Proceedings of the 5th Workshop on
Heuristics and Search for Domain-Independent Planning

HSDIP 2013

Rome, Italy - June 10, 2013

Edited By:

Malte Helmert, Michael Katz, Gabriele Röger, Jordan Thayer

Organizing Committee

Malte Helmert
University of Basel, Switzerland

Michael Katz
Saarland University, Germany

Gabriele Röger
University of Basel, Switzerland

Jordan Thayer
SIFT, USA

Program Committee

J. Benton, Arizona State University
Blai Bonet, Universidad Simón Bolívar
Ronen Brafman, Ben-Gurion University of the Negev
Carmel Domshlak, Technion
Alan Fern, Oregon State University
Héctor Geffner, ICREA & Universitat Pompeu Fabra
Patrik Haslum, Australian National University
Jörg Hoffmann, Saarland University
Rob Holte, University of Alberta
Adele Howe, Colorado State University
Erez Karpas, Technion
Thomas Keller, University of Freiburg
Emil Keyder, Google
Scott Kiesel, University of New Hampshire
Levi Lelis, University of Alberta
Derek Long, King's College London
Wheeler Ruml, University of New Hampshire
Rong Zhou, PARC
Sandra Zilles, University of Regina

Foreword

Heuristics and search algorithms are the two key components of heuristic search, one of the main approaches to many variations of domain-independent planning, including classical planning, temporal planning, planning under uncertainty and adversarial planning. 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.

The workshop on Heuristics and Search for Domain-Independent Planning (HSDIP) is the fifth workshop in a series that started with the Heuristics for Domain-Independent Planning (HDIP) workshops at ICAPS 2007, 2009 and 2011. At ICAPS 2012, the workshop was held for the fourth time and was changed to its current name and scope to explicitly encourage work on search for domain-independent planning. It was very successful under both names. Many ideas presented at these workshops have led to contributions at major conferences and pushed the frontier of research on heuristic planning in several directions, both theoretically and practically. The workshops, as well as work on heuristic search that has been published since then, have also shown that there are many exciting open research opportunities in this area. Given the considerable success of the past workshops, and since it has de facto become an annual event, we intend to continue holding it annually.

The main focus of the HSDIP workshop series is on contributions that help us find a better understanding of the ideas underlying current heuristics and search techniques, their limitations, and the ways for overcoming them. While the workshop series has originated mainly in classical planning, it is very much open to new ideas on heuristic schemes for more general settings, such as temporal planning, planning under uncertainty and adversarial planning. Contributions do not have to show that a new heuristic or search algorithm "beats the competition". Above all we seek crisp and meaningful ideas and understanding. Also, rather than merely being interested in the "largest" problems that current heuristic search planners can solve, we are equally interested in the simplest problems that they cannot actually solve well.

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

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

June 2013 — Malte Helmert, Michael Katz, Gabriele Röger, and Jordan Thayer.

Table of Contents

In Situ Selection of Heuristic Subsets for randomization in IDA* and A*

Santiago Franco and **Mike Barley** and **Pat Riddle**
Computer Science Department
Auckland University
Auckland, New Zealand

Abstract

The performance of optimal heuristic search algorithms, e.g. IDA* or A*, is critically dependent on the quality of the available heuristics. Most heuristics use abstractions to simplify the problem, and then solve this simplified problem to produce a cheap and admissible estimate of the actual distance to the goal. However, as far as we know, there is no universal technique to automatically and efficiently choose the best abstraction, and hence heuristic, for every specific problem instance. By efficiently we mean that the time it takes to choose the best abstraction does not offset its resulting savings, compared to simply combining, via maximization or randomization, the available abstractions for the domain.

In this paper we propose using a novel analytical model which efficiently chooses good subsets for the current problem instance. Specifically, we want to create a model of the problem-solver that, when given a set of domain-specific heuristics and an F-bound, can rank subsets of those heuristics based on the model's predictions of the problem solvers' run-times. Specifically, in this paper we consider combining heuristics using randomization, i.e. when the problem-solver randomly selects a heuristic from the chosen heuristic subset at every node. We compare the results versus using randomization over all the available heuristics.

Introduction

Optimal heuristic search algorithms, e.g. A*, use admissible heuristic estimates to reduce the number of states which must be evaluated before a guaranteed optimal path to a goal state is found. Most heuristics use some form of abstraction function to simplify the problem, and then solve this simplified problem to produce a cheap and admissible estimate of the distance to the goal. A significant amount of research goes into finding the best possible abstractions for specific problem domains. However, there is no universal technique to guarantee that the chosen abstraction gives the best balance between accuracy vs associated computational costs for all domains or even for all problems in complex domains. Generally, the larger the abstraction, the more accurate it is. But also the larger the abstraction, the more expensive it is in terms of either memory, computational costs or both.

Franco et al. (2009) proposed an analytical model called *RIDA**, which given a set of domain-competitive heuristics, automated heuristic selection on a problem basis. *Domain competitive heuristics* are selected to do well on average for the domain, but they are not guaranteed to be the best for each problem. RIDA* aimed to efficiently automate finding good problem-specific heuristics, while keeping its own meta-reasoning costs low. That paper used the standard admissible heuristic combination method, i.e. taking the maximal heuristic estimate for each node.

Zahavi et al. (2007) introduced a new heuristic combination method, picking out a heuristic at random instead of taking the maximal value. This was presented as a way to sidestep the *Diminishing Returns property*, i.e. each additional heuristic increases the overall computational costs, but the more heuristics used, the less the search space is pruned by using additional heuristics. In this paper we show how randomization only works well for those sets of heuristics which are designed to complement each other.

The main contribution of this paper is to present an analytical model which, given a set of domain-competitive heuristics, efficiently and online selects good heuristic subsets for randomization for each problem instance. By *efficiently* we mean that, including sampling and modeling costs, the overall solving time is, on average, better than simply randomizing the whole set. By *online* we mean that the selection of a heuristic subset is made while solving the problem.

The solution this paper proposes follows the same steps as in Franco et al. (2009), this is to divide the search for a solution into three steps: Sampling Phase, Prediction Phase and Solving Phase. In the *Sampling Phase* the problem is sampled using the initial IDA* iterations and two data structures called Heuristic Union Search Tree (HUST) and Culprit Counters (CCs). The *HUST* is a single search tree which subsumes each of the possible search trees[1] generated when

[1]Given a set of H available heuristics, there are up to 2^H heuristic combinations, and hence 2^H search trees which are subsumed into a single search tree.

using a different heuristic combination. This is achieved by changing the regular IDA* expansion rule, "prune any node if at least one heuristic prunes it", to exactly the opposite, "expand any node as long as at least one heuristic would expand it". This data is compressed further by the *CC*s, which use binary bits to index the number of times each combination of heuristics expanded a node. For instance, given a set of three heuristics, $CC(100) = 3$ means that only three nodes in the HUST were generated by parent nodes on which the first heuristic was bellow the F-bound but pruned by the other two (their F-value was higher than the current F-bound). In the second phase, the *Prediction Phase*, we present a new analytical model to use the CCs themselves to estimate which is the best subset of heuristics to use for randomization. This is the main change proposed in this paper to the RIDA* architecture. Finally, in the *Solving Phase*, RIDA* uses either A* or IDA* to solve the rest of the problem with the selected heuristic subset, using randomization as in Zahavi et al. (2007).

We will review the benefits and disadvantages of the proposed model both for A* (Hanoi Tower) and IDA* (Fifteen and Twenty-Four sliding tile puzzles) in the Experiments section. For the Fifteen Puzzle Domain we show that RIDA*'s use of the new prediction model is competitive even when the domain set-up is not ideal[2]. For the Twenty-four Puzzle, we show that RIDA* can find subsets which are significantly better when the available heuristics that are selected solely because each of them does well on average for the domain. In the Twenty-four puzzle we achieve a reduction of $\frac{1}{4}$ in the number of generated nodes and a reduction of $\frac{1}{5}$ in the time for the 100 heuristic set (Table 3). We also show the main caveat of the proposed model in the Towers of Hanoi. This is that RIDA* has no monitoring mechanism once it has chosen a heuristic subset. This is done in order to keep meta-reasoning costs low but it can lead to making poor choices.

Finally, note that the presented prediction model's objective is to maximize the trade-off between doing enough sampling and modeling to find a good subset while keeping the overall costs low enough to actually benefit from that selection. Also note that the main objective of this paper is not to find the fastest technique, be it randomization or maximization, but to build models which can efficiently point to the most efficient combination of heuristics for the current problem.

Literature Review

This paper is introducing a new analytical model, based on the data structures (HUST and CCs) presented in Franco et al. (2009), to predict the performance of a subset of heuristics when using randomization. In the last decade there has been significant research into improving heuristic performance prediction. Most of the work is based on the *KRE* model (Korf, Reid, and Edelkamp 2001), which is based on gathering statistics for the domain. The original KRE model is very accurate when assessing overall heuristic performance on a domain basis but it is significantly inaccurate for individual problem instances (Zahavi et al. 2008). Zahavi et al(2008) came up with changes to the KRE formula to make it more accurate for individual problems, but this came at the cost of even more computationally intensive data gathering.

KRE-based methods require extensive data gathering for each heuristic being compared. In this paper we are interested in comparing thousands of heuristic combinations, e.g. a twenty-five heuristic set has up to $2^{25} = 3.37 * 10^7$ combinations. It is impractical to gather KRE statistics on each possible heuristic combination. Note that individual heuristic predictions using the KRE model cannot be combined as if they were independent of the actual state being evaluated. This is clearly not the case for sets of high quality admissible heuristics. A new separate KRE model needs to be built for each possible heuristic combination. Hence a cheaper prediction model, albeit possibly more inaccurate, is needed to efficiently choose between a large number of possible heuristic combinations. Fortunately, we only need to be accurate enough to do a good ranking of the best heuristic combinations. Note that KRE's strengths and weaknesses, in the context of maximization as a combination method, was discussed in more detail in Franco et al. (2009). The same applies in the context of randomization.

Optimal search

There are many papers on how to generate the best abstractions-based heuristics, e.g. iPDB (Haslum et al. 2007), Merge-and-Shrink (Helmert et al. 2007), there are surprisingly few examples attempting to efficiently and automatically combine different types of admissible heuristics online in the context of optimal search[3].

All the heuristic generation procedures mentioned in the previous paragraph do a restricted online heuristic selection, in terms of which abstractions are likely to reduce most the search space. The main difference with our approach is that RIDA* is using a parametric model to estimate which heuristic combination will solve the problem faster. Generating the lowest amount of nodes does not guarantee the fastest solving time, otherwise maximizing all available heuristics would always be the fastest option. Even though it is not done in this paper, RIDA* can manage heuristics with very

[2]Our approach performs best when the heuristics are selected solely on their individual performance on problems in the domain. Any further selection, based on how well the heuristics complement each other, is RIDA*'s main purpose. Hence, RIDA* is more amenable to heuristic sets which have been generated without taking into account how well each heuristic in the set complements the rest.

[3]Note that Portfolio planners allocate time slices to different planners. The individual planners are akin to heuristics. Our approach is different because we are combining heuristics, via randomization, to improve the overall efficiency of the search algorithm. Portfolio-based planners have no mechanism to combine heuristics collaboratively, e.g. maximization or randomization.

different evaluation times, e.g. online calculated heuristics like LM-CUT versus pre-calculated abstractions like PDBs or even arithmetic heuristics like Manhattan Distance[4]. The second difference is that RIDA* is designed to be competitive with simply maximizing or randomizing a set of heuristics which are known to do well for the current domain. To achieve this, RIDA* has developed two mechanisms (the HUST and the Culprit Counters) which optimize *in situ* sampling and selection costs.

The only other system we found which makes efficient online heuristic selections in the context of optimal search, given a set of heuristics, is (Domshlak, Karpas, and Markovitch 2010). It is based on using Bayesian classifiers to select heuristics on a state by state basis. It has in common with RIDA* that it does online sampling, and that it aims to keep the costs (the costs of making the choices plus the cost of solving the problem with those choices) lower than solving the problem with the default option (maximization for them).

The first important difference between Domshlak et al. (2010) and RIDA* is that they associated online meta-reasoning costs for every generated node. RIDA* uses the early iterations of a problem to gather data up to an empirically determined sampling cap. Once RIDA* has made a selection for the current problem, there are no meta-reasoning costs for the remaining iterations. Of course this also means that RIDA* cannot adapt if its initial sampling is not representative of the remaining iterations.

Secondly, their system is designed to be used with heuristics which have a significant online calculation overhead. This helps ameliorate the fact that the added meta-reasoning increases the average overhead for all generated nodes. It would be harder for such a system to be efficient when using pre-calculated heuristics. RIDA*'s sampling mechanism is designed to be efficient when using heuristics with a small online overhead, e.g. pre-calculated abstractions like Pattern Databases (*PDBs*). PDBs are discussed in the experiments section.

Finally, their heuristic set is made of only two heuristics. Their paper states that the number of classifiers it needs to update online, and presumably the overhead costs, grow quadratically with respect to the number of heuristics in the set. This means that for large heuristic sets, e.g. we use up to a 100 heuristics for the Twenty-four Puzzle and 105 heuristics for the Towers of Hanoi, their online meta-reasoning costs would increase by four orders of magnitude. That would make their listed 2% meta-reasoning costs, with respect to the average overhead per node, too expensive for the larger heuristic sets used for the experiments in this paper. RIDA*'s online meta-reasoning costs grow linearly, instead of quadratically, with respect to the number of heuristics in the set (Franco and Barley 2009).

[4]We have a yet unpublished manuscript describing how to use RIDA* to combine heuristics, using maximization, whose evaluation costs differ by orders of magnitude. This is done in the context of domain independent planners.

The advantages of this paper's proposed selection mechanism for randomization is that it is designed to balance the need for good heuristic rankings while keeping meta-reasoning costs low. The actual solution depends on the problem features, e.g. if choosing between two heuristics on a domain basis we would recommend to use the formulas in Zahavi et al. (2008). If choosing between small sets of computationally intensive heuristics like in Domshlak et al. (2010) on a state-basis we believe their method is state-of-the-art. We claim that our heuristic selection method improves cost-effective automated online heuristic selection when dealing with large sets of heuristics which are cheap to evaluate. This is the case with the very effective family of PDB-based heuristics but it would also apply to any pre-calculated abstraction based heuristics.

Nonoptimal search

There are more methods which do heuristic selection in the context of non-optimal search. Even though we are not running non-optimal searches in this paper, we list a couple of the most recent approaches for completeness sake. Here we will briefly discuss two recent approaches: dovetailing(Valenzano et al. 2010) and bootstrapping (Jabbari Arfaee, Zilles, and Holte 2010).

Dovetailing was used by Valenzano et al. to significantly improve the performance of WIDA*. Dovetailing differs from classic portfolios because instead of giving a fixed amount of time for each planner, each planner is run in lockstep in terms of number of steps. As the authors noted, this approach will be —A— times more expensive than using the best setting, A being the number of settings being tried. But, as the best problem-specific setting is unknown, dovetailing works quite well compared to using any fixed setting for the domains and heuristics used in the paper. The reason it works well is that dovetailing does not get stuck in local minima.

Dovetailing can be compared to the HUST. The HUST is a compressed search tree which contains all the reachable nodes by any of the available heuristics. This would also apply to the weighted heuristics considered in their paper. Each weight is simply a new heuristic. It would be interesting to check whether using the HUST, in the context of dovetailing, could speed up the search. The reason it might is that the HUST would not duplicate node generation, as dovetailing does. But no claims can be made unless actual experiments are run.

Bootstrapping was used by Jabbari et al. (2010) to generate a non-optimal heuristic for each domain. It is intended to generate a good heuristic, almost as good as state of the art, when the only domain information available is a successor function(opaque domain). Time performance is indirectly taken into account, i.e. a new heuristic is only learnt if it solved enough new bootstrapped instances under a fixed time limit. It can take a large amount of time to generate a good heuristic (two days for Twenty-four puzzle), but then it can be reused to solve different problems with the same

goal. The main difference with RIDA* is that RIDA* tailors its heuristic selection to each specific problem, given that an existing set of competitive admissible heuristics is given. Hence, RIDA* has to optimize its sampling procedure (HUST and CCs) to take into account the trade-off between sampling effort vs improved speed up compared to simply combining all available heuristics. Also, RIDA* uses a parametrized model to estimate which heuristic combination is likely to solve the current problem faster. Bootstrapping is not intended to speed-up search if strong heuristics are already known for the domain.

Proposed Prediction Model

Example Tree

We will be referring to an example search tree (Fig. 1a) to explain the model. The model's objective is to estimate the number of generated nodes, for any subset, if randomization had been used for the current IDA*'s F-bound. RIDA* then uses the estimated generated nodes for each subset, together with the sampled Heuristic Branching Factor (*HBF*)(Korf, Reid, and Edelkamp 2001), to predict the number of generated nodes and ultimately search time for the next F-bound/s. The search tree in Fig. 1a is a HUST. For each sampled iteration RIDA* generates a single search tree which subsumes all possible search trees for all heuristic subsets. The HUST is a snapshot of the search at the end of some specific F-bound expansion. This is just a small handmade example to illustrate how the proposed randomization prediction model works. Each node is identified by its path, e.g. root's left child is called "L" and L's right child is called "LR". Note that the node also contains which heuristics expanded it (1) and which prune it (0), e.g. 100 means that only h_1 expands this node.

Regular Probabilistic Model

The most direct and precise estimation of the number of generated nodes for an IDA* iteration using randomization is to store the expansion probability for each reachable node, given that the parent node was generated. This is done in the probabilistic tree for our example (Fig. 1b). We assume all heuristics in a subset have the same chance of being picked, so the probability of a node being expanded, given that its parent was generated, is simply the number of heuristics which expand it over the total number of heuristics in the subset.

Equation (1) is the most accurate estimation we can make of the expected number of generated nodes when using randomization. Explanation follows: The number of generated nodes when combining heuristics using maximization is constant on every run. However, every time a randomized heuristic selection is run, different nodes might be expanded. The HUST contains all the reachable nodes by any individual heuristic for a given F-boundary and hence any random combination of the heuristics. Equation (1) models the randomization process, e.g. the probability of a node being expanded is the product of all the ancestor's probabilities being expanded. As we cannot predict which specific nodes will actually be expanded on a randomized run, we instead model the average number of nodes generated. Short of the actual randomized run, this is the most informed prediction which can be made. Note that as we use very large search trees for our experiments, the stochastic effect on the actual number of generated nodes in different runs is quite small.

$$N(H) = 1 + \sum_{\forall n \in HUST} P_{gen}(n) * P_{exp}(n) * BF(n) \quad (1)$$

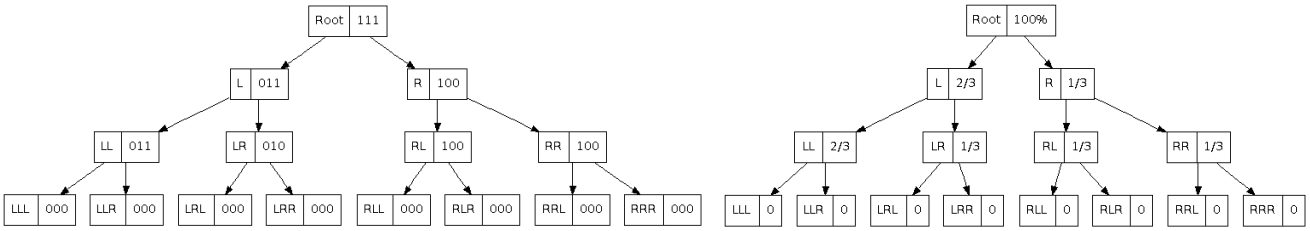
$$P_{gen}(n) = \prod_{\forall i \in HUST which\ are\ n's\ Ancestors} P_{exp}(i) \quad (2)$$

$$P_{exp}(i) = \frac{NumberExpandingHeuristics(i) \in H}{NumberOfHeuristics \in H} \quad (3)$$

$N(H)$ is the number of generated nodes, given a heuristic subset (H). n is an iterator for all nodes $\in$ HUST (The HUST contains all reachable nodes, for a specific F-boundary, for any of the heuristics in H). $BF(n)$ is node's n's branching factor (number of children). $P_{gen}(n)$ is the probability of generating the node n, which is the product of all its ancestor's expansion probabilities all the way to the root. i is an iterator of all nodes which are an ancestor of the node n. $P_{exp}(i)$ is the probability to expand ancestor node i, given that its parent was expanded. $NumberExpandingHeuristics(i) \in H$ is the number of heuristics in subset H which were below the F-boundary for node i. $NumberOfHeuristics \in H$ is the total number of heuristics present in H.

The problem with equation (1) is that, although it is very accurate, it also requires storing every node in the HUST, it is too memory intensive to be practical for large search trees. Regular RIDA* (Franco and Barley 2009) uses CCs to keep track of how many nodes were generated for each subset when doing maximization, see Table 1. For any subset the exact number of generated nodes, when using maximization, is the sum of those CC's whose expanding heuristics (characteristic set value=1) are *active* for the heuristic subset.

RIDA* cannot use the CCs to estimate the effect of randomization with the same accuracy as when doing maximization, as previously explained. It cannot use equation (1) either, because CCs cannot be used to calculate the exact conditional expansion probabilities of individual nodes, e.g. the probability of expanding node "LR" depends on the probabilities of expanding its two ancestors (Root and L). CCs compress the HUST by eliminating all node location information. Note that the CCs provide us with information on how well heuristics combine with each other on average, as we know how many nodes are expanded by which combination of heuristics. So we know for example, how frequently any subset of heuristics "agree" on pruning or expanding a node. This information is not available from probability distributions of individual heuristics. The following prediction can use this information to predict which heuristic subsets will do well for the current problem.



(a) Arbitrary HUST for a Three Heuristic set (h_1, h_2, h_3) (b) Probabilistic Tree for Same Three heuristic Set (h_1, h_2, h_3)

Figure 1: Example Trees for a Three Heuristic Set (h_1, h_2, h_3)

Populated Culprit Counters	
Characteristic Set	*Associated Children Nodes*
111	$CC(111)$=2
011	$CC(011)$=4
100	$CC(100)$=6
010	$CC(010)$=2
Generated Nodes When Using Maximization	
Heuristic Subset	*Generated Nodes*
$h1\&h2\&h3$	$1 + \sum$CC(111)= 3
$h1\&h2$	$1 + \sum$CC(11X)= 3
$h1$	$1 + \sum$CC(1XX)= 9
...	...

Table 1: Culprit Counters For Example Fig. 1a

Consistent CC's Prediction Model

Hereafter we show how to use RIDA*'s CCs to estimate, given a set of heuristics, the number of generated nodes for an IDA*'s iteration when using randomization. The suggested solution only requires an additional parameter (4) in order to calculate the HUST's average depth.

$$AvgHUSTDepth = \frac{\sum_{\forall i \in PrunedNodes \in HUST} depth(i)}{|PrunedNodes \in HUST|} \quad (4)$$

We keep using the example HUST (Fig. 1a,), whose average depth is 3 (Root's depth is 0). The proposed model's equations to translate each CC to the average number of nodes generated for a given heuristic subset when doing randomization are equations (5-8).

This model is an adaptation of (1) to estimate the number of generated nodes for any heuristic subset when the only information available is the CCs and the average HUST's depth. Following is a description of the terms: H compactly represents which heuristics are active for the current subset. Each heuristic ($h_x \in H$) is marked as "1" iff active, "0" otherwise. X is the CC's binary characteristic set, each bit is marked as "1" iff h_x expands the associated nodes, "0" otherwise. $CC(X)$ is the number of nodes in the HUST for which the same CC applies. $P_{Gen|CC(X)}$ is the prorated generation probability of $CC(X)$'s nodes for each depth. $P_{Exp|CC(X)}$ is the probability that nodes associated with $CC(X)$ are expanded. Y represents those CCs consistent with $CC(X)$, i.e. the set of CCs in Y are *consistent* with X if and only if their characteristic sets only have 1s in a location where $CC(X)$'s characteristic set also has a 1. $P_{Avg|CC(X)}$ is the average $CC(X)$'s ancestor's expansion probability if we assume the heuristics are consistent. $X \cap H$ is the characteristic set resulting from all those CCs whom have a '1' in at least one bit location which is the same for H.

$$N(H) = 1 + \sum_{\forall X \cap H} \left(CC(X) * P_{Gen|CC(X)} * P_{Exp|CC(X)} \right) \quad (5)$$

$$P_{Gen|CC(X)} = \frac{\sum_{D=0}^{D<AvgHUSTDepth} P^{D}_{Avg|CC(X)}}{AvgHUSTDepth} \quad (6)$$

$$P_{Avg|CC(X)} = \frac{\sum_{\forall Y} \left(CC(Y) * P_{Exp|CC(Y)} \right)}{\sum_{\forall Y} CC(Y)} \quad (7)$$

$$P_{Exp|CC(X)} = \frac{NumberExpandingHeuristics \in X}{NumberOfHeuristics \in H} \quad (8)$$

Equation (5) shows how for each subset we need to find those CCs which could contribute nodes to the Search Tree. The main difference between equations (5) and (1) is that, instead of looping through each node in the HUST, we only need to loop through the relevant CCs. The number of populated CCs for large trees is orders of magnitude smaller than the number of generated nodes in the HUST.

Equation (6) calculates the probabilities of each node associated to the relevant CC being generated. Equation (6) distributes the nodes associated to the relevant CC equally between all depths from 1 to the average HUST's depth (4). The CC's do not tell us where each node is located, so we assume all associated nodes have an equal chance of being at any given depth.

Equation (7) is the average probability for an ancestor node to be generated, given that the nodes are associated to a specific CC. There was no need to calculate average ancestor expansion probabilities when using equation (1), as it required storing every path in the HUST. Equation (7) takes advantage that the heuristics in the set are consistent to narrow down which CCs could themselves be ancestors of nodes associated with the current CC, e.g. if the

current CC is $CC(111)$, that is all three heuristics expanding, then the only valid ancestors would be nodes associated with $CC(111)$ themselves. Otherwise, heuristic consistency would not hold.

Finally, equation (8), translates the CC characteristic set into the probability of nodes associated to the CC expanding, given the current heuristic subset. Depending on which heuristics are active for the current subset, some of the CC's characteristic set bits may not apply. Hence, we have to recalculate for each heuristic subset the expansion probability for each CC.

Following we discuss, as an example, the model's accuracy when calculating the number of generated nodes when all three heuristics in Fig. 1 are active. The overall estimated number of generated nodes according to the proposed CC prediction model (5) is 1+2+2.11+1.16+0.46=6.73. The more accurate Prediction Model (1) estimates the number of generated nodes as 6.78. The accuracy ratio of the proposed model for this example is $\frac{6.73}{6.78} = 0.99$. As the objective of RIDA* is to choose good heuristic subsets at a reasonable cost, the proposed prediction model is better as it does not require storing node location information.

