ as $\mathsf{u}(s) - (h^*_{\text{MCF}})^{\mathscr{R}}(s, \mathsf{b}_s)$, which is $\mathsf{u}(s) - h^*_{\text{OSP}}(s, \mathsf{b}_s)$ according to Lemma 1.

From Definition 3 we have

$$h^{\mathscr{R}}_{\text{MCF}}(s, \mathsf{b}_s) = \mathsf{u}(s) - h_{\text{MCF}}(s, \mathsf{b}_s),$$

and from admissibility of h_{MCF} we have

$$h_{\text{MCF}}(s, \mathsf{b}_s) \leqslant h^*_{\text{MCF}}(s, \mathsf{b}_s),$$

so

$$\begin{aligned} h^{\mathscr{R}}_{\text{MCF}}(s, \mathsf{b}_s) &\geqslant \mathsf{u}(s) - h^*_{\text{MCF}}(s, \mathsf{b}_s) \\ &= \mathsf{u}(s) - (\mathsf{u}(s) - h^*_{\text{OSP}}(s, \mathsf{b}_s)) \\ &= h^*_{\text{OSP}}(s, \mathsf{b}_s). \end{aligned}$$

□

Abstraction Heuristics for MCF Planning

Having established how admissible heuristics of MCF planning tasks can be exploited for deriving admissible heuristics of OSP tasks, we now show a concrete example of this by deriving a merge-and-shrink heuristic for OSP. We start by introducing a generic scheme for abstraction heuristics.

Definition 4. *Let* Π_{MCF} *be an MCF task,* α *be an abstraction and* Θ^α *its abstract transition system. The heuristic* $h^\alpha_\Theta : (S \times \mathbb{R}^n) \mapsto \mathbb{R} \cup \{-\infty, \infty\}$ *is the* MCF planning abstraction heuristic *of* Π_{MCF} *if it maps a state* $s \in S$ *and bounds* $\mathsf{b}_1, \ldots \mathsf{b}_n$ *to the cost of a path* ρ *in the abstract transition system* Θ^α*, such that*

- *for all* $1 \leqslant i \leqslant n$*,* $\mathcal{C}_i(\rho) \leqslant \mathsf{b}_i$*, and*
- ρ *is cost-minimal among such paths according to the primary cost* $\mathcal{C}_0$*.*

If no such path to an abstract goal state exists, the heuristic value is ∞*. Otherwise, if there exists such a path that contains a cycle of a negative total cost under* $\mathcal{C}_0$*, then the heuristic value is* $-\infty$*.*

For an MCF planning task with one secondary cost function, an abstraction heuristic $h^\alpha_\Theta(s, \mathsf{b})$ can be computed using the following scheme:

(I) Construct abstract transition system Θ^α,

(II) Compute shortest path distances from $\alpha(s_I)$ to all abstract states in Θ^α according to the secondary cost function $\mathcal{C}_1$ and discard abstract states with abstract distances strictly larger than b, and

(III) Compute shortest path distances from all remaining abstract states to some abstract goal state, according to the primary cost function $\mathcal{C}_0$.

There are essentially two challenges in turning this scheme into an abstraction heuristic. First, since the primary cost function is potentially negative, there might be reachable cycles of total negative cost in Θ^α resulting in a uninformative heuristic. Concrete choice of methods for constructing Θ^α in step (I) should aim at preventing or at least

alleviate this problem. In this work, we use existing methods for constructing merge-and-shrink abstractions (Sievers *et al.* 2014), leaving the methods for constructing abstractions that avoid negative cost cycles for future work.

The second challenge lies in the runtime complexity of heuristic computation. The reachable abstract states in step (II) depend on the budget b, and for maximizing the informativeness of the heuristic, step (III) should be performed for every evaluated state, given the reachability of abstract states under the budget b for that concrete state. Additionally, the possibly negative cost function madates the use of a shortest path algorithm that supports negative weights. Such algorithms are computationally more expensive than the typically used shortest path algorithms for non-negative weights. We alleviate this problem by performing the computation in step (III) only once, for reachability defined under the initial budget b_0.