Simplified CC Prediction Model

We had to simplify the proposed prediction model, by changing the ancestor expansion probability equation (7), due to its high computational cost. Equation 7 takes advantage of heuristic consistency by ensuring that we only include as possible parent CCs those CCs whose "pruning heuristics" are not "expanding heuristics" for the current CC. Otherwise, heuristic consistency would not hold. The problem is that for each heuristic subset considered we already need to loop through each CC to check if it applies to the current subset. Calculating the average ancestor expansion probability, maintaining consistency as in (7), would require doing an additional loop, for each heuristic subset and CC pair, through all other CCs to find the CCs that are consistent with the current CC. It increases the computational complexity by an order of N^2, N being the number of populated CCs for the current iteration. For large heuristic sets the number of CCs can be in the order of tens of thousands or more, so using equation (7) is not practical for efficient online calculations.

There are two alternatives to (7), the first one would be to simply ignore consistency. This would mean that all CCs have a chance, measured by their relative size in associated nodes, to be associated to any ancestor nodes. This is equivalent to assuming no correlation between the heuristics expanding or pruning specific paths which is simply not true. We tried this model but the accuracy dropped significantly compared to the following alternative:

Instead of ignoring consistency we decided to simplify the consistency requirement (and add another error factor) by assuming that the ancestors of each node associated to a CC are themselves associated to the same CC. For most paths in large search trees, the chances are that the parent's node associated CC is the same or changes gradually. So the simplified version of equation (7) is equation (9).

$$P_{Avg|CC(X)} = P_{Exp|CC(X)} \tag{9}$$

Changing P_{Avg} for our example results in no changes for $C(111)$, $P_{Avg} = \frac{2}{3}$ instead of $P_{Avg} = \frac{14}{18}$ for $C(011)$, $P_{Avg} = \frac{1}{3}$ instead of $P_{Avg} = \frac{1}{2}$ for $C(100)$, $P_{Avg} = \frac{1}{3}$ instead of $P_{Avg} = \frac{16}{24}$ for $C(010)$. This results in the predicted amount of nodes to be 1+2+1.87+0.96+0.39=6.22 nodes when using the simplified CC prediction model instead of 6.73 when using the full consistency CC's prediction model. The correct estimation (1) is 6.78 nodes, so the accuracy ratio drops from 0.99 to $\frac{6.22}{6.78} = 0.91$.

Simplifying the CC's prediction model as in equation (9) results in adding a systematic error which lowers the estimated number of generated nodes for most subsets. But it is much cheaper to implement. The priority for the prediction model is to find good heuristic subset rankings as cheaply as possible.

Experiments

The main objective of this paper is to efficiently automate heuristic selection when using randomization. In order to do this we need to answer the following two questions: Is it always faster to use randomization for the complete heuristic set as suggested in (Zahavi et al. 2007), or can it be faster to randomize a problem-specific subset? If it is, can we modify RIDA* so it can efficiently choose good subsets for those problems in which randomization of the whole set is not the best option? For our experiments we used PDB-based heuristics as they are well suited to our needs. Explanation follows.

A significant number of current approaches use as heuristics a type of look-up databases known as Pattern Databases (*PDB*s)(Culberson and Schaeffer 1994). PDBs simplify the problem by selecting only a part of the original problem description (pattern) and projecting it onto an abstraction. Domain-specific PDBs can be calculated only once and stored as a databases which are then re-usable for as many problems as needed, as long as they share the same goal state.

However, given a fixed memory limit there is no domain-independent technique to efficiently select the best patterns for each problem. Domain-independent pattern selection is a variant of the bin-packing problem and it is NP-complete(Edelkamp and Schroedl 2011). Even for domain-specific approaches, the best patterns are problem specific (Korf and Felner 2002)(Holte et al. 2006)(Edelkamp and Schroedl 2011).

Fifteen Puzzle

Table 2 summarizes the results for the Fifteen Puzzle when using the same set of five PDB-based heuristics as in (Franco

and Barley 2009) for a suite of a thousand random problems. Table 2 is divided in two parts. This is done to separate the actual performance of the selected heuristic subsets from RIDA*'s sampling and modeling costs. A ratio less than one means that RIDA* is faster than using the whole set.

Note that the heuristics in the set were created manually to complement each other, i.e. each has patterns that are significantly better in different areas of the search space. Hence it is difficult for RIDA* to find subsets which will do significantly better. The reason we used this heuristic set is because, as we aim to create a method which can be used to automate heuristic selection, we need to make sure our method is robust. If the heuristic set is already well chosen for the current problem instances then we would like our method to not significantly hamper performance, compared to simply using the given heuristic set. We show in the 24 puzzle the savings that can be achieved when the heuristic set is amenable, i.e. the heuristic themselves were not manually designed to complement each other.

Last Iteration (No Sampling & Modeling Costs)				
Average $\frac{SubsetTime}{SetTime}$	StDev	Max	Min	Overall $\frac{\sum SubsetTime}{\sum SetTime}$
0.94	0.18	2.95	0.39	0.96
Total RIDA* Running Time				
1.54	1.42	17	0.57	0.997

Table 2: FIFTEEN PUZZLE RESULTS

Table 2's first half (Last Iteration) shows that RIDA*'s selected heuristics are only slightly faster than using the whole set. The more important result for us is in the second half: after including RIDA*'s meta-reasoning costs (sampling & modeling), the overall performance did not drop significantly. By overall we mean the time it takes to solve all the problems in the suite. The "Average" column refers to the average speed-up ratio per problem. The reason average performance drops significantly is that RIDA*'s meta-reasoning costs cap is empirically determined on a domain basis. This means that the smallest problems in the suite, approximately 10%, were solved while RIDA* was sampling all possible heuristic combinations. Hence, the costs of solving the smallest problems can be significantly higher than using the worst possible heuristic combination. But, as this only affects the smallest problems, the overall results are not significantly affected. A dynamic sampling cap for RIDA* is future work(Franco and Barley 2009).

Twenty-Four Puzzle

For this domain we decided to automatically generate sets of domain-competitive heuristics. The main difference with manual selection is that we did not take into account how well each generated heuristic would complement the other heuristics. This makes the heuristic set more amenable to our approach.

(Holte et al. 2006) showed how a combination of PDB-based heuristics can do better than the state-of-the-art single PDB-based heuristic. They manually designed the PDB-based heuristics (8 combinations of 5-5-5-5-4 disjoint patterns) so that they would complement each other. All patterns in each PDB followed the neighboring rule, i.e. to make competitive PDBs choose neighboring objects in the goal description. We decided to use the same neighboring rule, but to generate a larger set of heuristics. We did not try to make the generated heuristics complement each other, RIDA* should find the best heuristic subset *in situ* for each problem. The created PDB-based heuristics are pseudo-random and domain-competitive. We used three different sets to monitor the effects of increasing the number of heuristics in the set. Each set is respectively made up of the first twenty-five, fifty and hundred heuristics we generated.

Note that the maximum combination degree (*M.C.D.*), i.e. the maximum number of concurrently active heuristics for a candidate subset, was capped for this domain's heuristic sets. The meta-reasoning costs for the randomization model are much bigger for these larger sets (2^N combinations, N being the size of he initial set of heuristics.). The M.C.D was set to five heuristics for the 25 heuristic set, four for the 50 set and three for the 100 set. Also we limited the number of candidate subsets for each combination degree to one thousand heuristic subsets. Each of the candidate subsets was generated randomly with the only condition being to reject duplicate subsets. These compromises were made to keep modeling costs under control. The M.C.D for each heuristic set was determined on a domain basis by doing a few *a priori* test runs.

Finally we only did nine random problems for this set. The reason is that they take a long time (a month approximately) to be solved when using slowest set-up (combining the whole heuristic set instead of using RIDA*). In order to solve the problems in a timely manner we stopped the search as soon as the first solution was found. In order to eliminate the stochastic effects of goal placement biasing our small problem suite, we eliminated the last iteration from the data. We counted all the nodes (or time) up to the penultimate iteration. This mode of comparison does not benefit our model as the more iterations in the solving phase, the smaller RIDA*'s relative sampling and modeling costs.

Table 3 is divided into two parts. In the first part we report RIDA*'s performance, in terms of the number of generated nodes, compared to using randomization for the whole set. In the second part the comparison is made in terms of overall running time, including RIDA*'s modeling and sampling costs.

In this domain we achieved our best results, both node-wise and time-wise. First we discuss the results node-wise: The first column in table 3 is the average node-reduction ratio. The smaller it is, the better RIDA* did. RIDA* generated significantly less nodes than using the whole set, e.g. RIDA*'s generated, on average, less than a third of the nodes compared to randomizing the whole set for the 100 heuristic set. The overall ratio, this is the sum of generated nodes for all problems, was also at its best (0.27) for the hundred heuristic set. The results are also good for the other two sets.

Set	Nodes				
	AvgRatio	*StdDev*	*Max*	*Min*	*Overall Ratio*
	$\frac{SubSetNodes}{SetNodes}$				$\frac{\sum SubsetNodes}{\sum SetNodes}$
25	0.39	0.096	0.51	0.21	0.30
50	0.33	0.10	0.48	0.14	0.32
100	0.30	0.12	0.47	0.098	0.27
Set	**Time, including sampling and modeling costs.**				
	$\frac{SubSetTime}{SetTime}$				$\frac{\sum SubsetTime}{\sum SetTime}$
25	0.36	0.11	0.52	0.18	0.26
50	0.28	0.11	0.42	0.11	0.24
100	0.37	0.26	0.87	0.060	0.20

Table 3: 24 PUZZLE RESULTS

Finally it is also interesting to note that for the larger 100 heuristic set, we had both our best and worst results with any problem instance. The worse ratio (0.87) was due to the problem being one of the smallest but with very high modeling costs. The best ratio (0.060) was due to a node-reduction ratio of 0.098 which was "improved" by the bigger overhead of the whole set with respect to the selected subset.

Towers of Hanoi

The Towers of Hanoi problem is a classic search problem. The classic version consists of three pegs and a set of disks. All the disks have different sizes and can only be stacked on top of larger disks. The Towers of Hanoi problem is to move all the disks from one peg to another peg. Only the disk on the top of a peg can be moved, but it can be moved to any other peg as long as it is either free or contains a larger disk on top. We used four pegs because there is no known deterministic algorithm which guarantees an optimal solution (Felner, Korf, and Hanan 2004).

The heuristic set was created using the same pattern database as in (Felner, Korf, and Hanan 2004). Their heuristic generation approach is to calculate the largest pattern which can be fitted in memory. Then the domain symmetries can be used to generate large sets of domain-competitive heuristics (Felner, Korf, and Hanan 2004). This creates an interesting question which RIDA* is designed to address: which heuristics do we select? For this paper the question is, can the proposed model pick a heuristic subset that will do better than using the whole set with randomization?

We used for this domain the same basic RIDA* selection architecture as in the sliding-puzzle domains, i.e. first using IDA* and HUST/CCs to sample the initial iterations until all heuristics generated more nodes than the sampling cap, then using the proposed randomization prediction model to choose a subset. As in the Twenty-four puzzle we capped the M.C.D to five heuristics and also limited the number of candidates to a maximum of a thousand randomly generated but unique heuristic subsets per combination degree. The first change was to use A* instead of IDA* to solve the problem. There are too many alternative paths leading to the same state for IDA* to be competitive. The second change was to use a global duplicate check mechanism on the sampling phase, so that we predict more accurately the number of generated nodes, up to an F-bound, when using A*.

Whole Set	RIDA*'s Selected Subset
1.61×10^7	1.46×10^7

Table 4: Generated Nodes for A*

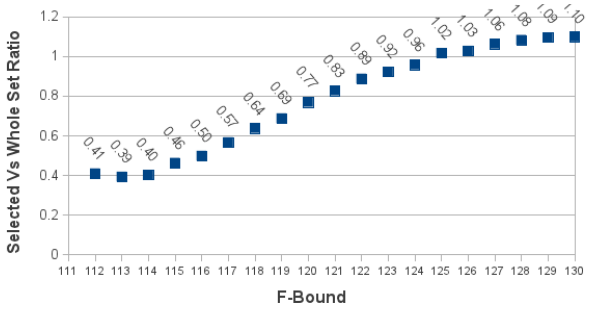


Figure 2: $\frac{GeneratedNodesSubset}{GeneratedNodesSet}$ for IDA* with state duplicate check

Fig. 2 and Table 4 shows the results, node-wise, when solving a 15 Disks-4 Pegs Tower of Hanoi problem using a 13 Disk PDB. The 13 Disk PDB was used to calculate a set of $\frac{15*14}{2} = 105$ heuristics (Felner, Korf, and Hanan 2004).

Table 4 also shows that the selected subset generates almost the same number of nodes compared to simply randomizing the whole set. This is not that surprising when we consider that, due to the domain symmetries, each heuristic in the set complements each other perfectly when the goal is moving all disks to a different peg (Felner, Korf, and Hanan 2004). Each heuristic is a PDB, made of two additive disjoint groups of thirteen disks and two disks, which is less informed when accounting for the two disks in the smaller two disk group than for the remaining thirteen disks in the larger group.

This domain shows an interesting *caveat* of the proposed model. RIDA* makes the assumption that we are doing enough sampling to have estimated the correct asymptotic Heuristic Branching Factor (HBF) for each heuristic subset. This works well for the heuristics in the sliding-puzzle, for which the different heuristic subset's asymptotic HBF changes very little once a few iterations have been sampled. This is not the case for the Towers of Hanoi. Lets take two Towers of Hanoi heuristics, h_1&h_2 created from the same PDB. h_1 is more accurate regarding the two bottom disks and h_2 is more accurate regarding the two top disks. h_1 will give higher h-values on average, compared to h_2, when close to the initial state. But this heuristic ranking reverses when close to the goal state, then the two bottom disks tend to be in their goal positions for more states. The proposed randomization model chooses a heuristic subset based on the sampled initial iterations, and hence chooses subsets which are more informed in the neighborhood of the initial state.

Fig. 2 shows how the chosen subset performs on consecutive IDA* iterations. We used IDA* with state duplicate check, instead of A*, for RIDA*'s solving phase in Fig. 2 to show how RIDA*'s chosen subset's performance decreases as the search space gets closer to the goal state.

Conclusions

This paper's main contribution is to automate heuristic selection in the context of randomization. Efficiently finding the right abstraction-based heuristics for each problem is considered too onerous, so a less efficient human-led design is done on a domain basis. To support our claim we first show that Diminishing Returns is still a problem when combining heuristics via random selection, i.e. given a set of domain competitive heuristics, choosing the right subset of heuristics can do significantly better than simply picking a heuristic at random from the whole set. Secondly, we present a new analytical performance prediction model to efficiently select problem specific heuristic subsets which do better than simply randomizing the whole heuristic set. By efficiently we mean that, given that there is a subset which can do significantly better than simply randomizing the whole set, the costs associated with finding that subset do not eliminate the resulting performance improvement. We used the proposed prediction model to modify an existing algorithm, called RIDA*, to test how well it did.

We tried the proposed prediction model for three domains. The actual speed-up is dependent on how complementary the heuristics in the input set are. Hence it is important for our proposed model to be robust, this is to not significantly hamper performance when the available heuristic set is close to optimal for the problems being solved. This was the case for the Fifteen puzzle. We showed our methodology resulted in no significant loss of performance when the heuristic set is not amenable. For the Twenty-Four Puzzle we automated heuristic generation. These sets were made of domain-competitive heuristics not specifically designed to complement each other. Thanks to this, RIDA* was able to find problem-specific heuristic subsets which were significantly faster than simply randomizing the whole set.

Perhaps the major caveat of the proposed model is the assumption that if enough initial sampling is done, then there is no need for any further sampling. This assumption is not true for the heuristics used in the Towers of Hanoi domain. This leads the proposed prediction model to estimate relatively large savings based solely on the data gathered in a few initial iterations. Our experiments showed this assumption was a bad assumption for the Towers of Hanoi domain but a good one for the slide puzzles. A monitoring mechanism to verify/update the model predictions is future work.

Acknowledgments

This material based on research sponsored by the Air Force Research Laboratory, under agreement number FA2386-12-1-4018. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Research Laboratory or the U.S. Government.

References

Culberson, J., and Schaeffer, J. 1994. Efficiently searching the 15-puzzle. Technical report.

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

Edelkamp, S., and Schroedl, S. 2011. *Heuristic Search: Theory and Applications*. Morgan Kaufmann. Elsevier Science.

Felner, A.; Korf, R. E.; and Hanan, S. 2004. Additive pattern database heuristics. *J. Artif. Intell. Res. (JAIR)* 22:279–318.

Franco, S., and Barley, M. 2009. Predicting the optimal combination of pattern databases for solving a problem. *International Symposium on Combinatorial Search*.

Haslum, P.; Botea, A.; Helmert, M.; Bonet, B.; and Koenig, S. 2007. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *Proceedings of the National Conference on Artificial Intelligence*, volume 22-2, 1007. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999.

Helmert, M.; ludwigs-universitt Freiburg, A.; Haslum, P.; and Hoffmann, J. 2007. Flexible abstraction heuristics for optimal sequential planning. In *In Proc. ICAPS 2007*, 176–183.

Holte, R.; Felner, A.; Newton, J.; Meshulam, R.; and Furcy, D. 2006. Maximizing over multiple pattern databases speeds up heuristic search. *Artificial Intelligence* 170(16-17):1123–1136.

Jabbari Arfaee, S.; Zilles, S.; and Holte, R. C. 2010. Bootstrap learning of heuristic functions. In *Third Annual Symposium on Combinatorial Search*.

Korf, R. E., and Felner, A. 2002. Disjoint pattern database heuristics. *J. Artif. Intell. Res. (JAIR)* 134:9–22.

Korf, R. E.; Reid, M.; and Edelkamp, S. 2001. Time complexity of iterative-deepening-a*. *Artif. Intell.* 129(1-2):199–218.

Valenzano, R.; Sturtevant, N.; Schaeffer, J.; and Buro, K. 2010. Simultaneously searching with multiple settings: An alternative to parameter tuning for suboptimal single-agent search algorithms. In *In Proc. ICAPS*, 177–184.

Zahavi, U.; Felner, A.; Schaeffer, J.; and Sturtevant, N. R. 2007. Inconsistent heuristics. In *AAAI*, 1211–1216.

Zahavi, U.; Felner, A.; Burch, N.; and Holte, R. C. 2008. Predicting the performance of ida* with conditional distributions. In Fox, D., and Gomes, C. P., eds., *AAAI*, 381–386. AAAI Press.

Catching Label Subsets for Relaxed Bisimulation: An Abstraction Refinement Approach

Marcel Steinmetz
Saarland University
Saarbrücken, Germany
s9mrstei@stud.uni-saarland.de

Jörg Hoffmann and **Michael Katz**
Saarland University
Saarbrücken, Germany
{hoffmann, katz}@cs.uni-saarland.de

Abstract

Merge-and-Shrink abstraction (M&S) is an approach for constructing admissible heuristic functions. A key issue in M&S abstractions is which states are mapped to the same abstract state: That decision directly controls the trade-off between the accuracy of the resulting heuristic function on the one hand, and the size of the abstract state space on the other hand. A recent approach towards tackling this issue introduced the notion of K*-catching bisimulation*. This class of abstractions is a bisimulation – preserving transition behavior exactly – but only for a subset K of the planning operators, ignoring all others. It has been shown that this form of relaxed bisimulation is invariant over the M&S process, and that choosing K appropriately may reduce abstraction size exponentially while still delivering a perfect heuristic. Determining those exact operator subsets K is, however, highly intractable, and practical approximations so far did not yield convincing results. Thus the question remains: How to select K?

We propose to answer that question by a counter-example guided abstraction refinement (CEGAR) approach. Given a K-catching bisimulation, we analyze its optimal solutions and identify new operators to be added to K, ultimately excluding all spurious solutions. We design, and experiment with, practical criteria to terminate the refinement process before that happens.

Introduction

One of the currently most successful approaches to solve problems in optimal planning is to use the A^* search algorithm with an admissible heuristic function. Heuristic functions estimate the cost of the cheapest operator sequence that, when applied on the initial state, leads to the goal. In addition, admissible heuristic functions give the guarantee that this estimation is always a lower bound on the cheapest cost of real solutions, and the better this estimation is, i.e., the smaller the difference to this cost, the faster will the search algorithm find a solution for the problem. Therefore, the main question becomes how good admissible heuristic functions can be computed automatically, for any given problem.

One approach to construct an admissible heuristic function is based on abstractions. Abstractions ignore the difference between certain states and consequently reduce the total size of the state space, while preserving all operator sequences that are possible in the concrete state space. With fewer states, the analysis of this abstract state space can become feasible so that the heuristic value for some state can be computed by computing the cost of the optimal solution of its representative in the abstract version of the state space.

Currently, two methods for building an abstraction are used in planning: Pattern databases (Haslum et al. 2007) and Merge-and-Shrink abstractions (Helmert, Haslum, and Hoffmann 2007), and the latter one, Merge-and-Shrink abstraction, strictly generalizes pattern databases. M&S builds an abstraction by iteratively combining, called *merging*, and further reducing the size of, called shrinking, basic parts of the state space. To preserve the entire behavior of the original state space in the abstract state space, Nissim et al. (2011) used the well-known notion of bisimulation (Milner 1990). They observe that the bisimulation requirement is unnecessarily strict for the purpose of computing heuristic functions. This has been addressed by two different relaxations of bisimulation: greedy bisimulation (Nissim, Hoffmann, and Helmert 2011), and K-catching bisimulation (Katz, Hoffmann, and Helmert 2012). Both are based on the same idea, ignoring the difference between more states by considering only a subset of transitions during the test for the bisimulation property. Greedy bisimulation ignores transitions based on a local condition, i.e., on a per-transition basis (abstract goal distances at the transition's end points). By contrast, K-catching bisimulation fixes a global criterion throughout the M&S process, simply catching a transition if its label – the planning operator inducing it – is contained in a label subset K fixed a priori. The key advantage of that criterion is its invariance over the M&S construction (**?**), providing direct control over the final abstraction in terms of the choice of K.

Katz et al. identify two different types of operators that, if caught by K, lead to a perfect heuristic, while the resulting K-catching bisimulation can be exponentially smaller than any general bisimulation. However, computing these sets K exactly is highly intractable: They consist of all operators that form part of an optimal solution for any state in (part of) the state space. Katz et al. devise some simple approximation methods, with mediocre empirical results. Herein, we instead explore the possibility to design K via *counterexample-guided abstraction refinement (CEGAR)*.

CEGAR was originally introduced in the context of model checking (Clarke et al. 2003) for the purpose of proving (un)reachability of states in transition systems. It computes an abstraction of a transition system by an incremental procedure that analyzes intermediate abstractions to extract information – counterexamples – that can be used to improve the abstraction. One starts with a simple initial abstraction of the transition system that is improved by incrementally distinguishing more states, in the so called *refinement loop*, as long as certain properties are not fulfilled. CEGAR consists of the three general components:

(1) *Initial abstraction.* Specification of the abstraction that is used to start the refinement. (In our case: K-catching bisimulation for the empty set K.)

(2) *Analysis of the abstract transition system.* Determining unintended behavior of the abstract transition system of the current abstraction. (In our case: Optimal abstract solutions that are spurious, failing to solve the original planning task.)

(3) *Refine the abstraction.* Computing a new abstraction, by using the information obtained by (2), that excludes the unintended behavior. (In our case: Including new operators into K.)

(2) together with (3) are called the refinement step and they are executed as long as the abstract transition system does not fulfill some specific criteria. In the following we show how (1), (2) and (3) are instantiated for the purpose of building a K-catching bisimulation.

Background

A **planning task** is a 5-tuple $\Pi = (\mathcal{V}, \mathcal{O}, c, s_0, s_*)$. $\mathcal{V}$ is a finite set of **variables** v, each $v \in \mathcal{V}$ associated with a finite domain $\mathcal{D}_v$. A **partial state** over $\mathcal{V}$ is a function s on a subset $\mathcal{V}_s$ of $\mathcal{V}$, so that $s(v) \in \mathcal{D}_v$ for all $v \in \mathcal{V}_s$; s is a **state** if $\mathcal{V}_s = \mathcal{V}$. The **initial state** s_0 is a state. The **goal** s_* is a partial state. $\mathcal{O}$ is a finite set of **operators**, each being a pair $(\mathit{pre}, \mathit{eff})$ of partial states, called its **precondition** and **effect**. Each $o \in \mathcal{O}$ is also associated with its **cost** $c(o) \in \mathbb{R}_0^+$.

The **state space** of a planning task is given by a **transition system**. Such a system is a 6-tuple $\Theta = (S, L, c, T, s_0, S_*)$ where S is a finite set of **states**, L is a finite set of **transition labels**, each associated with a **label cost** $c(l) \in \mathbb{R}_0^+$, $T \subseteq S \times L \times S$ is a set of **transitions**, $s_0 \in S$ is the **start state**, and $S_* \subseteq S$ is the set of **goal states**. We define the **remaining cost** $h^* : S \to \mathbb{R}_0^+$ as the minimal cost of any path (the sum of costs of the labels on the path), in Θ, from a given state s to any $s_* \in S_*$, or $h^*(s) = \infty$ if there is no such path.

In the state space of a planning task, S is the set of all states. The start state s_0 is the initial state of the task, and $s \in S_*$ if $s_* \subseteq s$. The transition labels L are the operators $\mathcal{O}$, and $(s, (\mathit{pre}, \mathit{eff}), s') \in T$ if s complies with pre, and $s'(v) = \mathit{eff}(v)$ for all $v \in \mathcal{V}_{\mathit{eff}}$ while $s'(v) = s(v)$ for all $v \in \mathcal{V} \setminus \mathcal{V}_{\mathit{eff}}$. The solution of a planning task, called **plan**, is a path from s_0 to any $s_* \in S_*$. The plan is **optimal** if its summed-up cost is equal to $h^*(s_0)$.

A **heuristic** is a function $h : S \to \mathbb{R}_0^+ \cup \{\infty\}$. The heuristic is **admissible** if, for every $s \in S$, $h(s) \leq h^*(s)$; it is **consistent** if, for every $(s, l, s') \in T$, $h(s) \leq h(s') + c(l)$; it is **perfect** if h coincides with h^*. The A^* algorithm expands states by increasing value of $g(s) + h(s)$ where $g(s)$ is the accumulated cost on the path to s. If h is admissible, then A^* returns an optimal solution. If h is consistent then A^* does not need to re-open any nodes.

One way of automatically constructing admissible heuristics is based on **abstractions**. This is a function α mapping S to a set of **abstract states** S^α. The **abstract state space** Θ^α is defined as $(S^\alpha, L, c, T^\alpha, s_0^\alpha, S_*^\alpha)$, where $T^\alpha := \{(\alpha(s), l, \alpha(s')) \mid (s, l, s') \in T\}$, $s_0^\alpha := \alpha(s_0)$, and $S_*^\alpha := \{\alpha(s_*) \mid s_* \in S_*\}$. The **abstraction heuristic** h^α maps each $s \in S$ to the remaining cost of $\alpha(s)$ in Θ^α; h^α is admissible and consistent. The **pre-image** of $s^\alpha \in S^\alpha$ under the abstraction α is the set $\mathit{Pre}_\alpha(s^\alpha) = \{s \in S \mid \alpha(s) = s^\alpha\}$. We will sometimes consider the **induced equivalence relation** $\sim^\alpha$, defined by setting $s \sim^\alpha t$ iff $\alpha(s) = \alpha(t)$.

How to construct a good α in general? Helmert et al. (2007) propose M&S abstraction as a method allowing fine-grained abstraction design, selecting individual pairs of (abstract) states to aggregate. The approach builds the abstraction in an incremental fashion, iterating between *merging* and *shrinking* steps. In detail, an abstraction α is an **M&S abstraction over** $V \subseteq \mathcal{V}$ if it can be constructed using these rules:

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

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

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

Rule (i) allows to start from **atomic projections**. These are simple abstractions $\pi_{\{v\}}$ (also written π_v) mapping each state $s \in S$ to the value of one selected variable v. **Rule (ii)**, the **shrinking step**, allows to iteratively aggregate an arbitrary number of state pairs, in abstraction β. Formally, this simply means to apply an additional abstraction γ to the image of β. In **rule (iii)**, the **merging step**, the merged abstraction $\alpha_1 \otimes \alpha_2$ is defined by $(\alpha_1 \otimes \alpha_2)(s) := (\alpha_1(s), \alpha_2(s))$.

To implement M&S in practice, we need a **merging strategy** deciding which abstractions to merge in (iii), and a **shrinking strategy** deciding which (and how many) states to aggregate in (ii). Throughout this paper, we use the same merging strategy as presented by Nissim et al. (2011). To obtain an abstraction that preserves the behavior of the original transition system, i.e., to obtain a perfect heuristic, Nissim et al. devise a shrinking strategy that computes a *bisimulation* of the state space.

Definition 1 *Let $\Theta = (S, L, c, T, s_0, S_*)$ be a transition system. An equivalence relation $\sim$ on S is a **bisimulation** for Θ if for every $s \sim t$ holds: (1) either $s, t \in S_*$ or $s, t \notin S_*$; (2) for every transition label $l \in L$, $\{[s'] \mid (s, l, s') \in T\} = \{[t'] \mid (t, l, t') \in T\}$.*

As usual, $[s]$ for a state s denotes the equivalence class of s. An abstraction α is a bisimulation iff the induced equivalence relation $\sim^\alpha$ is. Nissim et al. have shown that the bisimulation property is preserved throughout the merging steps:

If α_1 and α_2 are bisimulations for $\Theta^{\pi_{V_1}}$ and $\Theta^{\pi_{V_2}}$, where $V_1 \cap V_2 = \emptyset$, then $\alpha_1 \otimes \alpha_2$ is a bisimulation for $\Theta^{\pi_{V_1 \cup V_2}}$. Therefore, if α is constructed such that, in every application of (ii), γ is a bisimulation for Θ^{β}, then the overall M&S abstraction α will be a bisimulation for Θ^{π_V} (Nissim, Hoffmann, and Helmert 2011).

Unfortunately, bisimulations are exponentially big even in trivial examples. Katz et al. (2012) address this by the notion of *K-catching bisimulation*. This relaxes the definition of bisimulation by applying constraint (2) of Definition 1 to only a subset of the transitions in T, selected by a subset of transition labels K:

Definition 2 *Let $\Theta = (S, L, c, T, s_0, S_*)$ be a transition system and $K \subseteq L$. An equivalence relation $\sim$ on S is a **K-catching bisimulation** for Θ if it is a bisimulation for the transition system (S, K, c, T^K, s_0, S_*), where $T^K = \{(s, l, t) \mid (s, l, t) \in T, l \in K\}$.*

Katz et al. (2012) identify two theoretical classes of such label subsets that, when considered in a K-catching bisimulation, lead to abstract transition systems that still provide strong properties. First, if the label subset contains (at least) each label that is used in some optimal path from any state to a goal state, then the resulting abstraction heuristic will still be perfect, while possibly obtaining an abstract transition system that is exponentially smaller than the abstract transition system of any general bisimulation. Such labels are called globally relevant, i.e., a label $l \in L$ is called **globally relevant** if there is a transition $(s, l, s') \in T$ such that $h^*(s) = h^*(s') + c(l)$.

Second, K does not even have to contain all globally relevant labels. If K contains each label that is used in some optimal solution for any state *with cost less or equal than the remaining cost of the initial state*, then A^*, using the abstraction heuristic, will still only expand a number of states linear in the length of the plan returned (under the assumption that the planning task does not contain 0-cost operators). Such labels are called $h^*(s_0)$-relevant, i.e., a label $l \in L$ is called R**-relevant** if there is a transition $(s, l, s') \in T$ such that $h^*(s) \leq R$ and $h^*(s) = h^*(s') + c(l)$.

Unfortunately, computing either of these label subsets K involves solving the problem in first place. We now introduce a method to select K based on counterexample guided abstraction refinement.

Abstraction Refinement

The general algorithm is depicted in Figure 1. The whole refinement procedure is based on a *current label subset*. This label subset is initially empty. For the purpose of refining the K-catching bisimulation, we add new labels to this label subset and recompute the K-catching bisimulation with the updated label set. To catch the right labels, we compare the optimal solutions of the abstract transition system with solutions of the original transition system. If these abstract solutions are spurious, i.e., they do not correspond to solutions of the original transition system, then we extract labels to be added to the label subset, which will eliminate these solutions from the abstract transition system.

$K \leftarrow \emptyset$
$\alpha \leftarrow K$-catching bisimulation for Θ
$\Theta^\alpha \leftarrow$ Abstract transition system of α
while the given criterion is not satisfied **do**
  $K' \leftarrow$ *analyze*(Π, Θ^α)
  $K \leftarrow K \cup K'$
  $\alpha \leftarrow K$-catching bisimulation for Θ
  $\Theta^\alpha \leftarrow$ Abstract transition system of α
endwhile
return α

Figure 1: General abstraction refinement procedure for computing K-catching bisimulations, based on a given termination criterion.

This step is repeated until the abstract transition system fulfills certain constraints. As an example, one could require that every optimal solution of the abstract transition system is a solution of the original transition system.

In the following sections we show how the abstract transition system is analyzed, that is how spurious solutions are identified, and which labels have to be considered to exclude these paths from the abstract transition system.

Refinement Step

We distinguish between *forward propagation*, where optimal solutions of the abstract transition system are compared to paths in the original transitions system, and *backward propagation*, where backward optimal solutions, i.e. the cheapest paths from some abstract goal state to the abstract initial state, obtained by inverting all transitions of the abstract transition system, are considered.

Depending on the termination criterion, the abstract solutions should only partially correspond to paths of the original transition system. Therefore, the distinction between these two cases allows to determine whether the abstraction should be more precise around the initial state, or more precise around the goal states, i.e., whether the labels added to the label subset should be closer to the initial state, or closer to the goal states.

Forward Propagation

Forward propagation analyzes the abstract transition system by finding, for each optimal solution, an equivalent path in the original transition system that starts in the initial state and which is labeled with the same label sequence. Therefore, we say that a label sequence is *forward-applicable* in a state s if there is a path in the transition system that starts in s and contains the given labels. Formally it is defined as follows:

Definition 3 *Let $\Theta = (S, L, c, T, s_0, S_*)$ be a transition system, $\vec{a} = \langle a_1, \ldots, a_n \rangle \in L^n$ be a label sequence. Then $\vec{a}$ is **forward-applicable in** $t_0 \in S$ if there are states $t_1, \ldots, t_n \in S$ so that $(t_{i-1}, a_i, t_i) \in T$, for all $1 \leq i \leq n$. Moreover, t_n is called the **end-state**. If it further holds that for all $t'_i \in S \setminus \{t_i\}$, $(t_{i-1}, a_i, t'_i) \notin T$, for every $1 \leq i \leq n$, then $\vec{a}$ is called **deterministic forward-applicable in** t_0.*

A solution of the abstract transition system $s^\alpha_0, a_1, s^\alpha_1, \ldots, a_n, s^\alpha_n$ is called **spurious**, if its label

sequence $\langle a_1, \ldots, a_n \rangle$ is not forward-applicable in s_0, or if it is forward-applicable in s_0, leading to the path $s_0, a_1, s_1, \ldots, a_n, s_n$, but s_n is not a goal state, $s_n \notin S_*$.

We distinguish between three different classes of spurious solutions in order to find the label that must be added to the current label subset to remove the considered path from the abstract transition system.

If the label sequence of a spurious solution is deterministic forward-applicable in the abstract initial state, but it is not forward-applicable in the initial state of the original transition system, then it is enough to add only one label to the label subset in order to remove the considered path from the abstract transition system:

Theorem 1 *Let $\Theta = (S, L, c, T, s_0, S_*)$ be a transition system and let α be a K-catching bisimulation for Θ, with abstract transition system $\Theta^\alpha = (S^\alpha, L, c, T^\alpha, s_0^\alpha, S_*^\alpha)$. Then it holds for any state $t_0 \in S$ and for any action sequence $\langle a_1, \ldots, a_n \rangle \in L^n$ so that $\langle a_1, \ldots, a_{n-1} \rangle$ is deterministic forward-applicable in $\alpha(t_0)$: If $\langle a_1, a_2, \ldots, a_n \rangle$ is forward-applicable in $\alpha(t_0)$, and $\langle a_1, a_2, \ldots, a_{n-1} \rangle$ is forward-applicable in t_0, but $\langle a_1, a_2, \ldots, a_n \rangle$ is not forward-applicable in t_0, then $a_n \notin K$.*

Proof: Assume for contradiction that $a_n \in K$. Because $\langle a_1, a_2, \ldots, a_{n-1} \rangle$ is forward-applicable in t_0 in Θ, there must be states $t_1, \ldots, t_{n-1} \in S$, s.t. $(t_{i-1}, a_i, t_i) \in T$, for all $1 \leq i < n$. Then it follows by the definition of the abstract transition system, for the states $t_0^\alpha = \alpha(t_0)$, $t_i^\alpha := \alpha(t_i)$, that $(t_{i-1}^\alpha, a_i, t_i^\alpha) \in T^\alpha$, for all $1 \leq i < n$. By assumption $\langle a_1, \ldots, a_{n-1} \rangle$ is deterministic forward-applicable in t_0^α, implying that $t_0^\alpha, a_1, \ldots, a_{n-1}, t_{n-1}^\alpha$ is the *only* path in Θ^α with this label sequence and starting in t_0^α. Now, we know that $\langle a_1, \ldots, a_n \rangle$ is forward-applicable in t_0^α. As $t_0^\alpha, a_1, \ldots, a_{n-1}, t_{n-1}^\alpha$ is the only path matching the prefix $\langle a_1, \ldots, a_{n-1} \rangle$, we must have a transition $(t_{n-1}^\alpha, a_n, t_n^\alpha) \in T^\alpha$, for some abstract state $t_n^\alpha \in S^\alpha$. We can conclude by definition of Θ^α that there must be states $t'_{n-1}, t'_n \in S$ with $\alpha(t'_{n-1}) = t_{n-1}^\alpha$, $\alpha(t'_n) = t_n^\alpha$ and $(t'_{n-1}, a_n, t'_n) \in T$. Since α is a K-catching bisimulation for Θ and $a_n \in K$, it follows from (2) of Definition 1 that for every $t \in S$ with $\alpha(t) = t_{n-1}^\alpha$, there is a state $t' \in S$ with $\alpha(t') = t_n^\alpha$ such that $(t, a_n, t') \in T$. So in particular there is a state $t_n \in S$ with $\alpha(t_n) = t_n^\alpha$ such that $(t_{n-1}, a_n, t_n) \in T$. Therefore $\langle a_1, a_2, \ldots, a_n \rangle$ is actually forward-applicable in t_0 in Θ, which contradicts the assumption. It follows that a_n cannot be contained in K. So $a_n \notin K$. ∎

Let $s_0^\alpha, a_1, \ldots, a_m, s_m^\alpha$ be a spurious solution whose label sequence $\langle a_1, \ldots, a_m \rangle$ is not forward-applicable in the initial state of the original transition system. To refine the current label subset based on this path, we find the index $1 \leq n \leq m$ such that the sub-sequence $\langle a_1, \ldots, a_{n-1} \rangle$ is forward-applicable in s_0, but $\langle a_1, \ldots, a_n \rangle$ is not. Under the assumption that $\langle a_1, \ldots, a_{n-1} \rangle$ is deterministic forward-applicable in s_0^α, Theorem 1 implies that adding a_n to the current label subset will either remove the entire path from the abstract transition system, or it will at least destroy the deterministic forward-applicability of $\langle a_1, \ldots, a_{n-1} \rangle$ in s_0^α.

If the label sequence of the spurious path is actually forward-applicable in the initial state of the original transition system, then it can only be spurious if its execution in the original transition system does not end in a goal state.

Theorem 2 *Let $\Theta = (S, L, c, T, s_0, S_*)$ be a transition system and let α be a K-catching bisimulation for Θ, with abstract transition system $\Theta^\alpha = (S^\alpha, L, c, T^\alpha, s_0^\alpha, S_*^\alpha)$. It holds for any state $t_0 \in S$ and for any action sequence $\langle a_1, \ldots, a_n \rangle \in L^n$, such that $\langle a_1, \ldots, a_{n-1} \rangle$ is deterministic forward-applicable in $\alpha(t_0)$: If $\langle a_1, \ldots, a_n \rangle$ is forward-applicable in $\alpha(t_0)$ with end-state $t_n^\alpha \in S_*^\alpha$, and $\langle a_1, a_2, \ldots, a_n \rangle$ is forward-applicable in t_0, but for all possible end-states $t_n \notin S_*$, then $a_n \notin K$.*

Proof: Let $t_1, \ldots, t_n \in S$ be states so that $(t_{i-1}, a_i, t_i) \in T$, for all $1 \leq i \leq n$. Such states must exist since $\langle a_1, a_2, \ldots, a_n \rangle$ is applicable in t_0. By definition of Θ^α follows that $(\alpha(t_{i-1}), a_i, \alpha(t_i)) \in T^\alpha$, for all $1 \leq i \leq n$. By assumption $\langle a_1, \ldots, a_{n-1} \rangle$ is deterministic forward-applicable in $\alpha(t_0)$, implying that $\alpha(t_0), a_1, \ldots, a_{n-1}, \alpha(t_{n-1})$ is the only path in Θ^α with this label sequence and starting in $\alpha(t_0)$. But this means that t_n^α can only be an end-state of a forward-application of $\langle a_1, \ldots, a_n \rangle$ in $\alpha(t_0)$ if there is a transition $(\alpha(t_{n-1}), a_n, t_n^\alpha) \in T^\alpha$. By definition of Θ^α follows that there are states $t'_{n-1}, t'_n \in S$ with $\alpha(t'_{n-1}) = \alpha(t_{n-1})$ and $\alpha(t'_n) = t_n^\alpha$ such that $(t'_{n-1}, a_n, t'_n) \in T$. Rule (1) of Definition 1, together with $t_n^\alpha \in S_*^\alpha$ imply that $t'_n \in S_*$, which by assumption means that $t'_{n-1} \neq t_{n-1}$. Moreover, this rule implies that $\alpha(t_n) \neq t_n^\alpha = \alpha(t'_n)$. Therefore, we have found two states t_{n-1}, t'_{n-1} with $\alpha(t_{n-1}) = \alpha(t'_{n-1})$ such that there is a transition $(t'_{n-1}, a_n, t'_n) \in T$, but for all transition $(t_{n-1}, a_n, t_n) \in T$ holds that $\alpha(t_n) \neq \alpha(t'_n)$. Now, rule (2) of Definition 1 implies that $a_n \notin K$. ∎

Let $s_0^\alpha, a_1, s_1^\alpha \ldots, a_m, s_m^\alpha$ be a spurious solution such that $\langle a_1, \ldots, a_m \rangle$ is forward applicable in s_0, meaning that the execution of this label sequence in s_0 does not end in a goal state. Under the assumption that $\langle a_1, \ldots, a_{m-1} \rangle$ is deterministic forward-applicable in s_0^α, we can conclude with Theorem 2 that adding a_m to the current label subset will either remove the entire path from the abstract transition system, or it will at least destroy the deterministic forward-applicability of $\langle a_1, \ldots, a_{n-1} \rangle$ in s_0^α.

Finally, if the label sequence of a spurious path is not deterministic forward-applicable in the initial state of the abstract transition system, then we remove the non-deterministic choice along the execution of the considered label sequence in the abstract transition system.

Definition 4 *Let $\Theta = (S, L, c, T, s_0, S_*)$ be a transition system and let $l \in L$. Θ is **non-deterministic in** l if there are transitions $(s, l, t) \in T$ and $(s, l, t') \in T$, for some states $s, t, t' \in S$, such that $t \neq t'$.*

If the label sequence of the considered spurious solution is not deterministic forward-applicable in the abstract initial state, then the abstract transition system must be non-deterministic in at least one label, contained in this sequence. If such a label has been identified, then it is enough to add

it to the current label subset, in order to remove the non-determinism from the abstract transition system:

Theorem 3 *Let $\Theta = (S, L, c, T, s_0, S_*)$ be a transition system and let α be a K-catching bisimulation for Θ with abstract transition system $\Theta^\alpha = (S^\alpha, L, c, T^\alpha, s_0^\alpha, S_*^\alpha)$. If Θ^α is non-deterministic in $l \in L$, but Θ is not, then $l \notin K$.*

Proof: Because of Θ^α is non-deterministic in l, there must be states $s^\alpha, t_0^\alpha, t_1^\alpha \in S^\alpha$ with $t_0^\alpha \neq t_1^\alpha$ and $(s^\alpha, l, t_0^\alpha) \in T^\alpha$, $(s^\alpha, l, t_1^\alpha) \in T^\alpha$. By definition of Θ^α follows that there are states $s, s', t_0, t_1 \in S$ with $\alpha(s) = \alpha(s') = s^\alpha$, $\alpha(t_0) = t_0^\alpha$, $\alpha(t_1) = t_1^\alpha$ and $(s, l, t_0) \in T$, $(s', l, t_1) \in T$. By assumption Θ is deterministic in l, so $s = s'$ cannot hold, since $t_0 \neq t_1$. Therefore there are transitions $(s, l, t) \in T$ and $(s', l, t') \in T$ for $s \neq s'$, but $\alpha(s) = \alpha(s')$ and $\alpha(t) \neq \alpha(t')$. This means that $l \notin K$, otherwise α would not be a K-catching bisimulation. ∎

Let $s_0^\alpha, a_1, s_1^\alpha \ldots, a_m, s_m^\alpha$ be a spurious solution such that the label sequence $\langle a_1, \ldots, a_m \rangle$ is not deterministic forward-applicable in s_0^α. We find the first index $1 \leq n \leq m$, so that Θ^α is non-deterministic in a_n, and add this label a_n to the current label subset. Since the state space of a planning task is by definition deterministic, it follows by Theorem 3 that adding a_n to the label subset is sufficient to exclude the non-determinism of a_n from the abstract transition system.

Backward Propagation

Backward propagation analyzes the abstract transition system in a similar way as forward propagation. It considers also the optimal solutions of the abstract transition system, but instead of starting with the initial state and testing for the forward-applicability of the labels, backward propagation starts with the goal states of the original transition system and tries to *regress* these states with the label sequences of the solutions, until the initial state has been reached. Formally, regression is defined as follows:

Definition 5 *Let $\Theta = (S, L, c, T, s_0, S_*)$ be a transition system, $S' \subseteq S$ be a set of states, and $l \in L$ be a label. Then* ***regressing*** *S'* ***with*** *l is defined by $regr(S', l) = \{s \in S \mid (s, l, s') \in T, s' \in S'\}$. For a label sequence $\vec{a} = \langle a_1, \ldots, a_n \rangle \in L^n$, $regr(S', \vec{a}) = regr(regr(\ldots.(regr(S', a_n), \ldots), a_1)$. Moreover, $\vec{a}$ is called* ***reverse-applicable*** *in S' if $regr(S', \vec{a}) \neq \emptyset$.*

For $S' = \{s\}$ we often write $regr(s, l)$ instead of $regr(S', l)$. Then a solution $s_0^\alpha, a_1, s_1^\alpha, \ldots, a_n, s_n^\alpha$ of the abstract transition system is called **spurious** if $s_0 \notin regr(S_*, \langle a_1, \ldots, a_n \rangle)$.

First, observe that if all labels of a label sequence are contained in the label subset, then regressing an abstract state t^α is actually equivalent to regressing the states t of the original transition system, with $\alpha(t) = t^\alpha$:

Lemma 1 *Let $\Theta = (S, L, c, T, s_0, S_*)$ be a transition system, and α be a K-catching bisimulation for Θ with abstract transition system $\Theta^\alpha = (S^\alpha, L, c, T^\alpha, s_0^\alpha, S_*^\alpha)$. Moreover, let $t^\alpha \in S^\alpha$ be an abstract state, and $l \in K$ be label. If $s^\alpha \in regr_\alpha(t^\alpha, l)$, then for every state $s \in Pre_\alpha(s^\alpha)$, there is a state $t \in Pre_\alpha(t^\alpha)$ with $s \in regr(t, l)$.*

Proof: Let $s^\alpha \in regr_\alpha(t^\alpha, l)$. By definition of $regr_\alpha$, there is a transition $(s^\alpha, l, t^\alpha) \in T^\alpha$. Then, by definition of Θ^α follows that there are states $s, t \in S$ with $\alpha(s) = s^\alpha$ and $\alpha(t) = t^\alpha$, so that $(s, l, t) \in T$. Since α is a K-catching bisimulation for Θ and $l \in K$, it holds for all $s' \in S$, such that $\alpha(s') = \alpha(s)$, that there is a state $t' \in S$ with $\alpha(t') = \alpha(t)$ so that $(s', l, t') \in T$, and therefore $s' \in regr(t', l)$. So for every state $s \in Pre_\alpha(s^\alpha)$, there is a state $t \in Pre_\alpha(t^\alpha)$ with $s \in regr(t, l)$. ∎

Lemma 1 implies that if no goal state of the original transition system can be regressed with the label sequence of a spurious path, then there must be some label on this sequence that is not contained in the current label subset.

Theorem 4 *Let $\Theta = (S, L, c, T, s_0, S_*)$ be a transition system and α be a K-catching bisimulation for Θ with abstract transition system $\Theta^\alpha = (S^\alpha, L, c, T^\alpha, s_0^\alpha, S_*^\alpha)$. Then for any $\langle a_1, \ldots, a_n \rangle \in L^n$ and state $t^\alpha \in S^\alpha$ holds: If $\langle a_1, \ldots, a_n \rangle$ is reverse-applicable in t^α, and $\langle a_2, \ldots, a_n \rangle$ is reverse-applicable in some $t \in Pre_\alpha(t^\alpha)$, but $\langle a_1, \ldots, a_n \rangle$ is not reverse-applicable in any $t \in Pre_\alpha(t^\alpha)$, then there must be a label a_i with $a_i \notin K$, for some $1 < i \leq n$.*

Proof: Assume for contradiction that all a_i are already contained in K, i.e. $a_i \in K$, for all $1 < i \leq n$. Let $r^\alpha \in regr_\alpha(t^\alpha, \langle a_1, \ldots, a_n \rangle)$. Such an abstract state must exist because this label sequence is reverse-applicable in t^α. By definition of $regr_\alpha$, there must be a transition $(r^\alpha, a_1, s^\alpha) \in T^\alpha$, for some state $s^\alpha \in regr_\alpha(t^\alpha, \langle a_2, \ldots, a_n \rangle)$. By definition of Θ^α, there must be states $r, s \in S$, $\alpha(r) = r^\alpha$, $\alpha(s) = s^\alpha$, so that $(r, a_1, s) \in T$. Now, when recursively applying Lemma 1, then it follows by $s^\alpha \in regr(t^\alpha, \langle a_2, \ldots, a_n \rangle)$ that there is a state $t \in Pre_\alpha(t^\alpha)$ such that $s \in regr(t, \langle a_2, \ldots, a_n \rangle)$. Therefore, $r \in regr(t, \langle a_1, \ldots, a_n \rangle)$ which contradicts the assumption that $\langle a_1, \ldots, a_n \rangle$ is not reverse-applicable in t. Hence, there must be at least one $1 < i \leq n$, such that $s_i \notin K$. ∎

Let $s_0^\alpha, a_1, s_1^\alpha, \ldots, a_n, s_n^\alpha$ be a spurious solution such that the label sequence $\langle a_1, \ldots, a_n \rangle$ is not reverse-applicable in all goal states $s_* \in S_*$. We find the index $1 \leq i < n$, so that $\langle a_{i+1}, \ldots, a_n \rangle$ is reverse-applicable in some $s_* \in S_*$, but $\langle a_i, \ldots, a_n \rangle$ is not reverse-applicable in any $s_* \in S_*$. Theorem 5 implies that adding the labels $a_{i+1}, \ldots, a_n$ to the current label subset is sufficient to exclude the considered spurious solution from the abstract transition system.
Moreover, for spurious paths whose label sequences are actually reverse-applicable in some goal state of the original transition system, but the regression does not end in the initial state, it follows immediately by Lemma 1 that there must be at least one label in this sequence that is not already contained in the current label subset.

Theorem 5 *Let $\Theta = (S, L, c, T, s_0, S_*)$ be a transition system and α be a K-catching bisimulation for Θ with abstract transition system $\Theta^\alpha = (S^\alpha, L, c, T^\alpha, s_0^\alpha, S_*^\alpha)$. Then for any state $t^\alpha \in S^\alpha$ and $\langle a_1, \ldots, a_n \rangle \in L^n$ holds: If $s_0^\alpha \in regr_\alpha(t^\alpha, \langle a_1, \ldots, a_n \rangle)$, and $s_0 \notin regr(t, \langle a_1, \ldots, a_n \rangle)$, for any $t \in Pre_\alpha(t^\alpha)$, then there must be an action a_i with $a_i \notin K$, for some $1 \leq i \leq n$.*

procedure *analyze*(Π, Θ^α)
  $P \leftarrow$ compute the first S optimal solutions of Θ^α
  $labels \leftarrow \emptyset$
  for $p \in P$ **do**
    if p is spurious **then**
      $labels \leftarrow labels \cup$ *exclude*(p)
    endif
  done
  return $labels$

Figure 2: Analyze procedure, as called by the abstraction refinement procedure (Figure 1) within each refinement step.

Proof: Assume for contradiction that $a_i \in K$, for all $1 \leq i \leq n$. Since $s_0^\alpha \in regr_\alpha(t^\alpha, \langle a_1, \ldots, a_n \rangle)$ and $\alpha(s_0) = s_0^\alpha$, by definition of Θ^α, it follows by recursively applying Lemma 1 that $s_0 \in regr(t, \langle a_1, \ldots, a_n \rangle)$, for some $t \in Pre_\alpha(t^\alpha)$. But this contradicts the assumption. Therefore, it must hold $a_i \notin K$, for at least one $1 \leq i \leq n$. ∎

Let $s_0^\alpha, a_1, s_1^\alpha, \ldots, a_n, s_n^\alpha$ be a spurious solution such that the label sequence $\vec{a} = \langle a_1, \ldots, a_n \rangle$ is reverse-applicable in some goal state $s_* \in S_*$. This means that $s_0 \notin regr(s_*, \vec{a})$ for all $s_* \in S_*$, and it follows from Theorem 5 that adding the labels $a_1, \ldots, a_n$ to the current label subset is sufficient to remove the considered spurious solution from the abstract transition system.