Experimental Evaluation

To empirically evaluate the practical potential of our approach, we first create a benchmark set for oversubscription planning.

Creating a Benchmark Set for OSP

Since no official, publicly available benchmark set for oversubscrition planning is currently available, we had to create one. We created a benchmark suite similar to Domshlak and Mirkis (2015), based on the collection of classical International Planning Competition (IPC) domains. However, in contrast to previous approaches, we consider all planning tasks for which any solution is known, not only a provably optimal one. Such upper bounds on solution costs can be obtained from the information available at `planning.domains` (Muise 2016), a repository of planning benchmarks to which researchers are contributing meta-data on solved planning problems. We set the bounds for oversubscription planning tasks to either 25%, 50%, 75%, or 100% of the best known solution cost for the classical planning task, resulting in four variants for each classical planning domain. In the following, we refer to these numbers as different domain suites. Every fact in the goal of the classical planning task, we assigned the utility of 1, every other fact the value of 0.

We briefly describe how we modified the PDDL specification. We extended PDDL by two additional sections in the problem file. The first section (:BOUND contains the bound on the solution cost, while the second section contains the utility function. The second section (:UTILITY allows to provide a collection of function assignments of numeric values to grounded predicates, e.g., *(= (ON C B) 1)*. To translate the PDDL instances to a multi-valued formalism, we adapted the translator of the Fast Downward planning system to handle oversubscription planning tasks. Both the PDDL domain collection and the adapted translator are available on demand.

Transforming SAS$^+$ to SAS

Fast Downward translates PDDL into SAS$^+$ representation, which is more compact than SAS. Thus, to apply our techniques to the problems in our benchmark set, we need to transform these tasks to the SAS format. To achieve that, we need to modify operators with preconditions not specified for some effect variables. We used a procedure similar to the transition normalization (Pommerening and Helmert 2015) for this purpose. As the transition normalization increases the state space exponentially, we propose an optimization to moderate that increase. Note that our reformulation restricts the preconditions to be specified on effect variables only for variables with specified utility on at least one value. Thus, we do not modify the variables whose values do not have utilities specified.

	25%		50%		75%		100%	
Coverage	Bl	M&S	Bl	M&S	Bl	M&S	Bl	M&S
airport	**20**	9	**16**	9	**15**	9	**15**	9
miconic	85	85	**56**	55	**50**	49	45	45
mprime	**13**	12	**10**	9	7	7	6	6
mystery	10	10	**9**	8	7	7	7	7
scanalyzer08	**13**	12	12	12	**12**	11	**12**	11
scanalyzer11	**10**	9	9	9	**9**	8	**9**	8
tetris14	**17**	2	**14**	2	**10**	2	**8**	2
tidybot11	**20**	1	**20**	1	**16**	1	**13**	1
tidybot14	**20**	0	**17**	0	**12**	0	**6**	0
woodwork08	**25**	24	12	12	9	9	7	7
woodwork11	**18**	17	7	7	4	4	2	2
pipes-notank	**40**	18	**29**	18	**20**	17	14	**15**
pipes-tank	**28**	25	18	**19**	14	**15**	**11**	10
depot	15	15	8	**9**	6	**8**	4	**6**
openstacks08	29	**30**	24	**27**	23	**26**	22	**25**
openstacks11	20	20	17	**18**	17	**18**	17	**18**
openstacks14	19	19	10	**11**	5	**9**	3	**8**
parcprinter08	15	**16**	12	**13**	10	**12**	9	**10**
parcprinter11	11	**12**	8	**9**	6	**8**	5	**6**
parking11	10	10	1	**2**	0	**2**	0	**2**
parking14	11	11	0	**4**	0	**4**	0	**4**
satellite	8	**9**	6	6	3	3	3	3
Sum equal	587	587	450	450	376	376	349	349
Sum all	1044	953	765	710	631	605	567	554

Table 1: Per-domain coverage comparison of the blind heuristic (Bl) and merge-and-shrink (M&S) for the four domain suites. Top part depicts domains with advantage to the blind heuristic in all suites, middle part depicts domains with mixed results, while bottom parts shows domains with advantage to the merge-and-shrink heuristic in all suites.