A trivial consequence of our results is that, if we keep running abstraction refinement and adding labels as described, then eventually all abstract solutions will actually be plans for the original planning task. Of course, we can stop as soon as that is the case for at least one abstract solution. As one would expect, such a stopping criterion is impractical: The abstractions required for it to occur are, in most cases, infeasibly large. In our implementation, described next, we use earlier cut-offs.

Implementation

Our techniques are implemented in *Fast Downward* (Helmert 2006), as an extension of the M&S approach. The overall refinement procedure was already depicted in Figure 1. It has 4 input parameters: (1) forward vs. backward propagation; (2) the termination criterion; (3) the number S of abstract solutions considered in any refinemrnt step; and (4) a size limit L on the abstract transition system. Parameters (1) and (3) are used in the analyze function, described next. Parameters (2) and (4) are used in the termination criterion.

Analysis Procedure

The *analyze* function is depicted in Figure 2. It computes the labels *that will be added to* K, in order to refine the K-catching bisimulation. Our current code is optimized for readability rather than efficiency, storing the entire set of optimal abstract solutions in memory, prior to analyzing them (instead, one could generate and analyze each solution one-by-one). As there can be a huge number of such solutions, this can cause serious memory issues. For the moment, we control this via the parameter S: As soon as S abstract solutions have been generated, we stop and proceed to the analysis step.

During the analysis each abstract solution is analyzed in either forward or backward manner, as specified by parameter (1). Whenever the considered solution is spurious, then *exclude* will apply the theorems shown above and returns the corresponding labels.

Termination Criteria

A practical termination criterion cannot require that an optimal solution of the abstract transition system matches exactly to a solution of the original transition system. However, to still obtain a useful abstraction heuristic, one has to require that the abstract optimal solutions correspond at least somewhat to paths of the original transition system. We experimented with 3 different kinds of termination criteria:

- **All.** This criterion is satisfied if none of the extracted optimal abstract solutions is spurious. This extreme criterion just serves to illustrate what happens when putting all computational effort into the abstraction.
- **Lower Bounding (LB).** This criterion requires a lower bound on the solution cost h_{min}, i.e., $h_{min} \leq h^*(s_0)$. When the cost of the optimal abstract solution is greater or equal than this value, then the refinement loop will be terminated. In other words, as soon as for the current K-catching bisimulation α holds that $h^\alpha(s_0) \geq h_{min}$, the refinement loop will be stopped.
- **Cost Increase Threshold (CIT).** This criterion requires a threshold $\beta \in [1, \infty)$. It essentially tests whether the increase of the estimated goal distance between the K-catching bisimulation α_i, and the K'-catching bisimulation α_{i+1} of two consecutive refinement steps i and $i+1$ is higher than the minimum allowed increase. In other words, the refinement of the abstraction will be stopped as soon as the quotient $h^{\alpha_{i+1}}(s_0)/h^{\alpha_i}(s_0)$ is less than the given β.

Empirical Evaluation

We ran a total of 32 different variants of our abstraction refinement approach, 7 other M&S configurations, two competing heuristics, and a blind search instance on a total of 280 instances of the 14 benchmarks from the track of optimal planners at IPC'11. The experiments are performed on an Intel Core i7-3770K processor, limiting the run time to 5 minutes and the memory usage to 2 GB.

We ran the abstraction refinement procedure with both forward propagation and backward propagation. The termination criterion is set to either *All*, or *LB* with $h_{min} = h^1(s_0)$ (Haslum and Geffner 2000), or *CIT* with $\beta \in \{1.40, 2.0\}$. For (3), we extract a maximum of either $S = \infty$, or $S = 10K$ optimal solutions from the abstract transition system. For (4), we set a bound on the number of states of the abstract transition system: $L = 100K$, or $L = \infty$.

We ran also BJOLP (Domshlak et al. 2011) and LM-cut (Helmert and Domshlak 2009), two methods that were used in the portfolio winning the 1st prize in the track of optimal planners at IPC 11.

Refinement	Backward propagation							
Termination	CIT $\beta = 1.4$		CIT $\beta = 2.0$		LB h^1		All	
L	∞	$100K$	∞	$100K$	∞	$100K$	∞	$100K$
barman	AM=4, SM=8, AT=8	AM=4, SM=8, AT=8	AM=4, SM=8, AT=5, ST=3	AM=4, SM=8, AT=5, ST=3	AM=12, AT=7, ST=1	AM=12, AT=7 ST=1	AM=12, AT=8	AM=12, AT=8
elevators	C=**10**, AM=5, SM=5	C=**10**, AM=5, SM=5	C=**10**, AM=5, SM=5	C=**10**, AM=5, SM=5	C=7, AM=13	C=8, AM=9, SM=3	C=3, AM=17	C=8, AM=9, SM=3
floortile	C=**2**, AM=18	C=**2**, AM=18	C=**2**, AM=18	C=**2**, AM=18	C=**2**, SM=18	C=**2**, SM=18	C=**2**, AM=18	C=**2**, AM=18
nomystery	C=**18**, SM=2	C=**18**, SM=2	C=**18**, SM=2	C=**18**, SM=2	C=12, SM=8	C=12, SM=8	C=8, AM=12	C=14, AM=4, SM=2
openstacks	C=0, AM=15, AT=5	C=**3**, AM=15, AT=2	C=0, AM=15, AT=5	C=**3**, AM=14, AT=3	C=**3**, AM=14, AT=3	C=**3**, AM=15, AT=2	C=0, AM=15, AT=5	C=**3**, AM=14, AT=3
parcprinter	C=10, AM=9, SM=1	C=10, AM=9, SM=1	C=10, AM=9, SM=1	C=10, AM=9, SM=1	C=**11**, SM=9	C=**11**, SM=9	C=6, AM=10, AT=4	C=9, AM=9, SM=1, AT=1
parking	C=1, AM=7, SM=9, ST=3	C=1, AM=7, SM=8, ST=4	C=1, AM=7, SM=8, ST=4	C=1, AM=7, SM=8, ST=4	C=**6**, SM=14	C=**6**, SM=14	C=0, AM=20	C=0, AM=20
pegsol	C=1, AM=14, AT=5	C=4, AM=14, SM=2	C=3, AM=15, AT=2	C=4, AM=14, SM=2	C=**9**, AM=9, SM=2	C=**9**, AM=9, SM=2	C=1, AM=15, AT=4	C=4, AM=14, SM=2
scanalyzer	C=4, AM=15	C=4, AM=15	C=4, AM=15	C=4, AM=15	C=**9**, AM=3, SM=7	C=**9**, AM=3, SM=7	C=3, AM=16	C=3, AM=16
sokoban	C=**9**, AM=11	C=**9**, AM=11	C=**9**, AM=11	C=**9**, AM=11	C=6, AM=14	C=6, AM=14	C=0, AM=20	C=0, AM=20
tidybot	C=1, AM=19	C=1, AM=19	C=**11**, AM=6, ST=3	C=10, AM=6, ST=4	C=3, AM=17	C=3, AM=17	C=1, AM=19	C=1, AM=19
transport	C=**6**, SM=14	C=**6**, SM=14	C=**6**, SM=14	C=**6**, SM=14	C=5, AM=14, SM=1	C=**6**, AM=9, SM=5	C=4, AM=14, AT=2	C=**6**, AM=9, SM=5
visitall	C=8, AM=3, AT=9	C=8, AM=3, AT=9	C=8, AM=3, AT=9	C=8, AM=3, AT=9	C=**16**, SM=4	C=**16**, SM=4	C=8, AM=2, AT=10	C=8, AM=3, AT=9
woodworking	C=2, AM=17, SM=1	C=3, AM=16, SM=1	C=3, AM=16, SM=1	C=3, AM=16, SM=1	C=**5**, SM=15	C=**5**, SM=15	C=2, AM=18	C=3, AM=17
$\sum$	C=72, AM=137, SM=40, AT=27, ST=3	C=79, AM=136, SM=41, AT=19, ST=4	C=85, AM=124, SM=39, AT=21, ST=10	C=88, AM=122, SM=41, AT=17, ST=11	C=94, AM=96, SM=78, AT=10, ST=1	C=**96**, AM=88, SM=85, AT=9, ST=1	C=38, AM=208, AT=33	C=61, AM=184, SM=13, AT=21

Table 1: Experiment data for the abstraction refinement variants. S is always set to 10K. C represents the number of completed tasks. The other values represent the number of violations of the requirements, split into violations of the memory requirement (during refinement process: AM, during the search: SM), and into violations of the run time requirement (during the refinement process: AT, during the search: ST). The highest amount of solved task per domain is highlighted in bold.

We ran 4 M&S configurations based on K-catching bisimulations. The label subset is either computed by the approximation technique *Backward* h^1, or by the approximation technique *IntAbs* (Katz, Hoffmann, and Helmert 2012), and the size limit is set to either $N = 100K$, or $N = \infty$. Additionally, we ran 2 variants of greedy-bisimulation (Nissim, Hoffmann, and Helmert 2011), one with size limit $N = 100K$, and the other one with $N = \infty$. Finally, we ran another M&S instance using full bisimulation, without using a size limit ($N = \infty$).

The size limit N influences the size of the abstract transition system in a different way than L does. If the amount of states of the abstract transition system reaches this bound N, then the shrinking strategy of the M&S abstraction is forced to aggregate more states by dropping the bisimulation requirement. In contrast, whether the size limit L is exceeded is only tested between the refinement steps. If it is, then the refinement process is aborted, so one catches less labels instead of dropping the bisimulation requirement.

Backward h^1 computes the label subset by collecting all labels that occur within the radius, given by the product of $h^1(s_0)$ and some parameter $\beta \in [0, 1]$, around the goal states of the actual state space, i. e. all labels that are used in an optimal path to some goal state, with cost less or equal than this radius. If $\beta = 0$, then the smallest β is considered that leads a non-empty label subset. *IntAbs* computes the label subset by computing the standard bisimulation until some size limit M is reached. Afterwards, the exact label subset in the resulting abstract transition system, i. e. either all globally relevant, or all $h^\alpha(s_0)$-relevant labels, is computed. *Greedy bisimulation* relaxes the bisimulation criterion by ignoring all transitions (s, l, t) with $h^*(s) < h^*(t) + c(l)$, i.e., transitions that are not used in any optimal solution.

The number of solved tasks, as well as the number of violations of the run time and memory requirement for selected abstraction refinement variants are shown in Table 1. We show data only for backward refinement and for $S = 10K$, as the respective other settings are dominated consistently ($S = \infty$, resulted, in most of the cases, in a violation of the memory requirement within the first few refinement steps). In Table 2, we show the average ratio of considered labels for solved tasks, as well as the average amount of refinement steps for solved tasks.

No variant of abstraction refinement could solved any task of *barman*. The backward propagation variant, using *All* and $L = \infty$, shows that excluding all (extracted) spurious so-

Refinement	Backward propagation							
Termination	CIT $\beta = 1.4$		CIT $\beta = 2.0$		LB h^1		All	
L	∞	$100K$	∞	$100K$	∞	$100K$	∞	$100K$
elevators	9.4, 2.1	9.4, 2.1	9.4, 2.1	9.4, 2.1	15.4, 2.9	12.1, 2.5	49.0, 12.3	17.0, 3.2
floortile	25.0, 1.0	25.0, 1.0	25.0, 1.0	25.0, 1.0	0.0, 0.0	0.0, 0.0	50.7, 2.0	50.7, 2.0
nomystery	14.4, 1.2	14.4, 1.2	1.8, 1.0	1.8, 1.0	0.0, 0.0	0.0, 0.0	51.1, 3.4	30.7, 2.6
openstacks	–	57.3, 1.3	–	57.3, 1.3	45.8, 1.0	45.8, 1.0	–	57.3, 1.3
parcprinter	11.3, 1.0	11.3, 1.0	11.3, 1.0	11.3, 1.0	0.0, 0.0	0.0, 0.0	37.3, 14.8	26.7, 8.0
parking	0.3, 1.0	0.3, 1.0	0.3, 1.0	0.3, 1.0	0.0, 0.0	0.0, 0.0	–	–
pegsol	55.1, 4.0	10.8, 2.0	48.6, 3.0	10.8, 2.0	2.8, 1.1	2.8, 1.1	55.1, 4.0	10.8, 2.0
scanalyzer	29.6, 1.2	29.6, 1.2	20.1, 1.0	20.1, 1.0	0.0, 0.0	0.0, 0.0	58.5, 2.0	39.6, 1.7
sokoban	5.7, 1.1	5.7, 1.1	5.7, 1.1	5.7, 1.1	0.0, 0.2	0.0, 0.2	–	–
tidybot	0.1, 1.0	0.1, 1.0	0.1, 1.0	0.1, 1.0	0.0, 0.7	0.0, 0.7	0.1, 1.0	0.1, 1.0
transport	9.3, 2.3	9.3, 2.3	7.4, 2.0	7.4, 2.0	16.7, 7.4	14.9, 5.2	26.5, 15.0	16.5, 5.8
visitall	60.8, 1.1	60.8, 1.1	53.5, 1.0	53.5, 1.0	0.0, 0.0	0.0, 0.0	81.9, 2.4	76.0, 1.6
woodworking	24.2, 1.0	20.1, 1.0	20.1, 1.0	20.1, 1.0	0.0, 0.0	0.0, 0.0	38.1, 3.0	25.8, 1.3
∅	20.4, 1.5	19.5, 1.3	16.9, 1.4	17.1, 1.3	6.2, 1.0	5.8, 0.8	44.8, 6.0	31.9, 2.8

Table 2: Results for backward propagation. S is always set to 10K. The first value is the average ratio of caught labels (%), for solved tasks. The last value is the average amount of refinement steps, for solved tasks.

lutions from the abstract transition system is not feasible in practice. It can only solve 38 instances within the given memory and run time limits. The main reason for not finishing a task was the violation of the memory requirement during the construction of the abstractions. This implies that the resulting abstract transition systems are too big to store in memory. Bounding the size of the abstract transition system, i.e., $L = 100K$, improves the performance of *All*, while reducing the number of violations of the memory, and run time constraints (during the refinement process). Overall, it can solve 23 tasks more than the version without size limit. As can be observed in Table 2, bounding the size of the abstract transition system for *All* reduces the average amount of refinement steps, and consequently reduces the overall amount of caught labels. This does generally hold, independent of the used termination criterion. However, the difference between $L = 100K$ and $L = \infty$ for the other termination criteria is not as big as for *All*. This implies that the termination criteria *CIT* ($\beta \in \{1.4, 2.0\}$), and *LB* h^1 are often fulfilled even before the size limit is reached.

Comparing $\beta = 1.4$ and $\beta = 2.0$ for the termination criterion *CIT*, the latter is considerably better. For $L = \infty$, *CIT* with $\beta = 2.0$ is equally good in 11 domains, and it is better in 3 domains. Overall, it can solve 13 tasks more than *CIT* with $\beta = 1.4$, while reducing the number of violations of the memory requirement during the refinement process. When using $\beta = 2.0$, one terminates also slightly faster, i.e., one catches less labels, than when using $\beta = 1.4$.

Consider the termination criterion *LB* h^1. Surprisingly, many entries of Table 2 are 0. This implies that the first abstract transition system, i.e., the abstract transition system of a K-catching bisimulation with empty K, does already fulfill the termination criterion *LB* h^1 in many cases; evidently, in these cases $h^1(s_0)$ is very small. But if the termination criterion holds initially, then there is actually no refinement step executed, which results in an empty label subset. However, in our experiments, using backward propagation, *LB* h^1 and $L = 100K$ was the best abstraction refinement variant.

Comparing the per domain results of *LB* h^1 and *CIT* with $\beta = 2.0$, without a size limit, the former one can solve more tasks in 7 domains, is equally good in 2 domains, and worse in 5 domains. The comparison of the reasons for not completing a task shows that the *CIT* variant mainly fails due to a violation of the memory, or run time constraint during the refinement process, whereas *LB* h^1 often fails due to a violation of the memory constraint during the search. That is only logical: A $\emptyset$-catching bisimulation is very cheap to compute – it only distinguishes non-goal states from goal states – but does not provide a lot of search guidance.[1]

Table 3 compares the coverage data of the best abstraction refinement variant with the results of the non abstraction-refinement configurations.

In general, for all other M&S variants, using the size limit $N = 100K$ can solve more tasks than the equivalent variants without a size limit. *Greedy bisimulation* with bounded size is almost identical to *Backward* h^1 with bounded size. Moreover, comparing *IntAbs* and *Backward* h^1, both with $N = 100K$, the former one can solve slightly more tasks.

Comparing the best abstraction refinement variant, i.e., using backward propagation with the termination criterion LB h^1, $L = 100K$, and $S = 10K$, with *IntAbs*, using $N = 100K$, and $M = 10K$, then the abstraction refinement approach is better in 1 domain, equally good in 4 domains, and worse in 9 domains. It gives almost identical results in 3 domains. Overall, the best configuration of this experiment is LM-cut. Comparing it to the best M&S configuration, *IntAbs* with $N = 100K$ and $M = 10K$, LM-cut can solve more tasks in 9 domains, it is equally good in 4 domains, and worse in 1 domain.

Conclusions

In theory, abstraction refinement appears to be a suitable way to extract meaningful label subsets for K-catching bisimulations. Practical results, thus far, are disappointing. Despite this, we believe the approach is not without hope. As our empirical data vividly demonstrates, our current strategies

[1]Note that the failures during abstraction, for LB, are almost exclusively due to domains where K is *not* empty.

Approach	Abstraction Refinement Backward, *LB* h^1		IntAbs		Backward h^1		Greedy bisim.		Full bisim.	LM-cut	BJOLP	Blind
L/N	∞	$100K$	∞	$100K$	∞	$100K$	∞	$100K$	∞			
$S/M/\beta/M$	$10K$	$10K$	$10K$	$10K$	0	0.25	∞	$100K$	∞			
barman	0	0	**4**	**4**	**4**	**4**	0	**4**	0	**4**	**4**	**4**
elevators	7	8	9	11	9	11	4	11	4	**15**	14	9
floortile	2	2	2	2	2	2	2	2	2	**6**	2	2
nomystery	12	12	15	14	13	**20**	12	**20**	12	14	**20**	8
openstacks	3	3	**14**	**14**	0	**14**	3	**14**	3	12	11	**14**
parcprinter	11	11	12	12	11	12	8	12	8	**13**	10	6
parking	**6**	**6**	5	**6**	0	0	0	0	0	1	1	0
pegsol	9	9	**17**	**17**	13	**17**	15	**17**	14	**17**	**17**	**17**
scanalyzer	9	9	9	9	3	8	3	8	3	**10**	3	9
sokoban	6	6	19	19	17	20	4	19	5	**20**	**20**	17
tidybot	3	3	5	5	8	0	0	0	0	13	**14**	6
transport	5	**6**	**6**	**6**	**6**	**6**	4	**6**	4	**6**	**6**	**6**
visitall	**16**	**16**	9	9	8	9	8	9	8	10	10	9
woodworking	5	5	6	6	5	6	3	7	4	**10**	9	2
$\sum$	94	96	132	134	99	129	66	129	67	**151**	141	109

Table 3: Comparison of solved tasks over the selected domains. Best results are highlighted in bold.

spend way too much time and memory in the abstraction process. There are many possibilities to control the practical CEGAR process differently, reducing that overhead.

A major point regards greedier termination criteria. We have already seen that setting β in CIT to 2 rather than 1.4 reduces abstraction effort considerably. A first step thus simply is to systematically experiment with largr values of β. For LB, following one of the methods proposed by Katz et al., one could also introduce a β parameter and terminate as soon as the heuristic value has reached $h^1(s_0) * \beta$.

Another issue is the amount of abstract solutions considered, which was huge in many cases. It appears that many spurious solutions share the same error label, and therefore it would be enough to consider only a subset of representative solutions in the refinement process. How to find such good subsets efficiently is an interesting open question.

Finally, label reduction has not been covered in this work, but in some cases can reduce the size of the abstract transition system drastically. It remains to investigate how label reduction can be integrated into our approach.

Acknowledgments. We thank the anonymous HSDIP'13 reviewers, whose comments helped to improve the paper.

References

Clarke, E.; Grumberg, O.; Jha, S.; Lu, Y.; and Veith, H. 2003. Counterexample-guided abstraction refinement for symbolic model checking. *Journal of the Association for Computing Machinery* 50(5):752–794.

Domshlak, C.; Helmert, M.; Karpas, E.; Keyder, E.; Richter, S.; Röger, G.; Seipp, J.; and Westphal, M. 2011. BJOLP: The big joint optimal landmarks planner. In *IPC 2011 planner abstracts*, 91–95.

Haslum, P., and Geffner, H. 2000. Admissible heuristics for optimal planning. In Chien, S.; Kambhampati, R.; and Knoblock, C., eds., *Proceedings of the 5th International Conference on Artificial Intelligence Planning Systems (AIPS-00)*, 140–149. Breckenridge, CO: AAAI Press, Menlo Park.

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., 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 2009)*, 162–169. AAAI Press.

Helmert, M.; Haslum, P.; and Hoffmann, J. 2007. Flexible abstraction heuristics for optimal sequential planning. In Boddy, M.; Fox, M.; and Thiebaux, S., eds., *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS-07)*, 176–183. Providence, Rhode Island, USA: Morgan Kaufmann.

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

Katz, M.; Hoffmann, J.; and Helmert, M. 2012. How to relax a bisimulation? In Bonet, B.; McCluskey, L.; Silva, J. R.; and Williams, B., eds., *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS 2012)*. AAAI Press.

Milner, R. 1990. Operational and algebraic semantics of concurrent processes. In van Leeuwen, J., ed., *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics*. Elsevier and MIT Press. 1201–1242.

Nissim, R.; Hoffmann, J.; and Helmert, M. 2011. Computing perfect heuristics in polynomial time: On bisimulation and merge-and-shrink abstraction in optimal planning. In Walsh, T., ed., *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI'11)*, 1983–1990. AAAI Press/IJCAI.

Towards Rational Deployment of Multiple Heuristics in A*

David Tolpin
Tal Beja
Solomon Eyal Shimony
CS Department
Ben-Gurion University
Israel
{tolpin,bejat,shimony}@cs.bgu.ac.il

Ariel Felner
ISE Department
Ben-Gurion University
Israel
felner@bgu.ac.il

Erez Karpas
Faculty of IE&M
Technion
Israel
karpase@gmail.com

Abstract

The obvious way to use several admissible heuristics in A^* is to take their maximum. In this paper we aim to reduce the time spent on computing heuristics. We discuss *Lazy* A^*, a variant of A^* where heuristics are evaluated lazily: only when they are essential to a decision to be made in the A^* search process. We present a new rational meta-reasoning based scheme, *rational lazy* A^*, which decides whether to compute the more expensive heuristics at all, based on a myopic value of information estimate. Both methods are examined theoretically. Empirical evaluation on several domains supports the theoretical results, and shows that lazy A^* and rational lazy A^* are state-of-the-art heuristic combination methods.

1 Introduction

The A^* algorithm [Hart *et al.*, 1968] is a best-first heuristic search algorithm guided by the cost function $f(n) = g(n) + h(n)$. If the heuristic $h(n)$ is admissible (never overestimates the real cost to the goal) then the set of nodes expanded by A^* is both necessary and sufficient to find the optimal path to the goal [Dechter and Pearl, 1985].

This paper examines the case where we have several available admissible heuristics. Clearly, we can evaluate all these heuristics, and use their *maximum* as an admissible heuristic, a scheme we call A^*_{MAX}. The problem with naive maximization is that all the heuristics are computed for all the generated nodes. In order to reduce the time spent on heuristic computations, Lazy A^* (or LA^*, for short) evaluates the heuristics one at a time, lazily. When a node n is generated, LA^* only computes one heuristic, $h_1(n)$, and adds n to OPEN. Only when n re-emerges as the top of OPEN is another heuristic, $h_2(n)$, evaluated; if this results in an increased heuristic estimate, n is re-inserted into OPEN. This idea was briefly mentioned by Zhang and Bacchus (2012) in the context of the MAXSAT heuristic for planning domains. LA^* is as informative as A^*_{MAX}, but can significantly reduce search time, as we will not need to compute h_2 for many nodes. In this paper we provide a deeper examination of LA^*, and characterize the savings that it can lead to. In addition, we describe several technical optmizations for LA^*.

LA^* reduces the search time, while maintaining the informativeness of A^*_{MAX}. However, as noted by Domshlak *et al.* (2012), if the goal is to reduce search time, it may be better to compute a fast heuristic on several nodes, rather than to compute a slow but informative heuristic on only one node. Based on this idea, they formulated *selective max* (Sel-MAX), an online learning scheme which chooses one heuristic to compute at each state. Sel-MAX chooses to compute the more expensive heuristic h_2 for node n when its classifier predicts that $h_2(n) - h_1(n)$ is greater than some threshold, which is a function of heuristic computation times and the average branching factor. Felner *et al.* (2011) showed that randomizing a heuristic and applying *bidirectional pathmax* (BPMX) might sometimes be faster than evaluating all heuristics and taking the maximum. This technique is only useful in undirected graphs, and is therefore not applicable to some of the domains in this paper. Both Sel-MAX and Random compute the resulting heuristic *once*, before each node is added to OPEN while LA^* computes the heuristic lazily, in different steps of the search. In addition, both randomization and Sel-MAX save heuristic computations and thus reduce search time in many cases. However, they might be less informed than pure maximization and as a result expand a larger number of nodes.

We then combine the ideas of lazy heuristic evaluation and of trading off more node expansions for less heuristic computation time, into a *new* variant of LA^* called *rational lazy* A^* (RLA^*). RLA^* is based on rational meta-reasoning, and uses a myopic *value-of-information* criterion to decide whether to compute $h_2(n)$ or to bypass the computation of h_2 and expand n immediately when n re-emerges from OPEN. RLA^* aims to reduce search time, even at the expense of more node expansions than A^*_{MAX}.

Empirical results on variants of the 15-puzzle and on numerous planning domains demonstrate that LA^* and RLA^* lead to state-of-the-art performance in many cases.

2 Lazy A^*

Throughout this paper we assume for clarity that we have two available admissible heuristics, h_1 and h_2. Extension to multiple heuristics is straightforward, at least for LA^*. Unless stated otherwise, we assume that h_1 is faster to compute than h_2 but that h_2 is *weakly more informed*, i.e., $h_1(n) \leq h_2(n)$ for the majority of the nodes n, although counter cases where $h_1(n) > h_2(n)$ are possible. We say that h_2 *dominates* h_1, if such counter cases do not exist and $h_2(n) \geq h_1(n)$ for *all* nodes n. We use $f_1(n)$ to denote $g(n) + h_1(n)$. Likewise, $f_2(n)$ denotes $g(n) + h_2(n)$, and $f_{max}(n)$ denotes

Algorithm 1: Lazy A^*

```
   Input: LAZY-A*
1  Apply all heuristics to Start
2  Insert Start into OPEN
3  while OPEN not empty do
4      n ← best node from OPEN
5      if Goal(n) then
6          return trace(n)
7      if h2 was not applied to n then
8          Apply h2 to n
9          insert n into OPEN
10         continue      //next node in OPEN
11     foreach child c of n do
12         Apply h1 to c.
13         insert c into OPEN
14     Insert n into CLOSED
15 return FAILURE
```

$g(n) + \max(h_1(n), h_2(n))$. We denote the cost of the optimal solution by C^*. Additionally, we denote the computation time of h_1 and of h_2 by t_1 and t_2, respectively and denote the overhead of an *insert/pop* operation in OPEN by t_o. Unless stated otherwise we assume that t_2 is much greater than $t_1 + t_o$. LA^* thus mainly aims to reduce computations of h_2.

The pseudo-code for LA^* is depicted as Algorithm 1, and is very similar to A^*. In fact, without lines 7 – 10, LA^* would be identical to A^* using the h_1 heuristic. When a node n is generated we only compute $h_1(n)$ and n is added to OPEN (Lines 11 – 13), without computing $h_2(n)$ yet. When n is first removed from OPEN (Lines 7 – 10), we compute $h_2(n)$ and reinsert it into OPEN, this time with $f_{max}(n)$.

It is easy to see that LA^* is as informative as A^*_{MAX}, in the sense that both A^*_{MAX} and LA^*expand a node n only if $f_{max}(n)$ is the best f-value in OPEN. Therefore, LA^* and A^*_{MAX} generate and expand and the same set of nodes, up to differences caused by tie-breaking.

In its general form A^* generates many nodes that it does not expand. These nodes, called *surplus* nodes [Felner *et al.*, 2012], are in OPEN when we expand the goal node with $f = C^*$. All nodes in OPEN with $f > C^*$ are surely surplus but some nodes with $f = C^*$ may also be surplus. The number of surplus nodes in OPEN can grow exponentially in the size of the domain, resulting in significant costs.

LA^* avoids h_2 computations for many of these surplus nodes. Consider a node n that is generated with $f_1(n) > C^*$. This node is inserted into OPEN but will never reach the top of OPEN, as the goal node will be found with $f = C^*$. In fact, if OPEN breaks ties in favor of small h-values, the goal node with $f = C^*$ will be expanded as soon as it is generated and such savings of h_2 will be obtained for some nodes with $f_1 = C^*$ too. We refer to such nodes where we saved the computation of h_2 as *good* nodes. Other nodes, those with $f_1(n) < C^*$ (and some with $f_1(n) = C^*$) are called *regular nodes* as we apply both heuristics to them.

A^*_{MAX} computes both h_1 and h_2 for all generated nodes, spending time $t_1 + t_2$ on all generated nodes. By contrast, for *good* nodes LA^* only spends t_1, and saves t_2. In the basic implementation of LA^* (as in algorithm 1) *regular* nodes are inserted into OPEN twice, first for h_1 (Line 13) and then for h_2 (Line 9) while *good* nodes only enter OPEN once (Line 13). Thus, LA^* has some extra overhead of OPEN operations for *regular nodes*. We distinguish between 3 classes of nodes:
(1) *expanded regular* (ER) — nodes that were expanded after both heuristics were computed.
(2) *surplus regular* (SR) — nodes for which h_2 was computed but are still in OPEN when the goal was found.
(3) *surplus good* (SG) — nodes for which only h_1 was computed by LA^* when the goal was found.

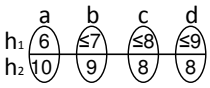


Figure 1: Example of HBP

Alg	ER	SR	SG
A^*_{MAX}	$t_1 + \mathbf{t_2} + 2t_o$	$t_1 + \mathbf{t_2} + t_o$	$t_1 + \mathbf{t_2} + t_o$
LA^*	$t_1 + \mathbf{t_2} + 4t_o$	$t_1 + \mathbf{t_2} + 3t_o$	$t_1 + t_o$

Table 1: Time overhead for A^*_{MAX} and for LA^*

The time overhead of A^*_{MAX} and LA^* is summarized in Table 1. LA^* incurs more OPEN operations overhead, but saves h_2 computations for the SG nodes. When t_2 (**boldface** in table 1) is significantly greater than both t_1 and t_o there is a clear advantage for LA^*, as seen in the SG column.

3 Enhancements to Lazy A^*

Several enhancements can improve basic LA^* (Algorithm 1), which are effective especially if t_1 and t_o are not negligible.

3.1 OPEN bypassing

Suppose node n was just generated, and let f_{best} denote the best f-value currently in OPEN. LA^* evaluates $h_1(n)$ and then inserts n into OPEN. However, if $f_1(n) \leq f_{best}$, then n will immediately reach the top of OPEN and h_2 will be computed. In such cases we can choose to compute $h_2(n)$ right away (after Line 12 in Algorithm 1), thus saving the overhead of inserting n into OPEN and popping it again at the next step ($= 2 \times t_o$). For such nodes, LA^* is identical to A^*_{MAX}, as both heuristics are computed before the node is added to OPEN. This enhancement is called *OPEN bypassing* (OB). It is a reminiscent of the *immediate expand* technique applied to generated nodes [Stern *et al.*, 2010; Sun *et al.*, 2009]. The same technique can be applied when n again reaches the top of OPEN when evaluating $h_2(n)$; if $f_2(n) \leq f_{best}$, expand n right away. With OB, LA^* will incur the extra overhead of two OPEN cycles only for nodes n where $f_1(n) > f_{best}$ and then later $f_2(n) > f_{best}$.

3.2 Heuristic bypassing

Heuristic bypassing (HBP) is a technique that allows A^*_{MAX} to omit evaluating one of the two heuristics. HBP is probably used by many implementers, although to the best of our knowledge, it never appeared in the literature. HBP works for a node n under the following two preconditions: **(1)** the operator between n and its parent p is bidirectional, and **(2)** both heuristics are *consistent* [Felner *et al.*, 2011].

Let C be the cost of the operator. Since the heuristic is consistent we know that $|h(p) - h(n)| \leq C$. Therefore, $h(p)$

provides the following upper- and lower-bounds on $h(n)$ of $h(p) - C \leq h(n) \leq h(p) + C$. We thus denote $\underline{h(n)} = h(p) - C$ and $\overline{h(n)} = h(p) + C$.

To exploit HBP in A^*_{MAX}, we simply skip the computation of $h_1(n)$ if $\overline{h_1(n)} \leq \underline{h_2(n)}$, and vice versa. For example, consider node a in Figure 1, where all operators cost 1, $h_1(a) = 6$, and $h_2(a) = 10$. Based on our bounds $h_1(b) \leq 7$ and $h_2(c) \geq 9$. Thus, there is no need to check $h_1(b)$ as $h_2(b)$ will surely be the maximum. We can propagate these bounds further to node c. $h_2(c) = 8$ while $h_1(c) \leq 8$ and again there is no need to evaluate $h_1(c)$. Only in the last node d we get that $h_2(d) = 8$ but since $h_1(c) \leq 9$ then $h_1(c)$ can potentially return the maximum and should thus be evaluated.

HBP can be combined in LA^* in a number of ways. We describe the variant we used. LA^* aims to avoid needless computations of h_2. Thus, when $\overline{h_1(n)} < \underline{h_2(n)}$, we delay the computation of $h_2(n)$ and add n to OPEN with $f(n) = g(n) + \underline{h_2(n)}$ and continue as in LA^*. In this case, we saved t_1, delayed t_2 and used $\underline{h_2(n)}$ which is more informative than $h_1(n)$. If, however, $\overline{h_1(n)} \geq \underline{h_2(n)}$, then we compute $h_1(n)$ and continue regularly. We note that HBP incurs the time and memory overheads of computing and storing four bounds and should only be applied if there is enough memory and if t_1 and especially t_2 are very large.

4 Rational Lazy A^*

LA^* offers us a very strong guarantee, of expanding the same set of nodes as A^*_{MAX}. However, often we would prefer to expand more states, if it means reducing search time. We now present *Rational Lazy A** (RLA^*), an algorithm which attempts to optimally manage this tradeoff.

Using principles of rational meta-reasoning [Russell and Wefald, 1991], theoretically every algorithm action (heuristic function evaluation, node expansion, open list operation) should be treated as an action in a sequential decision-making meta-level problem: actions should be chosen so as to achieve the minimal expected search time. However, the appropriate general meta-reasoning problem is extremely hard to define precisely and to solve optimally.

Therefore, we focus on just one decision type, made in the context of LA^*, when n re-emerges from OPEN (Line 7). We have two options: **(1)** Evaluate the second heuristic $h_2(n)$ and add the node back to OPEN (Lines 7-10) like LA^*, or **(2)** bypass the computation of $h_2(n)$ and expand n right way (Lines 11 -13), thereby saving time by not computing h_2, at the risk of additional expansions and evaluations of h_1. In order to choose rationally, we define a criterion based on value of information (VOI) of evaluating $h_2(n)$ in this context.

The only addition of RLA^* to LA^* is the option to bypass h_2 computations (Lines 7-10). Suppose that we choose to compute h_2 — this results in one of the following outcomes:
1: n is still expanded, either now or eventually.
2: n is re-inserted into OPEN, and the goal is found without ever expanding n.

Computing h_2 is helpful only in outcome 2, where potential time savings are due to pruning a search subtree at the expense of the time $t_2(n)$. However, whether outcome 2 takes place after a given state is not known to the algorithm until the goal is found, and the algorithm must decide whether to evaluate h_2 according to what it *believes to be* the probability of each of the outcomes. We derive a *rational policy* for when to evaluate h_2, under the myopic assumption that the algorithm continues to behave like LA^* afterwards (i.e., it will never again consider bypassing the computation of h_2).

The time wasted by being sub-optimal in deciding whether to evaluate h_2 is called the *regret* of the decision. If $h_2(n)$ is not helpful and we decide to compute it, the effort for evaluating $h_2(n)$ turns out to be wasted. On the other hand, if $h_2(n)$ is helpful but we decide to bypass it, we needlessly expand n. Due to the myopic assumption, RLA^* would evaluate both h_1 and h_2 for all successors of n.

	Compute h_2	Bypass h_2
h_2 helpful	0	$t_e + (b(n) - 1)t_d$
h_2 not helpful	t_d	0

Table 2: Regret in Rational Lazy A*

Table 2 summarizes the regret of each possible decision, for each possible future outcome; each column in the table represents a decision, while each row represents a future outcome. In the table, t_d is the to time compute h_2 and re-insert n into OPEN thus delaying the expansion of n, t_e is the time to remove n from OPEN, expand n, evaluate h_1 on each of the $b(n)$ ("local branching factor") children $\{n'\}$ of n, and insert $\{n'\}$ into the open list. Computing h_2 needlessly wastes time t_d. Bypassing h_2 computation when h_2 would have been helpful wastes $t_e + b(n)t_d$ time, but because computing h_2 would have cost t_d, the regret is $t_e + (b(n) - 1)t_d$.

Let us denote the probability that h_2 is helpful by p_h. The expected regret of computing h_2 is thus $(1 - p_h)t_d$. On the other hand, the expected regret of bypassing h_2 is $p_h(t_e + (b(n) - 1)t_d)$. As we wish to minimize the expected regret, we should thus evaluate h_2 just when:

$$(1 - p_h)t_d < p_h(t_e + (b(n) - 1)t_d) \tag{1}$$

or equivalently:

$$(1 - b(n)p_h)t_d < p_h t_e \tag{2}$$

If $p_h b(n) \geq 1$, then the expected regret is minimized by always evaluating h_2, regardless of the values of t_d and t_e. In these cases, RLA^* cannot be expected to do better than LA^*. For example, in the 15-puzzle and its variants, the effective branching factor is ≈ 2. Therefore, if h_2 is expected to be helpful for more than half of the nodes n on which LA^* evaluates $h_2(n)$, then one should simply use LA^*.

For $p_h b(n) < 1$, the decision of whether to evaluate h_2 depends on the values of t_d and t_e:

$$\textbf{evaluate } h_2 \textbf{ if } t_d < \frac{p_h}{1 - p_h b(n)} t_e \tag{3}$$

Denote by t_c the time to generate the children of n. Then:

$$\begin{aligned} t_d &= t_2 + t_o \\ t_e &= t_o + t_c + b(n)t_1 + b(n)t_o \end{aligned} \tag{4}$$

By substituting (4) into (3), obtain: **evaluate** h_2 **if**:

$$t_2 + t_o < \frac{p_h\left[t_c + b(n)t_1 + (b(n) + 1)t_o\right]}{1 - p_h b(n)} \tag{5}$$

	A^*		LA^*				RLA^*(Using Eq. 6)				
lookahead	generated	time	generated	Good1	h_2	time	generated	Good1	Good2	h_2	time
2	1,206,535	0.707	1,206,535	391,313	815,213	0.820	1,309,574	475,389	394,863	439,314	0.842
4	1,066,851	0.634	1,066,851	333,047	733,794	0.667	1,169,020	411,234	377,019	380,760	0.650
6	889,847	**0.588**	889,847	257,506	632,332	0.533	944,750	299,470	239,320	405,951	0.464
8	740,464	0.648	740,464	196,952	543,502	**0.527**	793,126	233,370	218,273	341,476	0.377
10	611,975	0.843	611,975	145,638	466,327	0.671	889,220	308,426	445,846	134,943	**0.371**
12	**454,130**	0.927	**454,130**	95,068	359,053	0.769	807,846	277,778	428,686	**101,378**	0.429

Table 3: Weighted 15 puzzle: comparison of $A^*_{\max}$, Lazy A^*, and Rational Lazy A^*

The factor $\frac{p_h}{1-p_h b(n)}$ depends on the potentially unknown probability p_h, making it difficult to reach the optimum decision. However, if our goal is just to do better than LA^*, then it is safe to replace p_h by an upper bound on p_h. Note that the values p_h, t_1, t_2, t_c may actually be variables that depend in complicated ways on the state of the search. Despite that, the very crude model we use, assuming that they are setting-specific constants, is sufficient to achieve improved performance, as shown in Section 5.

We now turn to implementation-specific estimation of the runtimes. OPEN in A^* is frequently implemented as a priority queue, and thus we have, approximately, $t_o = \tau \log N_o$ for some τ, where the size of OPEN is N_o. Evaluating h_1 is cheap in many domains, as is the case with Manhattan Distance (MD) in discrete domains, t_o is the most significant part of t_e. In such cases, rule (5) can be approximated as 6:

$$\textbf{evaluate } h_2 \textbf{ if } t_2 < \frac{\tau p_h}{1 - p_h b(n)}(b(n)+1)\log N_o \qquad (6)$$

Rule (6) recommends to evaluate h_2 mostly at late stages of the search, when the open list is large, and in nodes with a higher branching factor.

In other domains, such as planning, both t_1 and t_2 are significantly greater than both t_o and t_c, and terms not involving t_1 or t_2 can be dropped from (5), resulting in Rule (7):

$$\textbf{evaluate } h_2 \textbf{ if } \frac{t_2}{t_1} < \frac{p_h b(n)}{1 - p_h b(n)} \qquad (7)$$

The right hand side of (7) grows with $b(n)$, and here it is beneficial to evaluate h_2 only for nodes with a sufficiently large branching factor.

5 Empirical evaluation

We now present our empirical evaluation of LA^* and RLA^*, on variants of the 15-puzzle and on planning domains.

5.1 Weighted 15 puzzle

We first provide evaluations on the weighted 15-puzzle variant [Thayer and Ruml, 2011], where the cost of moving each tile is equal to the number on the tile. We used a subset of 36 problem instances (out of the 100 instances of Korf (1985)) which could be solved with 2Gb of RAM and 15 minutes timeout using the Weighted Manhattan heuristic (WMD) for h_1. As the expensive and informative heuristic h_2 we use a heuristic based on lookaheads [Stern *et al.*, 2010]. Given a bound d we applied a bounded depth-first search from a node n and backtracked when we reached leaf nodes l for which $g(l) + WMD(l) > g(n) + WMD(n) + d$. f-values from leaves were propagated to n.

Table 3 presents the results averaged on all instances solved. The runtimes are reported relative to the time of A^* with WMD (with no lookahead), which generated 1,886,397 nodes (not reported in the table). The first 3 columns of Table 3 show the results for A^* with the lookahead heuristic for different lookahead depths. The best time is achieved for lookahead 6 (0.588 compared to A^* with WMD). The fact that the time does not continue to decrease with deeper lookaheads is clearly due to the fact that although the resulting heuristic improves as a function of lookahead depth (expanding and generating fewer nodes), the increasing overheads of computing the heuristic eventually outweights savings due to fewer expansions.

The next 4 columns show the results for LA^* with WMD as h_1, lookahead as h_2, for different lookahead depths. The *Good1* column presents the number of nodes where LA^* saved the computation of h_2 while the h_2 column presents the number of nodes where h_2 was computed. Roughly 28% of nodes were *Good1* and since t_2 was the most dominant time cost, most of this saving is reflected in the timing results. The best results are achieved for lookahead 8, with a runtime of 0.527 compared to A^* with WMD.

The final columns show the results of RLA^* , with the values of τ, p_h, t_2 calibrated for each lookahead depth using a small subset of problem instances. The *Good2* column counts the number of times that RLA^* decided to bypass the h_2 computation. Observe that RLA^* outperforms LA^*, which in turn outperforms A^*, for most lookahead depths. The lowest time with RLA^* (0.371 of A^* with WMD) was obtained for lookahead 10. That is achieved as the more expensive h_2 heuristic is computed less often, reducing its effective computational overhead, with some adverse effect in the number of expanded nodes. Although LA^* expanded fewer nodes, RLA^* performed much fewer h_2 computations as can be seen in the table, resulting in decreased overall runtimes.

5.2 Planning domains

We implemented LA^* and RLA^* on top of the Fast Downward planning system [Helmert, 2006], and experimented with two state of the art heuristics: the admissible landmarks heuristic h_{LA} (used as h_1) [Karpas and Domshlak, 2009], and the landmark cut heuristic h_{LMCUT} [Helmert and Domshlak, 2009] (used as h_2). On average, h_{LMCUT} computation is 8.36 times more expensive than h_{LA} computation. We did not implement HBP in the planning domains as the heuristics we use are not consistent and in general the operators are not invertible. We also did not implement OB, as the cost of OPEN operations in planning is trivial compared to the cost of heuristic evaluations.

We experimented with all planning domains without conditional effects and derived predicates (which the heuristics we used do not support) from previous IPCs. We compare the performance of LA^* and RLA^* to that of A^* using each

	Problems Solved						Planning Time (seconds)						GOOD	
Domain	h_{LA}	lmcut	max	selmax	LA^*	RLA^*	h_{LA}	lmcut	max	selmax	LA^*	RLA^*	LA^*	RLA^*
airport	25	24	26	25	**29**	**29**	**0.29**	0.57	0.5	0.33	0.38	0.38	0.48	0.67
barman-opt11	**4**	0	0	0	0	3	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
blocks	26	27	27	27	**28**	**28**	1.0	**0.65**	0.73	0.81	0.67	0.67	0.19	0.21
depot	**7**	6	5	5	6	6	**2.27**	2.69	3.17	3.14	2.73	2.75	0.06	0.06
driverlog	10	**12**	**12**	**12**	**12**	**12**	2.65	**0.29**	0.33	0.36	0.3	0.31	0.09	0.09
elevators-opt08	12	**18**	17	17	17	17	14.14	4.21	4.84	4.85	**3.56**	3.64	0.27	0.27
elevators-opt11	10	**14**	**14**	**14**	**14**	**14**	26.97	8.03	9.28	9.28	**6.64**	6.78	0.28	0.28
floortile-opt11	2	**6**	**6**	**6**	**6**	**6**	8.52	**0.44**	0.6	0.58	0.5	0.52	0.02	0.02
freecell	**54**	10	36	51	41	41	**0.16**	7.34	0.22	0.24	0.18	0.18	0.86	0.86
grid	**2**	**2**	1	**2**	**2**	**2**	**0.1**	0.16	0.18	0.34	0.15	0.15	0.17	0.17
gripper	**7**	6	6	6	6	6	**0.84**	1.53	2.24	2.2	1.78	1.25	0.01	0.4
logistics00	**20**	17	16	**20**	19	19	**0.23**	0.57	0.68	0.27	0.47	0.47	0.51	0.51
logistics98	3	**6**	**6**	**6**	**6**	**6**	0.72	**0.1**	0.1	0.11	**0.1**	**0.1**	0.07	0.07
miconic	**141**	140	140	**141**	**141**	**141**	**0.13**	0.55	0.58	0.57	0.16	0.16	0.87	0.88
mprime	16	20	20	20	**21**	20	1.27	0.5	0.51	0.5	**0.44**	0.45	0.25	0.25
mystery	13	**15**	**15**	**15**	**15**	**15**	0.71	**0.35**	0.38	0.43	0.36	0.37	0.3	0.3
nomystery-opt11	**18**	14	16	**18**	**18**	**18**	**0.18**	1.29	0.58	0.25	0.33	0.33	0.72	0.72
openstacks-opt08	15	**16**	14	15	**16**	**16**	2.88	**1.68**	3.89	3.03	2.62	2.64	0.44	0.45
openstacks-opt11	10	**11**	9	10	**11**	**11**	13.59	**6.96**	19.8	14.44	12.03	12.06	0.43	0.43
parcprinter-08	14	**18**	**18**	**18**	**18**	**18**	0.92	**0.36**	0.37	0.38	0.37	0.37	0.17	0.26
parcprinter-opt11	10	**13**	**13**	**13**	**13**	**13**	2.24	**0.56**	0.6	0.61	0.58	0.59	0.14	0.17
parking-opt11	1	1	1	**3**	2	2	9.74	22.13	17.85	7.11	**6.33**	6.43	0.64	0.64
pathways	4	**5**	**5**	**5**	**5**	**5**	0.5	**0.1**	**0.1**	**0.1**	**0.1**	**0.1**	0.1	0.12
pegsol-08	**27**	**27**	**27**	**27**	**27**	**27**	1.01	**0.84**	1.2	1.1	1.06	0.95	0.04	0.42
pegsol-opt11	**17**	**17**	**17**	**17**	**17**	**17**	4.91	**3.63**	5.85	5.15	4.87	4.22	0.04	0.38
pipesworld-notankage	**16**	15	15	**16**	15	15	**0.5**	1.48	1.12	0.85	0.9	0.91	0.42	0.42
pipesworld-tankage	**11**	8	9	9	9	9	**0.36**	2.24	1.02	0.47	0.69	0.71	0.62	0.62
psr-small	**49**	48	48	**49**	48	48	**0.15**	0.2	0.21	0.19	0.19	0.18	0.17	0.49
rovers	6	**7**	**7**	**7**	**7**	**7**	0.74	**0.41**	0.45	0.45	0.41	0.42	0.47	0.47
scanalyzer-08	6	**13**	**13**	**13**	**13**	**13**	0.37	**0.25**	0.27	0.27	0.26	0.26	0.06	0.06
scanalyzer-opt11	3	**10**	**10**	**10**	**10**	**10**	**0.59**	0.64	0.75	0.73	0.67	0.68	0.05	0.05
sokoban-opt08	23	25	25	24	26	**27**	3.94	1.76	2.19	2.96	1.9	**1.32**	0.04	0.4
sokoban-opt11	**19**	**19**	**19**	18	**19**	**19**	7.26	2.83	3.66	5.19	3.1	**2.02**	0.03	0.46
storage	14	**15**	14	14	**15**	**15**	**0.36**	0.44	0.49	0.45	0.44	0.42	0.21	0.28
tidybot-opt11	**14**	12	12	12	12	12	**3.03**	16.32	17.55	9.35	15.67	15.02	0.11	0.18
tpp	**6**	**6**	**6**	**6**	**6**	**6**	0.39	**0.22**	0.23	0.23	0.22	0.22	0.32	0.4
transport-opt08	**11**	**11**	**11**	**11**	**11**	**11**	1.45	**1.24**	1.41	1.54	1.25	1.26	0.04	0.04
transport-opt11	**6**	**6**	**6**	**6**	**6**	**6**	12.46	**8.5**	10.38	11.13	8.56	8.61	0.0	0.0
trucks	7	**9**	**9**	**9**	**9**	**9**	4.49	**1.34**	1.52	1.44	1.41	1.42	0.07	0.07
visitall-opt11	12	10	**13**	12	**13**	**13**	0.2	0.34	0.19	**0.18**	0.18	0.18	0.38	0.38
woodworking-opt08	12	**16**	**16**	**16**	**16**	**16**	1.08	0.71	0.75	0.75	**0.66**	0.67	0.56	0.56
woodworking-opt11	7	**11**	**11**	**11**	**11**	**11**	5.7	2.86	3.15	3.01	**2.55**	2.58	0.52	0.52
zenotravel	8	**11**	**11**	**11**	**11**	**11**	0.38	**0.14**	0.14	0.14	0.14	0.14	0.17	0.19
OVERALL	698	697	722	747	747	**750**	1.18	0.98	0.98	0.89	0.79	**0.77**	0.27	0.34

Table 4: Planning Domains — Number of Problems Solved, Total Planning Time, and Fraction of Good Nodes

of the heuristics individually, as well as to their max-based combination, and their combination using selective-max (Sel-MAX) [Domshlak *et al.*, 2012]. The search was limited to 6GB memory, and 5 minutes of CPU time on a single core of an Intel E8400 CPU with 64-bit Linux OS.

When applying RLA^* in planning domains we evaluate rule (7) at every state. This rule involves two unknown quantities: $\frac{t_2}{t_1}$, the ratio between heuristic computations times, and p_h, the probability that h_2 is helpful. Estimating $\frac{t_2}{t_1}$ is quite easy — we simply use the average computation times of both heuristics, which we measure as search progresses.

Estimating p_h is not as simple. While it is possible to empirically determine the best value for p_h, as done for the weighted 15 puzzle, this does not fit the paradigm of domain-independent planning. Furthermore, planning domains are very different from each other, and even problem instances in the same domain are of varying size, and thus getting a single value for p_h which works well for many problems is difficult. Instead, we vary our estimate of p_h adaptively during search. To understand this estimate, first note that if n is a node at which h_2 was helpful, then we computed h_2 for n, but did not expand n. Thus, we can use the number of states for which we computed h_2 that were not yet expanded (denoted by A), divided by the number of states for which we computed h_2 (denoted by B), as an approximation of p_h. However, $\frac{A}{B}$ is not likely to be a stable estimate at the beginning of the search, as A and B are both small numbers. To overcome this problem, we "imagine" we have observed k examples, which give us an estimate of $p_h = p_{init}$, and use a weighted average between these k examples, and the observed examples — that is, we estimate p_h by $(\frac{A}{B} \cdot B + p_{init} \cdot k)/(B + k)$. In our empirical evaluation, we used $k = 1000$ and $p_{init} = 0.5$.