Comparison to a Baseline

In our experiments, we compare different heuristics within a best-first branch-and-bound (BFBB) search, which we implemented in Fast Downward planning system. BFBB uses two heuristic functions. One is for choosing the next node to expand (guidance heuristic), and another one for pruning the nodes (pruning heuristic). We compare the following configurations differing in their pruning heuristic:

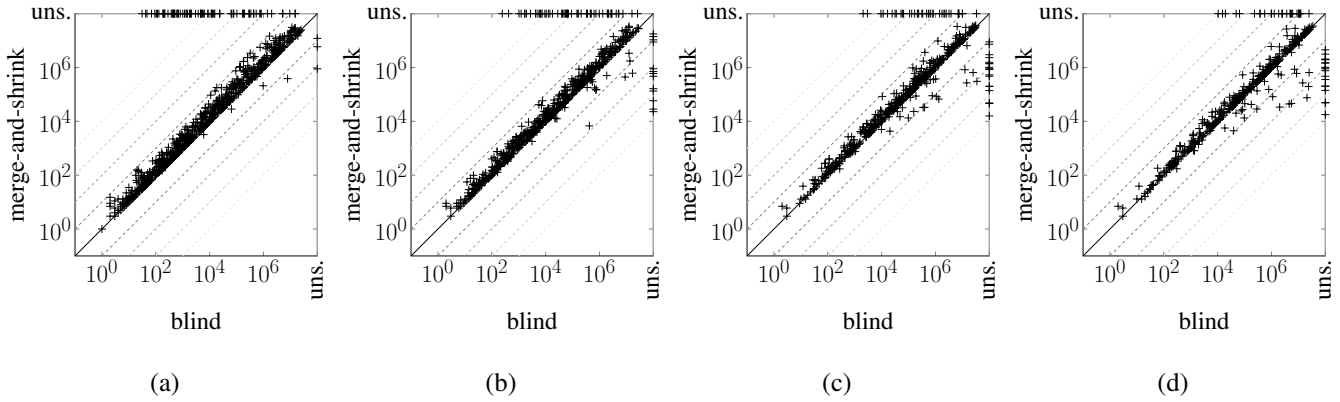


Figure 2: Comparison of the number of expansions performed with the blind and the merge-and-shrink heuristics for different problem suites, (a) 25%, (b) 50%, (c) 75%, and (d) 100%.

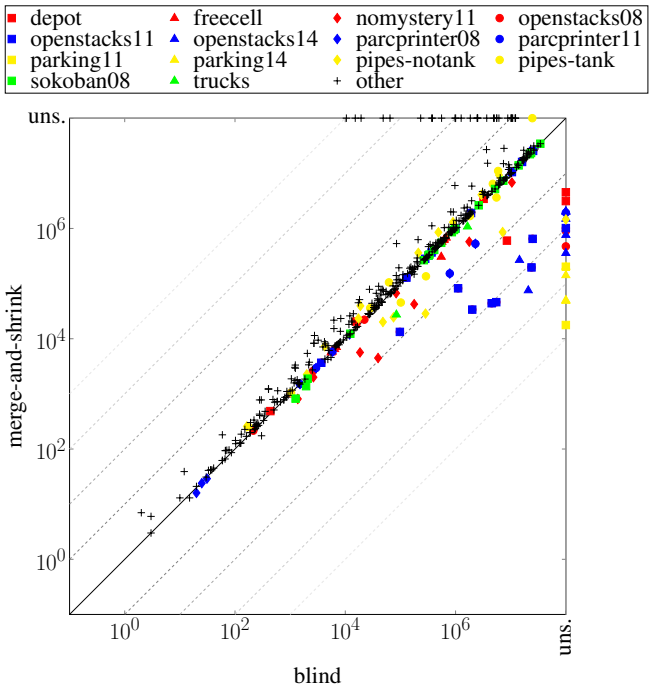


Figure 3: Domain-wise comparison of the number of expansions performed with the blind and the merge-and-shrink heuristics for the 100% problem suite. Domains where merge-and-shrink exhibits better performance in terms of the number of expansions are emphasized.

Bl Blind heuristic $h_{Bl}(s, \mathsf{b}) = M$

M&S A merge and shrink approach to compute $h^{\alpha}_{\Theta}(s, \mathbf{b})$. For step (I) we used the bisimulation based shrinking, and as merge strategy SCC-DFP (Sievers *et al.* 2014) according to secondary cost function $\mathcal{C}_1$. For step (III) we used the Bellman-Ford algorithm (Shimbel 1954) to compute (possibly negative) shortest path distances. For better runtime complexity, we do step (III) only once, with fixed budget B_0. The heuristic $h^{\alpha}_{\Theta}(s, \mathbf{b})$ is reformulated into an OSP heuristic according to Definition 3.

For a fair comparison, we set the guidance heuristic in all our approaches to the blind heuristic. To compare to previous state-of-the-art approaches to OSP, much work is still needed to adapt these techniques to work in an out-of-the-box fashion. For instance, the planner of Mirkis and Domshlak (2013) requires a specification of variable patterns to be used in their PDB heuristic. Similarly, the approach described in Mirkis and Domshlak (2014) also did not work out-of-the-box, since it is based on the previous one. However, the performance of these approaches is not too far from the simple blind heuristic, always returning the maximal utility, and therefore we use the blind heuristic as our baseline. The experiments were performed on Intel(R) Xeon(R) CPU E7-8837 @ 2.67GHz machines, with the time and memory limit of 30min and 2GB, respectively.

Results

Table 1 shows the per-domain coverage, comparing our approach to the baseline. On many domains, the performance of both approaches in terms of coverage is the same, for all suites. These rows are not shown in the table and summed in the "Sum equal" row. Overall, the baseline still achieves the higher coverage, with the difference getting smaller towards suites with larger cost bounds, namely, 91 for suite 25, 55 for suite 50, 26 for suite 75, and 13 for suite 100. We note that the domains AIRPORT, TETRIS, TIDYBOT11, TIDYBOT14, PIPESWORLD-NOTANKAGE, and PIPESWORLD-TANKAGE are responsible for most of the difference, due to the construction of merge-and-shrink abstraction not being finished within the time bound. With the exception of these 6 domains, merge-and-shrink loses at most one task in coverage per suite. Looking at the bottom part of the table, there are several domains where the performance improves significantly, across the suites. The improvement is getting larger towards suites with larger cost bounds. This is consistent with the overall results, hinting that merge-and-shrink would be beneficial for larger cost bounds.

In order to look beyond the coverage, Figure 2 depicts the comparison in terms of the number of node expansions

performed by the branch-and-bound search algorithm. The cost bound increases, from left to right. Figure 2 (a) shows the expansions for the suite 25, where there is a clear advantage to the blind heuristic, but as we move to larger bounds, the advantage becomes moderate, and then turns into somewhat complementary results in Figure 2 (d) for suite 100. Note that, in contrast to the classical optimal planning with A^*, here a dominating heuristic does not guarantee a smaller number of expansions. However, when the blind heuristic has a smaller number of expansions, it is always within one order of magnitude. For the other case, when merge-and-shrink dominates in the number of expansions, it can get to two orders, and more.

Further focusing on suite 100, the per-domain expansions can be observed in Figure 3. Improvement over the baseline can be observed in many domains, in particular in some domains where this improvement is not reflected in the overall coverage, probably due to the costly pre-search abstraction computation. These include FREECELL, NOMYSTERY, PIPESWORLD-NOTANKAGE, PIPESWORLD-TANKAGE, SOKOBAN08, and TRUCKS. There are additional 8 domains where the improvement in expansions is reflected in the coverage, namely DEPOTS, OPENSTACKS08, OPENSTACKS11, OPENSTACKS14, PARC-PRINTER08, PARC-PRINTER11, PARKING11, and PARKING14.