Table 4 depicts the experimental results. The leftmost part of the table shows the number of solved problems in each domain. As the table demonstrates, RLA^* solves the most problems, and LA^* solves the same number of problems as Sel-MAX. Thus, both LA^* and RLA^* are state-of-the-art in cost-optimal planning. Looking more closely at the results, note that Sel-MAX solves 10 more problems than LA^* and RLA^* in the freecell domain. Freecell is one of only three domains in which h_{LA} is more informed than h_{LMCUT} (the other two are nomystery-opt11 and visitall-opt11), violating the basic assumptions behind LA^* that h_2 is more informed than h_1. If we ignore these domains, both LA^* and RLA^* solve more problems than Sel-MAX.

The middle part of the Table 4 shows the geometric mean of planning time in each domain, over the commonly solved problems (i.e., those that were solved by all 6 methods). RLA^* is the fastest overall, with LA^* second. It is important to note that both LA^* and RLA^* are very robust, and even in cases where they are not the best they are never too far from the best. For example, consider the *miconic* domain. Here, h_{LA} is very informative and thus the variant that only computed h_{LA} is the best choice (but a bad choice overall). Observe that both LA^* and RLA^* saved 86% of h_{LMCUT} computations, and were very close to the best algorithm in this extreme case. In contrast, the other algorithms that consider

	Expanded	Generated
h_{LA}	183,320,267	1,184,443,684
lmcut	23,797,219	114,315,382
A^*_{MAX}	22,774,804	108,132,460
selmax	54,557,689	193,980,693
LA^*	22,790,804	108,201,244
RLA^*	25,742,262	110,935,698

Table 5: Total Number of Expanded and Generated States

Alg.	Generated	HBP1	HBP2	OB	Bad	Good	time
$h1 = \Delta X$, $h2 = \Delta Y$, Depth = 26.66							
A*	1,085,156	0	0	0	0	0	**415**
A*+HBP	1,085,156	216,689	346,335	0	0	0	417
LA*	1,085,157	0	0	734,713	37,750	312,694	417
LA*+HBP	1,085,157	140,746	342,178	589,893	37,725	115,361	416
$h1$ = Manhattan distance, $h2$ = 7-8 PDB, Depth 52.52							
A*	43,741	0	0	0	0	0	34.7
A*+HBP	43,804	30,136	1,285	0	0	0	33.6
LA*	43,743	0	0	42,679	47	1,017	34.2
LA*+HBP	43,813	7,669	1,278	42,271	21	243	**33.3**

Table 6: Results on the 15 puzzle

both heuristics (max and Sel-MAX) performed very poorly here (more than three times slower).

The rightmost part of Table 4 shows the average fraction of nodes for which LA^* and RLA^* did not evaluate the more expensive heuristic, h_{LMCUT}, over the problems solved by both these methods. This is shown in the *good* columns. Our first observation is that this fraction varies between different domains, indicating why LA^* works well in some domains, but not in others. Additionally, we can see that in domains where there is a difference in this number between LA^* and RLA^*, RLA^* usually performs better in terms of time. This indicates that when RLA^* decides to skip the computation of the expensive heuristic, it is usually the right decision.

Finally, Table 5 shows the total number of expanded and generated states over all commonly solved problems. LA^* is indeed as informative as A^*_{MAX} (the small difference is caused by tie-breaking), while RLA^* is a little less informed and expands slightly more nodes. However, RLA^* is much more informative than its "intelligent" competitor - Sel-MAX, as these are the only two algorithms in our set which selectively omit some heuristic computations. RLA^* generated almost half of the nodes compared to Sel-MAX, suggesting that its decisions are better.

5.3 Limitations of LA^*: 15 puzzle example

Some domains and heuristic settings will not achieve time speedup with LA^*. An example is the regular, unweighed 15-puzzle. Results for A^*_{MAX} and LA^* with and without HBP on the 15-puzzle are reported in Table 6. HBP_1 (HBP_2) count the number of nodes where HBP pruned the need to compute h_1 (resp. h_2). OB is the number of nodes where OB was helpful. *Bad* is the number of nodes that went through two OPEN cycles. Finally, *Good* is the number of nodes where computation of h_2 was saved due to LA^*.

In the first experiment, Manhattan distance (MD) was divided into two heuristics: Δx and Δy used as h_1 and h_2. Results are averaged over 100 random instances with average solution depth of 26.66. As seen from the first two lines, HBP when applied on top of A^*_{MAX} saved about 36% of the heuristic evaluations. Next are results for LA^* and LA^*+HBP. Many nodes are pruned by HBP, or OB. The number of *good* nodes dropped from 28% (Line 3) to as little as 11% when HBP was applied. Timing results (in ms) show that all variants performed equally. The reason is that the time overhead of the Δx and Δy heuristics is very small so the saving on these 28% or 11% of nodes was not significant to outweigh the HBP overhead of handling the upper and lower bounds.

The next experiment is with MD as h_1 and a variant of the additive 7-8 PDBs [Korf and Felner, 2002], as h_2. Here we can observe an interesting phenomenon. For LA^*, most nodes were caught by either HBP (when applicable) or by OB. Only 4% of the nodes were *good* nodes. The reason is that the 7-8 PDB heuristic always *dominates* MD and is always the maximum among the two. Thus, 7-8 PDB was needed at early stages (e.g. by OB) and MD itself almost never caused nodes to be added to OPEN and remain there until the goal was found.

These results indicate that on such domains, LA^* has limited merit. Due to uniform operator cost and the heuristics being consistent and simple to compute, very little space is left for improvement with *good* nodes. We thus conclude that LA^* is likely to be effective when there is significant difference between t_1 and t_2, and/or operators that are not bidirectional and/or with non-uniform costs, allowing for more *good* nodes and significant time saving.

6 Conclusion

We discussed two schemes for decreasing heuristic evaluation times. LA^* is very simple to implement and is as informative as A^*_{MAX}. LA^* can significantly speed up the search, especially if t_2 dominates the other time costs, as seen in weighted 15 puzzle and planning domains. Rational LA^* allows additional cuts in h_2 evaluations, at the expense of being less informed than A^*_{MAX}. However, due to a rational tradeoff, this allows for an additional speedup, and Rational LA^* achieves the best overall performance in our domains.

RLA^* is simpler to implement than its direct competitor, Sel-MAX, but its decision can be more informed. When RLA^* has to decide whether to compute h_2 for some node n, it already *knows* that $f_1(n) \leq C^*$. By contrast, although Sel-MAX uses a much more complicated decision rule, it makes its decision when n is first generated, and does not know whether h_1 will be informative enough to prune n. Rational LA^* outperforms Sel-MAX in our planning experiments.

RLA^* and its analysis can be seen as an instance of the rational meta-reasoning framework [Russell and Wefald, 1991]. While this framework is very general, it is extremely hard to apply in practice. Recent work exists on meta-reasoning in DFS algorithms for CSP) [Tolpin and Shimony, 2011] and in Monte-Carlo tree search [Hay *et al.*, 2012]. This paper applies these methods successfully to a variant of A^*. There are numerous other ways to use rational meta-reasoning to improve A^*, starting from generalizing RLA^* to handle more than two heuristics, to using the meta-level to control decisions in other variants of A^*. All these potential extensions provide fruitful ground for future work.

7 Acknowledgments

The research was supported by the Israeli Science Foundation (ISF) under grant #305/09 to Ariel Felner and Eyal Shimony and by Lynne and William Frankel Center for Computer Science.

References

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

[Domshlak *et al.*, 2012] Carmel Domshlak, Erez Karpas, and Shaul Markovitch. Online speedup learning for optimal planning. *JAIR*, 44:709–755, 2012.

[Felner *et al.*, 2011] A. Felner, U. Zahavi, R. Holte, J. Schaeffer, N. Sturtevant, and Z. Zhang. Inconsistent heuristics in theory and practice. *Artificial Intelligence*, 175(9-10):1570–1603, 2011.

[Felner *et al.*, 2012] A. Felner, M. Goldenberg, G. Sharon, R. Stern, T. Beja, N. R. Sturtevant, J. Schaeffer, and Holte R. Partial-expansion a* with selective node generation. In *AAAI*, pages 471–477, 2012.

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

[Hay *et al.*, 2012] Nicholas Hay, Stuart Russell, David Tolpin, and Solomon Eyal Shimony. Selecting computations: Theory and applications. In Nando de Freitas and Kevin P. Murphy, editors, *UAI*, pages 346–355. AUAI Press, 2012.

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

[Helmert, 2006] Malte Helmert. The Fast Downward planning system. *JAIR*, 26:191–246, 2006.

[Karpas and Domshlak, 2009] Erez Karpas and Carmel Domshlak. Cost-optimal planning with landmarks. In *IJCAI*, pages 1728–1733, 2009.

[Korf and Felner, 2002] R. E. Korf and A. Felner. Disjoint pattern database heuristics. *Artificial Intelligence*, 134(1-2):9–22, 2002.

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

[Russell and Wefald, 1991] Stuart Russell and Eric Wefald. Principles of metereasoning. *Artificial Intelligence*, 49:361–395, 1991.

[Stern *et al.*, 2010] Roni Stern, Tamar Kulberis, Ariel Felner, and Robert Holte. Using lookaheads with optimal best-first search. In *AAAI*, pages 185–190, 2010.

[Sun *et al.*, 2009] X. Sun, W. Yeoh, P. Chen, and S. Koenig. Simple optimization techniques for A*-based search. In *AAMAS*, pages 931–936, 2009.

[Thayer and Ruml, 2011] Jordan T. Thayer and Wheeler Ruml. Bounded suboptimal search: A direct approach using inadmissible estimates. In *Proceedings of the Twenty-second International Joint Conference on Artificial Intelligence (IJCAI-11)*, 2011.

[Tolpin and Shimony, 2011] David Tolpin and Solomon Eyal Shimony. Rational deployment of CSP heuristics. In Toby Walsh, editor, *IJCAI*, pages 680–686. IJCAI/AAAI, 2011.

[Zhang and Bacchus, 2012] Lei Zhang and Fahiem Bacchus. Maxsat heuristics for cost optimal planning. In *AAAI*, 2012.

Heuristics for Planning under Partial Observability with Sensing Actions

Guy Shani and **Ronen Brafman** and **Shlomi Maliah**
Ben Gurion University

Erez Karpas
Technion

Abstract

In planning under partial observability with sensing actions (PPOS) problems, the solution progresses from one sensing action to another, until sufficient information is gathered and the goal can be reached. In between sensing actions, one can use classical planning to derive the path to the next sensing action. We suggest an online algorithm that repeatedly selects the next sensing action to execute, and plans to achieve it in a classical setting. Our algorithm avoids the difficulty in representing and updating a belief space. Our heuristic uses landmarks, and we explain how landmarks can be computed over a relaxation of the PPOS problem. We compare our Heuristic Contingent Planner (HCP) to state-of-the-art, translation-based online contingent planners, and show how it solves many problems much faster than previous approaches.

Introduction

Agents acting under partial observability must acquire information about the true state of the world using sensing actions to achieve their goals. Such problems can be modeled using contingent planning, where action effects may be conditioned on some unknown world features. Contingent planning is difficult because the plan must branch given different sensor values, resulting in potentially large plan trees.

Currently, there are two popular techniques for generating contingent plans. Both approaches can be used in an offline scenario – where the entire plan tree is generated offline, or in an online scenario — where only the branch corresponding to the online observations is generated incrementally.

The first technique builds on a compilation method for conformant planning (Palacios and Geffner, 2009), translating the contingent planning problem, into a classical planning problem that "reasons" about the agent's state of knowledge. Solutions to this classical problem correspond to complete or partial contingent plans (Albore, Palacios, and Geffner, 2009; Shani and Brafman, 2011; Bonet and Geffner, 2011a). A major difficulty with this approach is the size of the generated classical planning problems and the time required for their solution.

The second technique directly searches in belief space (Bonet and Geffner, 2000; Hoffmann and Brafman, 2005; To, Son, and Pontelli, 2011). As the effect of sensing actions is non-deterministic, the search can be modeled as an AND/OR tree, and various search heuristics can be applied. The major difficulty with this approach is to compactly represent and efficiently update the belief state, and currently no known method works well in all domains (To, Pontelli, and Son, 2011). Generating good heuristics estimates in belief space is also a challenge.

This paper describes a new, online method that overcomes the weaknesses of both methods. We solve a sequence of simple classical planning problems defined on the original state space. Thus, we avoid reasoning about the agent's knowledge, and maintaining and updating its belief state. Our method currently focuses on contingent planning domains in which sensing actions do not alter the state of the world — a condition which is true in all current contingent planning benchmarks. In such domains, every branch of a contingent plan contains a sequence of sensing actions. Between every two sensing actions, there is a sequence of non-sensing actions. This is well known, and most online contingent planners leverage it, whether explicitly or implicitly.

The key insight in our method, though, is that the sequence of non-sensing actions in between every two sensing actions can be viewed as a classical planning problem defined over the original (in fact, even slightly reduced) state space of the problem, rather than the belief space. Thus, if at each stage we are given the next sensing action to perform by an oracle, we can quickly plan to achieve its preconditions using a suitable classical planner.

Given the agent's current state, there may be multiple reachable sensing actions. Thus, we require a method for selecting among them — an oracle. Indeed, technically, our main contribution is the development of a method for approximating the value of information of every achievable sensing action using an estimate of the number of landmarks that can be achieved following the execution of each sensing action. This requires, in turn, adjusting landmark generation techniques to the setting of partial observability and sensing.

Our Heuristic Contingent Planner (HCP) first computes a set of landmarks over a specially designed relaxation of the planning problem. Next, at each step, we compute the set of reachable sensing actions and estimate their value of information. We use then a classical planner to generate a plan from the current state to the seemingly best sensing action. We repeat this process until we reach the goal.

We empirically evaluate HCP on a set of contingent planning benchmarks. Our experiments show that HCP is much faster than the state-of-the-art contingent planners SDR (Brafman and Shani, 2012b) and CLG (Albore, Palacios, and Geffner, 2009), on domains with structured belief spaces, where certain conditions that we explain hold, and good landmarks can be discovered.

Partially Observable Contingent Planning

Partially observable contingent planning problems are characterized by uncertainty about the initial state of the world, partial observability, and the existence of sensing actions. Actions may be non-deterministic, but much of the literature focuses on deterministic actions, and in this paper we will assume deterministic actions, too.

Problem Definition

A contingent planning problem is a quadruple: $\pi = \langle P, A, \varphi_I, G\rangle$. P is a set of propositions, A is a set of actions, φ_I is a formula over P that describes the set of possible initial states, and $G \subset P$ is the goal propositions. We often abuse notation, treating a set of literals as a conjunction of the literals in the set, as well as an assignment of the propositions in it. For example, $\{p, \neg q\}$ will also be treated as $p \wedge \neg q$ and as an assignment of *true* to p and *false* to q.

A state of the world, s, assigns a truth value to all elements of P. A *belief-state* is a set of possible states, and the initial belief state, $b_I = \{s : s \models \varphi_I\}$ defines the set of states that are possible initially. An action $a \in A$ is a three-tuple, $\{$*pre*(a),*effects*(a),*obs*$(a)\}$. *pre*(a) is a set of literals denoting the action's preconditions. *effects*(a) is a set of pairs (c, e) denoting conditional effects, where c is a set (conjunction) of literals and e is a single literal. Finally, *obs*(a) is a set of propositions, denoting those propositions whose value is observed when a is executed. We assume that a is well defined, that is, if $(c, e) \in$ *effects*(a) then $c \wedge$*pre*(a) is consistent, and that if both $(c, e), (c', e') \in$ *effects*(a) and $s \models c \wedge c'$ for some state s then $e \wedge e'$ is consistent. In current benchmark problems, either the set *effects* or the set *obs* are empty. That is, actions either alter the state of the world but provide no information, or they are pure sensing actions that do not alter the state of the world, but this is not a mandatory limitation.

We use $a(s)$ to denote the state that is obtained when a is executed in state s. If s does not satisfy all literals in *pre*(a), then $a(s)$ is undefined. Otherwise, $a(s)$ assigns to each proposition p the same value as s, unless there exists a pair $(c, e) \in$ *effects*(a) such that $s \models c$ and e assigns p a different value than s. Observations affect the agent's belief state. We assume throughout that all observations are deterministic and accurate, and reflect the state of the world *prior* to the execution of the action. It is possible to have observation reflect the post-action state, at the price of slightly more complicated notation. Thus, if $p \in$*obs*(a) then following the execution of a, the agent will observe p if p holds now, and otherwise it will observe $\neg p$. Thus, if s is the true state of the world, and b is the current belief state of the agent, then $b_{a,o}$, is the belief state following the execution of a and observing o. The new belief state corresponds to the progression through a of all states in b where o is observed.

A complete plan for a contingent planning problem can be described as an tree $\tau = (N, E)$. The nodes, N, are labeled with actions, and the edges, E, are labeled with observations. A node labeled by an action with no observations has a single child, and the edge leading to it is labeled by the null observation *true*. Otherwise, each node has one child for each possible observation value. The edge leading to this child is labeled by the corresponding observation. τ is a *solution plan* (or *plan*) for π if $\tau(s) \models G$ for every $s \in b_I$.

We illustrate these ideas using a 4×4 Wumpus domain (Albore, Palacios, and Geffner, 2009), which will serve as our running example. Figure 1 illustrates this domain, where an agent is located on a 4×4 grid. The agent can move in all four directions, and if moving into a wall, it remains in place. The agent initially is in the low-left corner and must reach the top-right corner. There are two monsters called Wumpuses hidden along the grid diagonal, the agent knows that Wumpus 1 can be at location 3,2 or 2,3, and Wumpus 2 can be at location 4,3 or 3,4. Thus the possible states are: $\{wat(3,2) \wedge wat(4,3), wat(3,2) \wedge wat(3,4), wat(2,3) \wedge wat(4,3), or wat(2,3) \wedge wat(3,4)\}$. The stench of a Wumpus carries to all adjacent locations, and the agent can observe the stench in order to deduce the whereabouts of the Wumpuses.

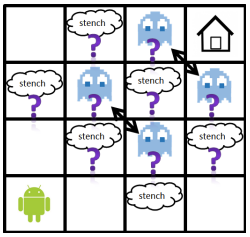


Figure 1: The 4×4 Wumpus domain

Landmarks in Classical Planning

In general, a landmark Φ for a classical planning task $\Pi = \langle P, A, I, G\rangle$ is a logical formula over the facts P, which must be satisfied at some state along every solution of Π (Hoffmann, Porteous, and Sebastia, 2004). As in most literature dealing with landmarks, we will restrict our attention to landmarks that are simple disjunctions or conjunctions over facts. Each planning task has some trivial landmark, consisting of all goal facts, and all facts in the initial state. In the Wumpus problem described above, e.g., these trivial landmarks are the goal, *at-4-4*, and the initial state fact *at-1-1*.

Another related notion is that of orderings between landmarks. Several types of orderings have been defined (Hoffmann, Porteous, and Sebastia, 2004). We use the most general type of ordering (that is, the one with the weakest condition), called reasonable-ordering, denoted $\Phi \rightarrow \Psi$. Intuitively, this implies that a "reasonable" plan will achieve Φ before Ψ; the exact definitions of the different orderings are not important in this context.

Although it is PSPACE-hard even to check whether a given fact is a landmark or not, there are several efficient

algorithms which return a set of landmarks and orderings (Hoffmann, Porteous, and Sebastia, 2004; Zhu and Givan, 2003; Richter and Westphal, 2010; Keyder, Richter, and Helmert, 2010). These algorithms all exploit the principle that, if some fact $p \in P$ is a landmark that is not true in the initial state, and all actions which achieve p have some fact $q \in P$ as a precondition, then q is also a landmark. Furthermore, q must be achieved before p, and so we also have the ordering $q \rightarrow p$. If there is no common precondition, we can still find a set of facts which occur in the preconditions of all actions, and use them as a disjunctive landmark. For example, the landmark *at-4-4* has two achievers, going up from *at-4-3*, and going right from *at-3-4*. These actions do not have a common precondition, but we can still infer that *at-4-3* $\vee$ *at-3-4* is a landmark, which is ordered before *at-4-4*.

Originally, landmarks were used as subgoals (Hoffmann, Porteous, and Sebastia, 2004), guiding a base planner inside a control loop. At each iteration of the loop, the set of landmarks which could be achieved — that is, the landmarks which do not have an unachieved landmark that is ordered before them — are passed to the base planner as a disjunctive goal. The base planner then returns a plan which achieves one or more of these landmarks, the landmarks which have been achieved are marked, and the control loop continues planning from the state which was reached. Although this technique has been shown to speed up planning significantly, it can not guarantee the optimality of the solution found, nor even the completeness of the overall planning process. This is because the base planner might reach a state which is a dead-end for the planning task Π, and the control loop does not backtrack.

More recently, the number of landmarks which are yet to be achieved was used as a path-dependent heuristic in the LAMA planner (Richter and Westphal, 2010), winner in the sequential satisfying track in IPC-2008 and IPC-2011. This is an inadmissible heuristic estimate, because an action might achieve more than one landmark. If optimal planning is of interest, it is possible to derive an admissible landmarks-based heuristic by performing an action cost partitioning over the landmarks (Karpas and Domshlak, 2009).

Landmark Detection under Partial Observability

Adapting landmark detection to PPOS, we must be able to handle uncertainty and sensing. In principle, we could run classical landmark detection in belief space or in the pseudo belief space of the translation approach, both of which are potentially exponentially larger. To be useful, however, landmark detection must be efficient and informative.

One of the main goals and contributions of this work is to use classical techniques on the original state space when possible, while planning in a partially observable domain. To reduce the detection cost, as in classical planning, we run landmark detection on a relaxed problem. We use the popular delete-relaxation approach in which negative effects of actions are ignored.

In addition to ignoring delete-effects, we augment the domain with additional artificial actions that help us deal with uncertainty and observation. As the value of some propositions is unknowns, there is typically no path from the initial state to a goal state that does not involve observations and deductions from these observations concerning the set of possible states. The actions that we add help us bridge this gap to a certain extent, without requiring us to represent the agent's belief state.

Our relaxation is motivated by features of current benchmarks, as we explain below. In that sense, it is not general and may fail to produce meaningful landmarks when certain conditions do not apply. That being said, we demonstrate how our relatively simple relaxation is highly useful in a large set of benchmarks. Like other landmark detection algorithm, our approach is sound, in that every landmark that is detected is indeed a landmark of the real domain, but incomplete, in that many landmarks may go undetected. We skip the soundness proof due to the lack of space.

Conditional Effects Removal

We break down each action into actions with unary effects, by replacing each action $a \in A$ with conditional effects $\{(c_i, e_i) : i = 1..k\}$, with k actions, such that $pre(a_{(c_i,e_i)}) = pre(a) \wedge c_i$ and $\mathit{effects}(a_{(c_i,e_i)}) = \mathit{effects}(a) \wedge e_i$.

In general such a compilation is unsound, because several conditions may apply together at a given state, while we allow only one condition per action. As we are computing a heuristic, though, and since we will apply these actions in a delete-relaxation, forward-search manner, we will allow the application of multiple actions concurrently (Blum and Furst, 1997). Thus, all conditions that apply will be executed simultaneously at the same phase of the forward search.

Reasoning over the Initial Uncertainty

In many cases the uncertainty over some sets of propositions is encoded into the initial belief state, and remains unchanged throughout the execution of the solution (Bonet and Geffner, 2011b). In the Wumpus domain, e.g., the monsters remain at a given location and never move, and the stench around them is unchanged. For such sets of propositions, we can create reasoning actions that, upon detecting the value of some propositions of that set, reason about the rest.

The detection of these sets of propositions is simple, as propositions whose value is constant do not appear in the effects of any action. We find clauses of the initial belief formula that contain only constant propositions, and use these to create reasoning actions. We assume here that the initial state formula is expressed as a conjunction of either simple disjunctions of literals, or xor (so called "one-of") clauses.

For each such clause c containing only unchanged propositions we create a set of "reasoning" actions A_c, as follows:

- If $c = \bigvee_{i=1..k} l_i$, then $A_c = \{a_{l_i}\}_{i=1}^{k}$, with $pre(a_{l_i}) = \bigwedge_{j=1..k, j\neq i} \neg l_j$, and $\mathit{effects}(a_{l_i}) = l_i$.
- If $c = \oplus_{i=1..k} l_i$, then $A_c = \{a_{l_i}\}_{i=1}^{k}$, with $pre(a_{l_i}) = l_i$, and $\mathit{effects}(a_{l_i}) = \bigwedge_{j=1..k, j\neq i} \neg l_j$.

Joining Immediate Reasoning and Observations

In a PPOS we can split the propositions into 3 disjoint sets — propositions whose value is always known (e.g. the location of the agent in the Wumpus problem), propositions whose value may be unknown, but for which there is an observation action (e.g., the stench in cells near the Wumpus lair), and propositions whose value can not be observed (e.g., the location of the lair of the Wumpus). Note that we must have some way to reason about the values of the variables which can not be observed, as otherwise we can just ignore them.

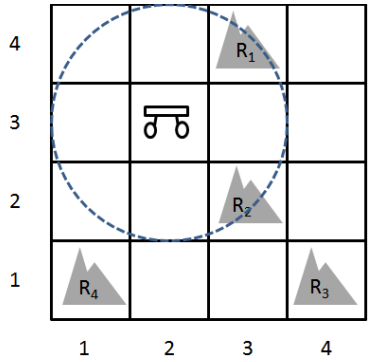


Figure 2: A Mars Rover rock sampling example, with a 4×4 grid and 4 rocks. The rover is at location $2, 3$, and the dotted line shows the range of its mineral sensor. Hence, only rocks 1 and 2 are within range at this location.

In many cases it is natural to define the value of the observable propositions through conditional effects over unobservable propositions. Consider for example the Mars Rover rock sample (Smith and Simmons, 2004; Brafman and Shani, 2012b), where the rover must sample rocks containing a desirable mineral in a grid (Figure 2). The rocks locations are known, but the agent must sense for the mineral using a long range sensor. The sensor reports the existence of the mineral in some rock within its range.

A natural formalization of this domain may have an action *activate-sensor-at-2-3* with preconditions *at-2-3* and conditional effects *good-*$rock_1$ $\rightarrow$ *good-rocks-in-range* and *good-*$rock_2$ $\rightarrow$ *good-rocks-in-range*, and an additional action *observe-rocks-in-range* which observes the *good-rocks-in-range* proposition which signifies the sensor's output. While these are separate actions, used together they allow us to reason about which rocks contain the good mineral. In this case, *at-x-y* is always known, *good-rocks-in-range* is unknown but directly observable, and *good-*$rock_i$ is unknown and not directly observable. We now explain how these two actions — *activate-sensor* and *observe-good-rocks-in-range* can be joined to allow us to observe and reason about certain propositions together.

Let a be an action that contains a set of conditional effects of the form (c_i, e) where c_i is unknown and unobservable and e is observable, and there is no other action that affects the value of e that is not mutually exclusive, i.e., that can be executed at the same state as a. In our example the *activate-sensor-at-x-y* actions are the only actions that affect *good-rocks-in-range*, and each such action requires the agent to be at a different location. For all such actions we consider the reverse of the conditions — $(e, \bigvee_i c_i)$.