Conclusions and Future Work

In this work we have introduced a reformulation of an oversubscription planning task to a classical planning task with two cost functions on operators, allowing to ease the adaptation of the existing heuristics for classical planning to the oversubscription planning setting. We have shown with the merge-and-shrink heuristic how such an adaptation can be done. Our experimental evaluation shows the feasibility of such an approach. In order to perform the experimental evaluation, in the absense of a standard benchmark set and a PDDL fragment for describing oversubscription planning tasks, we have introduced such a fragment and created the benchmark set, as well as provided a translator from PDDL to a multi-valued variables formalism SAS, which is used internally by most modern planners. By adapting the Fast Downward planning framework, with many classical planning heuristics implemented, to oversubscription planning formalism we have simplified the future adaptation of classical planning heuristics to oversubscription planning via the suggested reformulation.

In future work we intend to investigate such adaptations. Further, we intend to investigate the interplay between the reformulation and heuristics additivity criteria, such as action cost partitioning (Katz and Domshlak 2008; 2010b) or disjointness for pattern databases (Haslum *et al.* 2007). We would also like to integrate and automate the approach of Mirkis and Domshlak (Mirkis and Domshlak 2013; 2014) and explore the connections between the reformulation and their approach. In addition, we would like to explore various heuristics for nodes ordering in the branch-and-bound search. Last, but not least, we would like to adapt the existing search pruning techniques for classical planning (Domshlak *et al.* 2012; Alkhazraji *et al.* 2012) to the branch-and-bound search over the oversubscription planning tasks.

References

Yusra Alkhazraji, Martin Wehrle, Robert Mattmüller, and Malte Helmert. A stubborn set algorithm for optimal planning. In Luc De Raedt, Christian Bessiere, Didier Dubois, Patrick Doherty, Paolo Frasconi, Fredrik Heintz, and Peter Lucas, editors, *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI 2012)*, pages 891–892. IOS Press, 2012.

Christer Bäckström and Inger Klein. Planning in polynomial time: the SAS-PUBS class. *Computational Intelligence*, 7(3):181–197, 1991.

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

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

Joseph C. Culberson and Jonathan Schaeffer. Pattern databases. *Computational Intelligence*, 14(3):318–334, 1998.

Carmel Domshlak and Vitaly Mirkis. Deterministic oversubscription planning as heuristic search: Abstractions and reformulations. *Journal of Artificial Intelligence Research*, 52:97–169, 2015.

Carmel Domshlak, Michael Katz, and Alexander Shleyfman. Enhanced symmetry breaking in cost-optimal planning as forward search. In Lee McCluskey, Brian Williams, José Reinaldo Silva, and Blai Bonet, editors, *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS 2012)*, pages 343–347. AAAI Press, 2012.

Carmel Domshlak, Jörg Hoffmann, and Michael Katz. Red-black planning: A new systematic approach to partial delete relaxation. *Artificial Intelligence*, 221:73–114, 2015.

Stefan Edelkamp. Planning with pattern databases. In Amedeo Cesta and Daniel Borrajo, editors, *Proceedings of the Sixth European Conference on Planning (ECP 2001)*, pages 84–90. AAAI Press, 2001.

Patrik Haslum and Héctor Geffner. Admissible heuristics for optimal planning. In Steve Chien, Subbarao Kambhampati, and Craig A. Knoblock, editors, *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling (AIPS 2000)*, pages 140–149. AAAI Press, 2000.

Patrik Haslum, Adi Botea, Malte Helmert, Blai Bonet, and Sven Koenig. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence (AAAI 2007)*, pages 1007–1012. AAAI Press, 2007.

Malte Helmert and Carmel Domshlak. Landmarks, critical paths and abstractions: What's the difference anyway? In Alfonso Gerevini, Adele Howe, Amedeo Cesta, and Ioannis

Refanidis, editors, *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, pages 162–169. AAAI Press, 2009.