We now create a new action $a \circ a_{obs}$ where a_{obs} is an observation action over e. The preconditions of the new action will be the conjunction of the preconditions of a and a_{obs}. The observation of $a \circ a_{obs}$ is e, and effects $\mathit{effects}(a \circ a_{obs}) = \mathit{effects}_u(a) \wedge \bigvee_i c_i$, where $\mathit{effects}_u(a)$ are the unconditional effects of a. When there is more than a single c_i, we remove the non-determinism, by creating a set of actions $a_i \circ a_{obs}$ whose preconditions $pre(a \circ a_{obs}) \wedge \bigwedge_{j \neq i} \neg c_j$, and whose effect is $\mathit{effects}(a_i \circ a_{obs}) = \mathit{effects}_u(a) \wedge c_i$. In the example above we have two such joined actions with preconditions *at-2-3* $\wedge$ $\neg$*good-*$rock_j$, observation *good-rocks-in-range* and effect *good-*$rock_i$ where $i = 1$ and $j = 2$ or vice versa.

Note that this method is not general, in that there might be a sequence of unobservable propositions $p_1, ..., p_k$ and a set of actions a_i with conditional effects (p_i, p_{i+1}), and an observation action only for p_k. Our translation only allows reasoning over the value of p_{k-1}, and not about any of its predecessors. That being said, the only benchmark that exhibits this behavior is *localize*, while many other (e.g. *medpks, rock-sample*) contain the behavior we exploit above.

Finding Fact Landmarks

We now run a landmark detection algorithm on our relaxed domain that contains the actions created above, as well as all the "regular" actions that have no conditional effects, and operates only on the known propositions. In our experiments we used back-chaining using the possible first achievers (Richter, 2010). However, one can use any landmark detection algorithm on the relaxed problem described above. The only change in the landmark detection algorithms is with respect to the sensing actions, where we assume that they optimistically provide any required value of the observed proposition.

The Heuristic Contingent Planner Algorithm

We can now present our online contingent planning solver. The planner progresses by repeatedly identifying a reachable observation action that heuristically provides valuable information towards the achievement of the goal. The planner then plans in a classical setting to execute the observation action, assuming all unknown propositions to have a negative value. The plan is executed, followed by the observation action. Now, the process repeats with the additional information that was provided by the observation action. This process is repeated until the goal can be achieved without executing any additional observation actions. Algorithm 1 shows this high level algorithm.

Algorithm 2 shows the process for selecting the next sensing action. It estimates the myopic value of information of the observation action, i.e., how much value will be achieved from executing the action, ignoring future observations.

We first compute the set of achievable literals in the relaxed problem. Then, we see which observation actions can

Algorithm 1 Heuristic Contingent Planner

Input: π — a PPOS problem
1: $\pi_{relaxed} \leftarrow$ The relaxation of π as explained above
2: $L \leftarrow$ The set of landmarks for $\pi_{relaxed}$
3: $s \leftarrow$ All known literals at the initial state
4: **while** There is no classical plan that achieves G from s with no observation actions **do**
5: $a_{obs} \leftarrow ChooseNextSensingAction(\pi_{relaxed}, L, s)$
6: $P_{classic} \leftarrow Plan(s, pre(a_{obs}))$
7: $s' \leftarrow Execute(P_{classic}, s)$
8: $Execute(a_{obs})$ and observe literal l
9: $s \leftarrow s' \cup \{l\}$
10: **end while**
11: $P_{classic} \leftarrow Plan(s, G)$
12: $Execute(P_{classic}, s)$

be executed that sense the value of some unknown proposition. These are the candidate actions to be returned by the algorithm. To choose the heuristically best observation action, we analyze the value of observing p, by assuming that we have observed p, and computing which literals now become reachable. Then, we assume that we have observed $\neg p$ and compute again which literals become available.

Our policy for returning the heuristically best action first looks at the number of satisfied landmarks following the observation. Given multiple observation actions that satisfy the same number of landmarks, we break ties by looking at the sum of the number of literals and new observation actions that become achievable following the execution of the observation action. Finally, we break ties again in favor of the action which requires the minimal number of actions (in the relaxed domain) to execute.

Algorithm 2 Choosing the Next Sensing Action

Input: $\pi_{relaxed}$ — a relaxed PPOS problem, L a set of landmarks, s the set of currently known literals
1: $s' \leftarrow$ the set of achievable literals given $\pi_{relaxed}$ and s
2: $\Omega \leftarrow \{a : a \in A, pre(a) \in s', obs(a) \neq \phi, obs(a) \notin s'\}$
3: **for each** action $a \in \Omega$ **do**
4: $p \leftarrow obs(a)$, $s'_+ \leftarrow s' \cup \{p\}$, $s'_- \leftarrow s' \cup \{\neg p\}$
5: $s''_+ \leftarrow$ the set of achievable literals given $\pi_{relaxed}$ and s'_+
6: $s''_- \leftarrow$ the set of achievable literals given $\pi_{relaxed}$ and s'_-
7: $score_{landmarks}(a) \leftarrow$ the number of landmarks satisfied in s''_+ and s''_-, but not in s'
8: $score_{literals}(a) \leftarrow$ the number of literals achievable in s''_+ and s''_-, but not in s'
9: $score_{obs}(a) \leftarrow$ the number of sensing actions achievable in s''_+ and s''_-, but not in s'
10: $score_{cost}(a) \leftarrow$ the number of actions required from s before a can be executed in $\pi_{relaxed}$
11: $score(a) \leftarrow \langle score_{landmarks}(a), score_{literals}(a) + score_{obs}(a), score_{cost}(a)\rangle$
12: **end for**
13: **return** $argmax_{a\in\Omega} score(a)$

Empirical Evaluation

We now compare HCP to state-of-the-art online contingent planners, CLG (Albore, Palacios, and Geffner, 2009), SDR (Brafman and Shani, 2012b), MPSR (Brafman and Shani, 2012a), and K-Planner (Bonet and Geffner, 2011b) on various benchmarks. The experiments were conducted on a Windows Server 2008 machine with 24 2.66GHz cores (although each experiment uses only a single core) and 32GB of RAM. The underlying classical planner is FF (Hoffmann and Nebel, 2001).

Table 1 shows that HCP is much faster than all other planners, except for the K-Planner. The plan quality (number of actions) of HCP is also typically quite good. The only domain that could not be solved by HCP is *localize*, because it does not conform to our assumptions concerning reasoning about the hidden propositions.

K-Planner is very fast, but it can only run on domains where the hidden propositions remain constant throughout the execution. Thus, it is unsuitable for solving many of the benchmarks. Except for K-Planner, HCP is by far the fastest contingent planner, and in many cases improves runtime by more than an order of magnitude.

Related Work

Bonet and Geffner (2000) first proposed using heuristic search in belief space. Since then, several heuristics for belief states were proposed. Bryce, Kambhampati, and Smith (2006) argue that belief state heuristics typically aggregate distance estimates from the individual states which the belief state describes to the goal. Taking the maximum distance corresponds to assuming positive interaction (that is, the plan for reaching the goal from s1 also helps to reach the goal from s2), while summarizing the distances corresponds to assuming independence between these plans (that is, the plans for s1 and s2 neither help or interfere with each other).

Contingent-FF (CFF) (Hoffmann and Brafman, 2005) uses delete-relaxation, assuming *generous execution semantics*, which ignores actions whose preconditions are not satisfied during execution. They construct a relaxed conformant plan by building a variant of the relaxed planning graph, accounting for which facts are known at each layer. CFF represents and reasons about knowledge through a logic formula over the history. CFF also identifies *observation goals* — observations needed by a later action.

The DNF planner (To, Pontelli, and Son, 2009) uses a heuristic based on the number of satisfied goals, the cardinality of the belief state, and a measure called the *square distance* of the belief state to the goal, which is the sum of the number of unsatisfied goals in each individual state in the belief state. Goals are trivially also landmarks, and thus the number of unsatisfied goals can be seen as a special case of the number of unsatisfied landmarks, with a trivial landmark discovery method.

HCP is related to these planners as it also searches in belief space, although it doesn't explicitly represent and reasons about it. On the other hand, our use of landmark detection is far beyond any heuristics currently applied by other belief search planners.

Surprisingly, online planners that plan repeatedly once new knowledge has been acquired are less popular in this line of research. This reduces the scalability of these methods, because the complete plan tree can be exponential in the

Table 1: Comparing the performance of state of the art contingent planners. Blank cells represent problems that the planners were unable to solve. CSU denotes models that CLG can solve but cannot simulate execution for.

	HCP		MPSR		SDR		CLG		K-Planner	
Name	Actions	Time	Actions	Time	Actions	Time	Actions	Time	Actions	Time
cloghuge	55.48 (0.304)	**5.9** (0.0452)			61.17 (0.44)	117.13 (4.19)	**51.76** (0.33)	8.25 (0.08)		
ebtcs-70	42.32 (0.6712)	**1.12** (0.0188)	44.5 (0.7)	22.4 (0.3)	**35.52** (0.75)	3.18 (0.07)	**36.52** (0.86)	73.96 (0.14)		
elog7	**20** (0.076)	**0.32** (0.0016)	23.5 (0.1)	1.4 (0.1)	21.76 (0.07)	0.85 (0.01)	**20.12** (0.05)	1.4 (0.08)		
CB-9-5	**324** (2.24)	158.9 (1.76)			392.16 (2.81)	505.48 (8.82)	CSU		358.08 (15.8)	**94.18** (3.31)
CB-9-7	**425** (2.2636)	373 (2.28)			487.04 (2.95)	833.52 (15.82)	CSU		458.36 (14.64)	**116.63** (3.24)
doors13	**96.68** (0.52)	**30** (0.1296)	197.92 (1.2)	105.5 (2.1)	120.8 (0.93)	158.54 (2.01)	105.48 (0.89)	330.73 (0.21)	109.72 (4.76)	37.96 (1.72)
doors15	**137.9** (1.1052)	**52.6** (0.6228)	262.2 (1.9)	190 (3.3)	143.24 (1.36)	268.16 (3.78)			150.88 (4.7)	**55.24** (2)
doors17	**170** (1.456)	91 (0.708)	368.25 (3.4)	335.3 (5.3)	188 (1.64)	416.88 (6.16)			188.8 (5.79)	**79.24** (2.62)
localize17			59.8 (0.9)	**230.4** (7.7)	**45** (0.86)	928.56 (33.2)	CSU			
unix3	**40.48** (1.156)	**1.77** (0.0448)	69.7 (1.7)	5.2 (0.1)	56.32 (1.72)	5.47 (0.18)	51.32 (0.97)	18.56 (0.05)	**45.48** (4.59)	16.87 (1.56)
unix4	94.56 (1.88)	**20.21** (0.2868)	158.6 (4.3)	30.4 (1.1)	151.72 (4.12)	35.22 (0.94)	**90.8** (2.12)	189.41 (0.6)	87.04 (8.54)	38.81 (3.53)
Wumpus15	**65.08** (1.1052)	9.57 (0.11248)	**65** (1.6)	126.6 (3.1)	120.14 (2.4)	324.32 (7.14)	101.12 (0.67)	330.54 (0.25)	107.64 (4.6)	**7.17** (0.6)
Wumpus20	90 (1.3984)	34 (0.3396)	**71.6** (1.2)	261.1 (7)	173.21 (3.4)	773.01 (20.78)	155.32 (0.95)	1432 (0.47)	151.52 (6.29)	**16.03** (1)
RockSample 8-12	**105.76** (0.3984)	**6.3** (0.0496)			127.24 (0.68)	113.4 (0.79)				
RockSample 8-14	**135** (0.52)	**9** (0.038)			142.08 (0.8)	146.75 (1.19)				

worst case in the number of propositions. Thus, even fully specifying the plan tree may be impossible.

A second popular approach to contingent planning is the compilation-based approach (Albore, Palacios, and Geffner, 2009; Shani and Brafman, 2011; Brafman and Shani, 2012a; Bonet and Geffner, 2011a). These methods translate a PPOS into a classical planning problem, directly reasoning about the possible hidden state. Such methods add new "knowledge" propositions and modify actions so that the state space is transformed into a belief space, essentially allowing a classical planner to plan in belief space. This reduction allows leveraging advances in classical planning, such as recent, powerful heuristic generation methods. In this setting online approaches were developed, that plan only for branches of the plan tree that can be reached given the true hidden state at runtime.

Some of these methods make simplifying assumptions concerning the problem structure, as we do. For example CLG (Albore, Palacios, and Geffner, 2009) assumes limited uncertainty, modeled formally through the notion of *conformant-width*. Specifically, CLG cannot solve problems of width larger than 1, like Rock Sample. K-Planner Bonet and Geffner (2011a) has an even more strict assumption — it can solve only problems where the unknown proposition remain constant through the plan execution. This makes reasoning about the hidden state much easier, and our reasoning actions are inspired by this observation.

Conclusion and Future Work

We introduced a new approach to contingent planning, relying on heuristics computed over a relaxation of the domain description, without maintaining a belief state explicitly, or translating the problem into classical planning, which are the two popular approaches to contingent planning under partial observability.

Our planner, HCP, leverages certain properties of many benchmarks in order to avoid the explicit maintenance of belief states. Domains which do not conform to these properties, cannot be solved by HCP.

In the future we intend to generalize our assumptions concerning these properties, and formally identify domains where HCP works well, and provide a proof for its soundness and completeness for these domains. We will also look into less strict assumptions that may help us to solve domains which are currently impossible for HCP.

References

Albore, A.; Palacios, H.; and Geffner, H. 2009. A translation-based approach to contingent planning. In *IJCAI*, 1623–1628.

Blum, A., and Furst, M. L. 1997. Fast planning through planning graph analysis. *Artif. Intell.* 90(1-2):281–300.

Bonet, B., and Geffner, H. 2000. Planning with incomplete information as heuristic search in belief space. In *Proc. AIPS'00*, 52–61.

Bonet, B., and Geffner, H. 2011a. Planning under partial observability by classical replanning: Theory and experiments. In *IJCAI'11*.

Bonet, B., and Geffner, H. 2011b. Planning under partial observability by classical replanning: Theory and experiments. In *IJCAI*, 1936–1941.

Brafman, R. I., and Shani, G. 2012a. A multi-path compilation approach to contingent planning. In *AAAI*.

Brafman, R. I., and Shani, G. 2012b. Replanning in domains with partial information and sensing actions. *Journal of Artificial Intelligence Research (JAIR)* 45:565–600.

Bryce, D.; Kambhampati, S.; and Smith, D. E. 2006. Planning graph heuristics for belief space search. *JOURNAL OF AI RESEARCH* 26:35–99.

Hoffmann, J., and Brafman, R. I. 2005. Contingent planning via heuristic forward search with implicit belief states. In *ICAPS*, 71–80.

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

Hoffmann, J.; Porteous, J.; and Sebastia, L. 2004. Ordered landmarks in planning. *Journal of AI Research* 22:215–278.

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

Keyder, E.; Richter, S.; and Helmert, M. 2010. Sound and complete landmarks for and/or graphs. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, 335–340.

Palacios, H., and Geffner, H. 2009. Compiling uncertainty away in conformant planning problems with bounded width. *JAIR* 35:623–675.

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

Richter, S. 2010. *Landmark-Based Heuristics and Search Control for Automated Planning*. Ph.D. Dissertation, Griffith University.

Shani, G., and Brafman, R. I. 2011. Replanning in domains with partial information and sensing actions. In *IJCAI*, 2021–2026.

Smith, T., and Simmons, R. 2004. Heuristic search value iteration for POMDPs. In *UAI 2004*.

To, S. T.; Pontelli, E.; and Son, T. C. 2009. A conformant planner with explicit disjunctive representation of belief states. In *ICAPS*.

To, S. T.; Pontelli, E.; and Son, T. C. 2011. On the effectiveness of cnf and dnf representations in contingent planning. In *IJCAI*, 2033–2038.

To, S. T.; Son, T. C.; and Pontelli, E. 2011. Contingent planning as and/or forward search with disjunctive representation. In *ICAPS*.

Zhu, L., and Givan, R. 2003. Landmark extraction via planning graph propagation. In *ICAPS 2003 Doctoral Consortium*, 156–160.

Abstraction Pathologies In Markov Decision Processes

Manel Tagorti and **Bruno Scherrer** and **Olivier Buffet**
INRIA
Nancy, France
{manel.tagorti,bruno.scherrer,olivier.buffet}@inria.fr

Jörg Hoffmann
Saarland University
Saarbrücken, Germany
hoffmann@cs.uni-saarland.de

Abstract

Abstraction is a common method to compute lower bounds in classical planning, imposing an equivalence relation on the state space and deriving the lower bound from the quotient system. It is a trivial and well-known fact that refined abstractions can only improve the lower bound. Thus, when we embarked on applying the same technique in the probabilistic setting, our firm belief was to find the same behavior there. We were wrong. Indeed, there are cases where *every* direct refinement step (splitting one equivalence class into two) yields strictly *worse* bounds. We give a comprehensive account of the issues involved, for two wide-spread methods to define and use abstract MDPs.

Introduction

In classical planning, an abstraction is a mapping α from the set of all states into a smaller set of abstract states ($[s]_\alpha$ denoting the set of states t where $\alpha(t) = \alpha(s)$). This is used to derive a lower bound $h^\alpha(s)$ on the remaining cost of any state s. Namely, α induces an abstract planning problem over the abstract state space: (i) an abstract state $[s]_\alpha$ is a goal iff it contains at least one original goal state, and (ii) a transition from $[s]_\alpha$ to $[s']_\alpha$ exists iff there exist $t \in [s]_\alpha$ and $t' \in [s']_\alpha$ so that the original state space has a transition from t to t'. The abstract planning problem is a relaxed version of the original one –or, conversely, the original problem is more constrained– so that, given a state s, the cost of an abstract plan starting from $[s]_\alpha$ is at most equal to the cost of a plan starting from s, and can thus be used as a lower bound $h^\alpha(s)$. Prominent examples of this method are pattern databases (Edelkamp 2001; Haslum et al. 2007) and merge-and-shrink abstractions (Helmert, Haslum, and Hoffmann 2007; Nissim, Hoffmann, and Helmert 2011; Katz, Hoffmann, and Helmert 2012).

A refinement of α is an abstraction α' resulting from α by splitting some of the block states, i.e., for all s we have $[s]_{\alpha'} \subseteq [s]_\alpha$. It is commonplace that refinements can only improve the heuristic, i.e., $h^\alpha(s) \leq h^{\alpha'}(s)$ for all s: If we split block states apart, then the solution paths can only get longer and thus more costly (assuming non-negative costs as usual). Indeed, this observation is so simple that, to our knowledge, no-one yet bothered to state it in a paper and its first appearance is in Malte Helmert's 2010 lecture slides.[1]

Our initial agenda in this research was to solve MDPs using heuristic search methods like LRTDP (Bonet and Geffner 2003), our focus being to compute heuristic functions by starting with a coarse abstraction and iteratively refining it. Against the background described above, as a warm-up exercise we embarked on proving that the essential property of refinements – they can only improve the heuristic – is true in that setting as well. Which was all fine, except we ended up proving the opposite.

First things first, to conduct this kind of research for MDPs one needs to first define what the "quotient system" and the corresponding heuristic functions are. This is non-trivial because, in difference to the classical case where all we are interested in is which states can transition to which other states in principle, now we need to define transition *probabilities* for the abstract MDP. To illustrate, if action a maps s into a state from $[s']_\alpha$ with probability 0.9, but maps $t \in [s]_\alpha$ into a state from $[s']_\alpha$ with probability 0.1, which probability should we assign for a to map $[s]_\alpha$ into $[s']_\alpha$?

A simple answer is to assign the average probability over all states in $[s]_\alpha$. In the example, this would yield the transition probability 0.5. A main issue with this approach is that the resulting heuristic function – the value function of the abstract MDP – is neither a lower bound nor an upper bound on the value function of the original MDP. Givan et al. (2000) fix this by basically considering intervals of transition probabilities (in the example, the interval $[0.1, 0.9]$). They derive a lower bound on the original value function (expected reward) by selecting the probabilities pessimistically, and derive an upper bound of the original value function by selecting the probabilities optimistically.

We first proved that, for the average-probability approach, there exist an MDP, a state s, an abstraction α, and refined α' so that the error of $h^{\alpha'}(s)$ relative to the original value function is larger than that of $h^\alpha(s)$. This may be duly understood as an accident pertaining to the sketchy nature of this approach; indeed, as we show, the original MDP does not have to be non-deterministic to provoke this kind of behavior. However, we next proceeded to prove the same property for Givan et al.'s approach. Worse, even: We constructed an MDP, a state s, and α so that *all* direct refinements α' (resulting from α by splitting a single block state) result in

[1] http://www.informatik.uni-freiburg.de/~ki/teaching/ss10/aip/aip10.pdf

strictly worse bounds. More naturally than for the average-probability approach, non-determinism is required for this, i.e., if the original MDP is deterministic then every refinement results in better bounds.

In the remainder of the paper, we first give the necessary background definitions. We then explain our results in the order described above.

Markov Decision Processes and Abstractions

Markov Decision Processes (MDPs) are a general framework for modeling decision-making problems in stochastic environments. We define an MDP as follows

Definition 1. *A Markov Decision Process is given by a (finite) state space S, a (finite) action space A, a reward function $R : S \times A \to \mathbb{R}$ and transition probabilities $p(s, a, s')$ which determine the probabilities of transition when performing action a in state s.*

A deterministic policy $\pi : S \to A$ assigns an action to each state, and we are looking for the optimal policy π^*, i.e., one that maximizes for all $s \in S$ the return

$$V^{\pi}(s) = E_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t R_t | s_0 = s\right],$$

where γ is the discount rate taken in $[0, 1)$.

The related value V^* is the unique solution of the equation $V^* = TV^*$ where T is the Bellman operator defined as :

$$\forall s,\ V^*(s) = TV^*(s) = \max_a R(s, a) + \gamma \sum_{s' \in S} p(s, a, s')V^*(s').$$

Determining π^* may be infeasible in MDPs with large state spaces. In this paper we simplify the problem by employing state abstractions. Abstractions provide a smaller representation M_α of the original MDP M. The image of M under an abstraction α is defined on a state space S_α smaller than S. Indeed S_α is a partition of S consisting of block states $[s]_\alpha$. We assign to each block-state, given an action $a \in A$, the reward $R([s]_\alpha, a)$ and the transition probabilities $p([s]_\alpha, a, [s_1]_\alpha)$ for all $[s_1]_\alpha \in S_\alpha$. The useful abstractions are the ones that induce a small error of approximation when considering M_α instead of M. We would like to identify such abstractions by comparing a given abstraction to its (direct) refinement, in terms of approximation error.

Abstractions' Refinement

Definition 2. *Let α and α' be two abstractions of an MDP M. We say that α' is* finer *than α, denoted $\alpha' \succeq \alpha$, iff for any states $s, s' \in S$, $\alpha'(s) = \alpha'(s')$ implies $\alpha(s) = \alpha(s')$. We can also say that α is* coarser *than α', denoted $\alpha \preceq \alpha'$.*

We have α' a direct refinement *of α if there exist states $s_1, s_2 \in S$ such that $[s_1]_\alpha = [s_2]_\alpha$, $[s_1]_{\alpha'} \neq [s_2]_{\alpha'}$, $[s_1]_\alpha = [s_1]_{\alpha'} \cup [s_2]_{\alpha'}$, and $\alpha'(s) = \alpha(s)$ for all $s \in S \setminus [s_1]_\alpha$.*

We show in what follows that the error induced by α' may in some cases be higher than the one induced by α. But before that we have first to specify the parameters (rewards and transition probabilities) related to the abstract representation M_α of M.

Average MDPs

We consider in this section the abstraction α that connects an MDP M to its average representation. In other words, α maps an MDP M defined on S to an average MDP M_α, defined on S_α, and admits as parameters, the averages of rewards and transitions over all states contained in the block state $[s]_\alpha$ ((Ortner 2011)), i.e., we have for all $a \in A$:

$$R([s]_\alpha, a) = \frac{1}{|[s]_\alpha|} \sum_{s_1 \in [s]_\alpha} R(s_1, a) \text{ and}$$

$$p([s]_\alpha, a, [s']_\alpha) = \frac{1}{|[s]_\alpha|} \sum_{s_1 \in [s]_\alpha} \sum_{s_2 \in [s']_\alpha} p(s_1, a, s_2).$$

We denote here by $|[s]_\alpha|$ the cardinal of all states in $[s]_\alpha$.

We choose as approximation error E_α, the average error, estimated by taking the average difference between the true value V taken in a state s and the value V_α of its corresponding block state $[s]_\alpha$,

$$E_\alpha = \frac{1}{|S|} \sum_{s \in S} |V_\alpha([s]_\alpha) - V(s)|.$$

This approximation error may increase when we refine the abstraction α. We illustrate in Figure 1 an example in which the number of states where the (local) error increases-after a refinement-is greater than the one where the (local) error decreases, causing the increase of the average error.

Proposition 1. *There exists a deterministic MDP M, an abstraction α and a refinement α' of α such that $E_\alpha < E_{\alpha'}$.*

Proof. Consider the MDP M in Figure 1 with a single action $\{a\}$ and a discount rate $\gamma = 1$ (the result would not change for $\gamma < 1$ but close enough to 1). The states in $\{2, ..., k\}$ ($k > 2$) are similar: they admit the same rewards ($R = 0$) and have the same dynamics : they reach the neighboring state with probability one. The states in $\{k + 1, ..., n\}$ are also similar: they all reach the goal G with probability one and they admit a non-negative reward R_2. The state 1 admits a reward $R(1) = R_1 \ll R_2$ and reaches the goal with probability one.

Taking $V(G) = 0$, we then have $V(1) = R_1$, and $V(i) = R_2$ forall i in $\{2, ..., n\}$.

Based on those similarities one can construct a perfectly suitable abstraction α_0 ($E_{\alpha_0} = 0$) which aggregates similar states in the same block, i.e., in our case $\alpha_0 : S \to 1, \{2, ..., k\}, \{k + 1, ..., n\}, G$.