Malte Helmert, Patrik Haslum, and Jörg Hoffmann. Flexible abstraction heuristics for optimal sequential planning. In Mark Boddy, Maria Fox, and Sylvie Thiébaux, editors, *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling (ICAPS 2007)*, pages 176–183. AAAI Press, 2007.

Malte Helmert, Patrik Haslum, Jörg Hoffmann, and Raz Nissim. Merge-and-shrink abstraction: A method for generating lower bounds in factored state spaces. *Journal of the ACM*, 61(3):16:1–63, 2014.

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

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

Erez Karpas and Carmel Domshlak. Cost-optimal planning with landmarks. In Craig Boutilier, editor, *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, pages 1728–1733. AAAI Press, 2009.

Michael Katz and Carmel Domshlak. Optimal additive composition of abstraction-based admissible heuristics. In Jussi Rintanen, Bernhard Nebel, J. Christopher Beck, and Eric Hansen, editors, *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS 2008)*, pages 174–181. AAAI Press, 2008.

Michael Katz and Carmel Domshlak. Implicit abstraction heuristics. *Journal of Artificial Intelligence Research*, 39:51–126, 2010.

Michael Katz and Carmel Domshlak. Optimal admissible composition of abstraction heuristics. *Artificial Intelligence*, 174(12–13):767–798, 2010.

Michael Katz and Vitaly Mirkis. In search of tractability for partial satisfaction planning. In Subbarao Kambhampati, editor, *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI 2016)*, pages 3154–3160. AAAI Press, 2016.

Emil Keyder and Héctor Geffner. Heuristics for planning with action costs revisited. In *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI 2008)*, pages 588–592, 2008.

Emil Keyder and Héctor Geffner. Soft goals can be compiled away. *Journal of Artificial Intelligence Research*, 36:547–556, 2009.

Emil Keyder, Silvia Richter, and Malte Helmert. Sound and complete landmarks for and/or graphs. In Helder Coelho, Rudi Studer, and Michael Wooldridge, editors, *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI 2010)*, pages 335–340. IOS Press, 2010.

Vitaly Mirkis and Carmel Domshlak. Abstractions for oversubscription planning. In Daniel Borrajo, Subbarao Kambhampati, Angelo Oddi, and Simone Fratini, editors, *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling (ICAPS 2013)*, pages 153–161. AAAI Press, 2013.

Vitaly Mirkis and Carmel Domshlak. Landmarks in oversubscription planning. In Torsten Schaub, Gerhard Friedrich, and Barry O'Sullivan, editors, *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI 2014)*, pages 633–638. IOS Press, 2014.

Christian Muise. Planning.Domains. In *26th International Conference on Automated Planning and Scheduling, System Demonstrations and Exhibits*, 2016.

Florian Pommerening and Malte Helmert. A normal form for classical planning tasks. In Ronen Brafman, Carmel Domshlak, Patrik Haslum, and Shlomo Zilberstein, editors, *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2015)*, pages 188–192. AAAI Press, 2015.

Silvia Richter, Malte Helmert, and Matthias Westphal. Landmarks revisited. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI 2008)*, pages 975–982. AAAI Press, 2008.

Alfonso Shimbel. Structure in communication nets. *Proceedings of the symposium on information networks*, 4, 1954.

Silvan Sievers, Martin Wehrle, and Malte Helmert. Generalized label reduction for merge-and-shrink heuristics. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI 2014)*, pages 2358–2366. AAAI Press, 2014.

David E. Smith. Choosing objectives in over-subscription planning. In Shlomo Zilberstein, Jana Koehler, and Sven Koenig, editors, *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 2004)*, pages 393–401. AAAI Press, 2004.

Menkes van den Briel, Romeo Sanchez, Minh B. Do, and Subbarao Kambhampati. Effective approaches for partial satisfaction (over-subscription) planning. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI 2004)*, pages 562–569. AAAI Press, 2004.