Consider now the abstraction $\alpha_1 : S \to \{1, ..., k\}, \{k + 1, ..., n\}, G$, where the states 1 and $\{2, ..., k\}$ are in the same block (Figure 1). The similarity will be then broken resulting in a strictly positive error E_α. The related values are

$$V_{\alpha_1}(\{k + 1, ..., n\}) = \frac{(n - k)R_2}{n - k} = R_2 \text{ and}$$

$$V_{\alpha_1}(\{1, ..., k\}) = \frac{R_1}{k} + \frac{k - 2}{k} V_{\alpha_1}(\{1, ..., k\}) + \frac{1}{k} V_{\alpha_1}(\{k + 1, ..., n\}) = \frac{R_1 + R_2}{2} \neq R_2.$$

And the induced error is

$$E_{\alpha_1} = \frac{1}{n}\sum_{i=1}^{k} |V(i) - V_{\alpha_1}(\{1,...,k\})| \sim k\frac{R_2}{2n} \text{ for } R_1 \ll R_2.$$

Let $\alpha_2 : S \to \{1,...,n\}, G$ be the abstraction which aggregates states $1, 2, ..., n$ together (Figure 1), the value $V_{\alpha_2}(\{1,...,n\})$ is equal to

$$V_{\alpha_2}(\{1,...,n\}) = \frac{R_1 + (n-k)R_2}{n} + \frac{k-1}{n}V_{\alpha_2}(\{1,...,n\}) \sim R_2 \text{ for } k \ll n$$

and hence

$$E_{\alpha_2} \sim \frac{1}{n}|V(1) - V_{\alpha_2}(\{1,...,n\})| \sim \frac{1}{n}R_2.$$

We can see that the error E_{α_1} is strictly larger than E_{α_2} for a number of states k strictly greater than 2. □

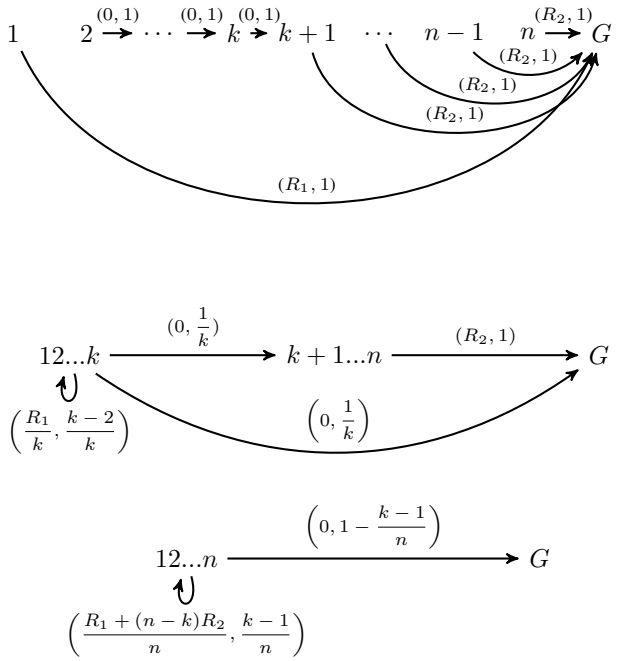


Figure 1: From the top to the bottom: The MDP M, the abstraction α_1 and the abstraction α_2 (α_0 is not shown). The parentheses denote the reward (on the left) and the transition probabilities (on the right).

Bounded-parameter MDPs

Rather than approaching V^* with fixed values V_α, it is possible to establish bounds on the MDP's parameters so as to get bounds on V^*. So we considered the abstraction α that associates, for each (action) a in A, each block state $[s]_\alpha$ with the intervals' parameters ((Givan, Leach, and Dean 2000)):

$$R^{\updownarrow}([s]_\alpha, a) = [\min_{s\in[s]_\alpha} R(s,a), \max_{s\in[s]_\alpha} R(s,a)]$$

$$p^{\updownarrow}([s]_\alpha, a, [s']_\alpha) = [\min_{s_1\in[s]_\alpha} \sum_{s_2\in[s']_\alpha} p(s_1,a,s_2), \max_{s_1\in[s]_\alpha} \sum_{s_2\in[s']_\alpha} p(s_1,a,s_2)].$$

We get hence what we call a *a bounded parameter Markov Decision process* (BMDP) more commonly defined as:

Definition 3. *A Bounded parameter Markov Decision Process is given by a (finite) state space Σ, a (finite) action space A, an interval of rewards $R^{\updownarrow}(\sigma, a)$, $\forall\sigma \in \Sigma, a \in A$, and an interval of transition probabilities $p^{\updownarrow}(\sigma, a, \sigma')$, $\forall\sigma, \sigma' \in \Sigma, a \in A$.*

Each state of the BMDP has a range of values depending on R and p. We can assign to each state a closed interval of value functions $[V^-(\sigma), V^+(\sigma)]$ where V^- corresponds to the pessimistic bound and V^+ to the optimistic one.

Our abstract representation M_α is then a BMDP where the state space Σ coincides with state space S_α. We would like to estimate the value bounds V_α^+ ans V_α^- related to M_α. Givan *et al.* have proposed an algorithm, the interval value iteration IVI, to do so. To make explicit the two Bellman's operators hidden behind this algorithm (T^+ and T^-), we first need to introduce the notion of *compatibility* with respect to an abstraction.

Definition 4. *An MDP N is compatible with M with respect to the abstraction α if for all s, and for all a, $R_N(s,a) \in R_M^{\updownarrow}([s]_\alpha, a)$ and for all s, s', a, $\sum_{s_1\in[s']_\alpha} p_N(s,a,s_1) \in p_M^{\updownarrow}([s]_\alpha, a, [s']_\alpha)$.*
The set of all MDPs compatible with M with respect to α is denoted $[M]_\alpha$.

Figure 2 gives an example of an MDP N compatible with an MDP M with respect to the abstraction $\alpha : 1, 2, 3 \to \{1, 2\}, 3$.

We claim that the Bellman operators T^+ and T^- used to estimate V_α^+ and V_α^- can be written in this way, for all s:

$$T^+[V](s) = \max_{a\in A} \max_{N\in[M]_\alpha} R_N(s,a) + \gamma \sum_{s'\in S} p_N(s,a,s')V(s'),$$

$$T^-[V](s) = \max_{a\in A} \min_{N\in[M]_\alpha} R_N(s,a) + \gamma \sum_{s'\in S} p_N(s,a,s')V(s').$$

By taking iteratively the $\max$ (resp the $\min$) on the set of compatible MDPs and by choosing the optimal policy, we can see that those two operators converge to fixed values. Indeed T^+ and T^- are γ-contracting so, by the Banach fixed point Theorem, they admit unique fixed points V_α^+ and V_α^- (which are constant per block), i.e., $T^+V_\alpha^+ = V_\alpha^+$ and $T^-V_\alpha^- = V_\alpha^-$. There exists an optimistic MDP M^{opt} (respectively a pessimistic MDP M^{pes}) and a corresponding optimal (optimistic) policy π^{opt} (respectively an optimal (pessimistic) policy π^{pes}) for which the value V_α^+ (resp V_α^-) is reached. The MDPs M^{opt} and M^{pes} belong to $[M]_\alpha$.

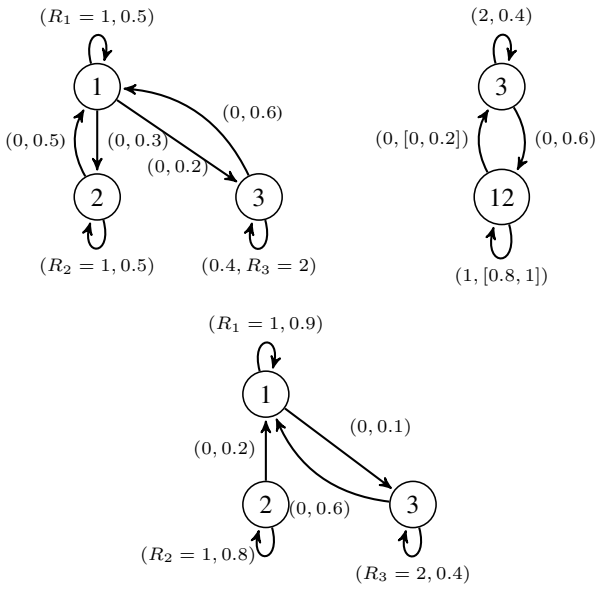


Figure 2: From left to right, the MDP M, the BMDP M_α, $\alpha : 1, 2, 3 \to \{1, 2\}, 3$ and the MDP N compatible with M with respect to α.The parenthesis denote the reward (on the left) and the transitions (on the right).

Before proceeding to the next step –the choice of the abstraction that would ensure better bounds–, we first show that these values are indeed bounds on V^*. This is precisely what is stated in this following theorem.

Theorem 1. *(Givan, Leach, and Dean 1997) For any MDP M and abstraction α of the states of M, bounds on the BMDP M_α apply also to M, i.e., $\forall s \in S$, $V^*(s) \in [V_\alpha^-([s]_\alpha), V_\alpha^+([s]_\alpha)]$.*

Proof. [2] The proof is done using Value Iteration. With initial value $V^0 = V_\alpha^+$ we have, for all s,

$$V^1(s) = \max_a R_M(s, a) + \gamma \sum_{s' \in S} p_M(s, a, s') V_\alpha^+([s']_\alpha).$$

Since V_α^+ is a fixed point of T^+ and $M \in [M]_\alpha$, we can see that

$$\begin{aligned} V^1(s) &\leq \max_{a \in A} \max_{N \in [M]_\alpha} R_N(s, a) + \gamma \sum_{s' \in S} p_N(s, a, s') V_\alpha^+([s']_\alpha) \\ &\leq V_\alpha^+([s]_\alpha) = V^0(s). \end{aligned}$$

The Bellman operator T is monotone, so that we have $T^n V^1 \leq T^n V^0 = V^0$. By taking the limit, we get $V^*(s) \leq V_\alpha^+([s]_\alpha)$. A similar proof may be applied to the pessimistic bound. □

[2] *This proof does not appear in (Givan, Leach, and Dean 1997) but it has been established thanks to a correspondence with R. Givan.*

Value Bounds using finer abstractions

As previously done for the average model, we will compare the two errors G_α and $G_{\alpha'}$ induced by the BMDP M_α and its direct refinement $M_{\alpha'}$, where, for an abstraction α, G_α measures the gap between the two bounds in each state, for all s:

$$G_\alpha(s) = \left(V_\alpha^+([s]_\alpha) - V_\alpha^-([s]_\alpha)\right).$$

The proposition below states a sufficient condition under which the finer abstraction α' yields better value bounds than α and therefore decreases the error G_α. Indeed, if the set of MDPs compatible with M with respect to the finer abstraction are also compatible with M with respect to the coarser abstraction, then we get the inclusion of the value function intervals.

Proposition 2. *Given an MDP M and two abstractions α and α' s.t. $\alpha' \succeq \alpha$ and $[M]_{\alpha'} \subseteq [M]_\alpha$ then, for all $s \in S$,*

$$[V_{\alpha'}^-([s]_{\alpha'}), V_{\alpha'}^+([s]_{\alpha'})] \subseteq [V_\alpha^-([s]_\alpha), V_\alpha^+([s]_\alpha)].$$

Proof. Value Bounds computation can be considered as a case of an alternating two players stochastic game where one choose the optimal policy while the other choose the optimal MDP. For the upper bound, we can then invert the two $\max$ in the Bellman operators' expressions ((Givan, Leach, and Dean 2000)). This would not change the final result, so we have

$$V_\alpha^+([s]_\alpha) = \max_{N \in [M]_\alpha} V_N^*(s).$$

For the lower bound, more detailed arguments have been established in (Bertsekas and Tsitsiklis 1996) to set the inversion of the $\max$ and the $\min$ terms, so we get:

$$V_\alpha^-([s]_\alpha) = \min_{N \in [M]_\alpha} V_N^*(s).$$

Given that $[M]_{\alpha'}$ is included in $[M]_\alpha$, then by taking the $\max$, (respectively the $\min$) the result follows. □

We will study next two cases : the deterministic case, where the sufficient condition is always satisfied, and the stochastic case, where it is not necessarily satisfied, for which we will give an example.

Deterministic case: probabilities in $\{0, 1\}$

Corollary 1. *If we consider a deterministic MDP and two abstractions α and α', where α' is a direct refinement of α, then for all $s \in S$,*

$$[V_{\alpha'}^-([s]_{\alpha'}), V_{\alpha'}^+([s]_{\alpha'})] \subseteq [V_\alpha^-([s]_\alpha), V_\alpha^+([s]_\alpha)].$$

Proof. Let us consider an MDP N compatible with M under α'. We will show that the sufficient condition of Proposition 2 is satisfied: N is also compatible with M under α. Note that having N compatible with M with respect to an abstraction α, according to Definition 4, is equivalent to having the inclusion of the parameter intervals i.e., for all s, and for all a, $R_N^\updownarrow([s]_\alpha, a) \subseteq R_M^\updownarrow([s]_\alpha, a)$ and for all s, s', a, $p_N^\updownarrow([s]_\alpha, a, [s']_\alpha) \subseteq p_M^\updownarrow([s]_\alpha, a, [s']_\alpha)$. Let $[s_1]_\alpha$ be the state block in S_α that we split into two blocks $[s_2]_{\alpha'}$ and $[s_3]_{\alpha'}$ (i.e., we have $[s_2]_{\alpha'} \cup [s_3]_{\alpha'} = [s_1]_\alpha$). It is easy to

check the inclusion of reward intervals under α as for each action a:

$$\min_{s\in[s_1]_\alpha} R_N(s,a) = \min(\min_{s\in[s_2]_{\alpha'}} R_N(s,a), \min_{s\in[s_3]_{\alpha'}} R_N(s,a)).$$

Also, using the α' compatibility hypothesis we have

$$\min_{s\in[s_i]_{\alpha'}} R_N(s,a) \geq \min_{s\in[s_i]_{\alpha'}} R_M(s,a) \text{ for } i \text{ in } \{2,3\}$$

then

$$\min_{s\in[s_1]_\alpha} R_N(s,a) \geq \min_{s\in[s_1]_\alpha} R_M(s,a).$$

By reasoning in a similar way for the upper bound, we get the inclusion of the reward intervals. The same arguments may be employed to state the inclusion of outgoing transition probabilities $p([s_1]_\alpha, a, [s_4]_\alpha)$ for all s_4 in S and a in A. So we will mainly focus on ingoing transition probabilities $p([s_5]_\alpha, a, [s_1]_\alpha)$ for all s_5 in S and we will show that

$$\min p_N([s_5]_\alpha, a, [s_1]_\alpha) \geq \min p_M([s_5]_\alpha, a, [s_1]_\alpha). \quad (1)$$

Since we work in a deterministic environment, probabilities can only take the values 0 or 1. For the case where $\min p_N([s_5]_\alpha, a, [s_1]_\alpha) = 1$, inequality (1) is always verified. Now, $\min p_N([s_5]_\alpha, a, [s_1]_\alpha) = 0$ implies that there exist a state s' in $[s_5]_\alpha$ and a block state $[s_6]_\alpha$ distinct from $[s_1]_\alpha$ such that $p_N(s', a, [s_6]_\alpha) = 1$. So we can find a state s'' in $[s_5]_\alpha$ such that $p_M(s'', a, [s_6]_\alpha) = 1$ as N is compatible with M under α'. We then have Equation (1) and by Proposition 2 the final result follows.

□

Stochastic case We would like to have an equivalent of Proposition 2 for the stochastic case but it turns out that in general the sufficient condition is no more fulfilled. In fact when we move to stochastic transitions, the sufficient condition in Proposition 2 becomes harder to satisfy. The successful MDP M has to verify specific conditions that depend on the choice of the abstractions α and α'. In other words, given an MDP M and an abstraction α there does not always exists a refined abstraction α' such that the sufficient condition is satisfied. Figure 3 shows a model of MDP in which we can not find the appropriate direct refinement α'.Three states which admit the same rewards $R(1) = R(2) = R(3) = 1$ and behave identically in the block $\{1,2,3\}$ (the same probabilities in regards to the block $\{1,2,3\}$, $p(1,\{1,2,3\}) = p(2,\{1,2,3\}) = p(3,\{1,2,3\}) = 0.7$). States 4 and 5 are goal states ($V(4) = V(5) = 0$). We can find an MDP N compatible with M with respect to the abstraction α' ($S \to \{1,2\}, 3, 4, 5$) but not compatible with M with respect to the abstraction α ($S \to \{1,2\}, 3, 4, 5$). It suffices to take an MDP N whose parameter intervals are included in M's parameter intervals under α' but admits real intervals rather than fixed real values under α.

The condition stated in Proposition 2 is a sufficient condition, we can not *a priori* conclude about the existence of a refinement that would make the error of approximation decrease. Nevertheless, we have identified models of MDPs where every direct refinement strictly increases the error.

Proposition 3. *There exists an MDP M and an abstraction α such that, for any direct refinement α' of α, we have: (1) for all s in S, $G_{\alpha'}(s) \geq G_\alpha(s)$, and (2) there exists s in S where $G_{\alpha'}(s) > G_\alpha(s)$.*

Proof. The MDP shown in Figure 3 is an example of such a model. The value $V_\alpha(\{1,2,3\})$ related to the block $\{1,2,3\}$ is a perfect heuristic for the states $1, 2$ and 3. In fact we can see (for $\gamma = 1$), using Theorem 1, that the optimistic and pessimistic bounds coincide: $V^-_\alpha(\{1,2,3\}) = V^+_\alpha(\{1,2,3\}) = V(1) = V(2) = V(3) = 3.33$. If we refine the block $\{1,2,3\}$ by splitting the block $\{1,2,3\}$ into $\{1,2\}$ and $\{3\}$, then the error of approximation will strictly increase and we will obtain an interval of values rather than a precise value given by the coarser abstraction (we got those values by using Givan et al.'s algorithm). The approximation error G_α equals to 0 for all the states while the approximation error $G_{\alpha'}$ is:

$$G_{\alpha'}(1) = G_{\alpha'}(2) \simeq 1 > 0 = G_\alpha(1) = G_\alpha(2) \text{ and}$$
$$G_{\alpha'}(3) \simeq 1 > G_\alpha(3) = 0.$$

Only one of all the possible refinements is detailed here but the same happens for the two other direct refinements:

- For the abstraction
$$\alpha' : 1,2,3,4,5 \to \{1,3\},\{2\},\{4\},\{5\}$$
we have $V^\pm_{\alpha'}(\{1,3\}) = [2.5, 4.78]$ and $V^\pm_{\alpha'}(\{2\}) = [2.75, 4.34]$.
- For the abstraction
$$\alpha' : 1,2,3,4,5 \to \{1\},\{2,3\},\{4\},\{5\}$$
we have $V^\pm_{\alpha'}(\{2,3\}) = [2.25, 5.29]$ and $V^\pm_{\alpha'}(\{1\}) = [2.9, 4.11]$.

□

We can even have a model inspired from the one above where the error strictly increases in each state. Indeed by taking $\gamma = 0.9$ and by changing the transition probabilities in the initial model M to $p(4,4) = p(5,5) = 0.9$ and $p(4,1) = p(5,1) = 0.1$ (we keep the same rewards $R(4) = R(5) = 0$), we can see that the gap $G_{\alpha'}$ is strictly higher than $G_\alpha = 0$ for each direct refinement α'.

Related Work

Our work shows the limitations of some abstractions (state abstractions) in which refining an abstraction may increase the approximation error. This has already been observed in (Waugh et al. 2009) in the case of an i-player poker game (i greater than 2). They looked at the exploitability[3] of each player's strategy with respect to each abstraction and they showed that it may increase while considering finer card abstractions. The same phenomenon has been observed

[3]The exploitability is a metric connected to the Nash equilibrium strategy. It is equal to 0 in the case of a two-player extensive game.

when they considered betting abstractions – by restricting the number of betting options in each sequence of the game. Lately another example about "action abstractions" pathologies has been provided in (Sandholm and Singh 2012).

It is important to notice here that, contrary to what has been done in those works, this paper deals with single action and player models. This suggests that, in some abstractions, notably BMDP abstraction, stochasticity alone can explain this pathological behavior, and that we do not need to consider the more general case of two-player game: the issues appear even in the case of a one-player game (MDP). Indeed, our counter-examples do not even contain any actual action choice, thus identifying this kind of pathology in a very canonical framework.

Interestingly, Kattenbelt et al. (2010) introduce a variant of abstraction for MDPs that, according to their results, does not exhibit said pathology: Every abstraction refinement step results in an improved bound. An interesting open question is how exactly their framework relates to BMDP abstraction.

Conclusions

Somewhat surprisingly, refining an abstraction does not guarantee, in the MDP setting, a refined, i.e., better, approximation of the value function. From a practical perspective, this observation might be reasonably classified as "odd but not crucial" – the loss of this guarantee is not per se an argument against trying to apply abstraction techniques for the computation of heuristic functions, as known from classical planning. The observation *might* be relevant to the practical effectiveness of such methods, where paying a higher price for the abstraction may result in less accuracy. But it remains to be seen whether that is of practical importance.

From a theoretical perspective, we believe that our observations could be of importance for a better understanding of the methods involved. In that regard, our investigation is but a small start into the subject matter. In particular, our vision was and is to identify sufficient criteria, in the bounded-parameter MDP setting, for the error to not increase. If such a criterion is efficiently testable, or can at least be reasonably well approximated, then it could serve as a well-informed guidance during the abstraction refinement process. For the moment, we don't know how such a criterion could be designed. An interesting observation in this context is that our counter-example refines an abstraction that already is a bisimulation.[4] Does that tell us something about the general case? Another question is whether increasing the "extent" of the per-step refinement helps (instead of splitting a single block-state, split 2, 3, ... block-states). Does there exist a non-trivial method (not refining all the way to the original MDP) that guarantees, for any MDP and abstraction thereof, the existence of a refinement step reducing the error? And

[4]If we aggregate the states 4 and 5 together, in the abstract representation α, we get a bisimulation: all the states behave similarly in regards to each block of the partition $\{1,2,3\},\{4,5\}$. We have $R(1) = R(2) = R(3) = 1$, $p(1,\{1,2,3\}) = p(2,\{1,2,3\}) = p(3,\{1,2,3\}) = 0.7$ and $p(1,\{4,5\}) = p(2,\{4,5\}) = p(3,\{4,5\}) = 0.3$, states 4 and 5 are goal states.

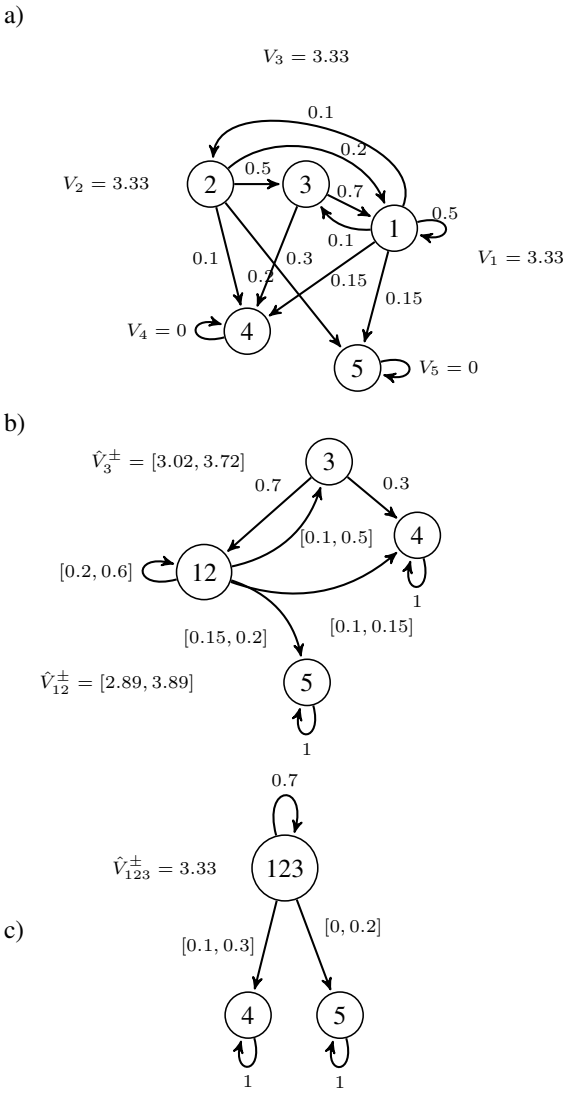


Figure 3: From top to bottom: the MDP M, the abstraction α', the abstraction α.Edges are annotated with transition probabilities. The reward is: $R(1) = R(2) = R(3) = 1 (= R(\{1,2\}) = R(\{1,2,3\}))$, and $R(4) = R(5) = 0$.

can that method be made practical? We believe these are interesting questions for future research, and hope other researchers might join us in exploring them.

Acknowledgments. We thank the anonymous HSDIP'13 reviewers, whose comments helped to improve the paper.

References

Bertsekas, D. P., and Tsitsiklis, J. 1996. *Neuro-Dynamic Programming*. Athena Scientific.

Bonet, B., and Geffner, H. 2003. Labeled RTDP: Improving the convergence of real-time dynamic programming. In Giunchiglia, E.; Muscettola, N.; and Nau, D., eds., *Proceedings of the 13th International Conference on Automated Planning and Scheduling (ICAPS-03)*, 12–21. Trento, Italy: Morgan Kaufmann.

Edelkamp, S. 2001. Planning with pattern databases. In Cesta, A., and Borrajo, D., eds., *Recent Advances in AI Planning. 6th European Conference on Planning (ECP-01)*, Lecture Notes in Artificial Intelligence, 13–24. Toledo, Spain: Springer-Verlag.

Givan, R.; Leach, S. M.; and Dean, T. 1997. Model reduction techniques for computing approximately optimal solutions for Markov Decision Processes. *Artificial Intelligence* 122(1-2):1–8.

Givan, R.; Leach, S. M.; and Dean, T. 2000. Bounded-parameter Markov Decision Processes. *Artificial Intelligence* 122(1-2):71–109.

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.; and Hoffmann, J. 2007. Flexible abstraction heuristics for optimal sequential planning. In Boddy, M.; Fox, M.; and Thiebaux, S., eds., *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS-07)*, 176–183. Providence, Rhode Island, USA: Morgan Kaufmann.

Kattenbelt, M.; Kwiatkowska, M.; Norman, G.; and Parker, D. 2010. A game-based abstraction-refinement framework for Markov Decision Processes. *Formal Methods in System Design* 36(3):246–280.

Katz, M.; Hoffmann, J.; and Helmert, M. 2012. How to relax a bisimulation? In Bonet, B.; McCluskey, L.; Silva, J. R.; and Williams, B., eds., *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS 2012)*. AAAI Press.

Nissim, R.; Hoffmann, J.; and Helmert, M. 2011. Computing perfect heuristics in polynomial time: On bisimulation and merge-and-shrink abstraction in optimal planning. In Walsh, T., ed., *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI'11)*, 1983–1990. AAAI Press/IJCAI.

Ortner, R. 2011. Adaptive aggregation for reinforcement learning in average reward Markov Decision Processes. *Annals of Operationnal Research*.

Sandholm, T., and Singh, S. 2012. Lossy stochastic game abstraction with bounds. In *Proceedings of the 13th ACM Conference on Electronic Commerce*, EC '12, 880–897. ACM.

Waugh, K.; Schnizlein, D.; Bowling, M.; and Szafron, D. 2009. Abstraction pathologies in extensive games. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '09, 781–788.

Landmark-based Meta Best-First Search Algorithm: First Parallelization Attempt and Evaluation

Simon Vernhes, Guillaume Infantes and Vincent Vidal
Onera Toulouse, France
firstname.lastname@onera.fr

Abstract

In this paper, we revisit the idea of splitting a planning problem into subproblems hopefully easier to solve with the help of landmark analysis. While this technique initially proposed in the first approaches related to landmarks has been outperformed by landmark-based heuristics, we believe that it is still a promising research direction. To this end, we propose a new method for problem splitting based on landmarks which has two advantages over the original technique: it is complete (if a solution exists, the algorithm finds it), and it uses the precedence relation over the landmarks in a more flexible way. We lay in this paper the foundations of a meta best-first search algorithm, which explores the landmark orderings to create subproblems and can use any embedded planner to solve subproblems. Furthermore, we propose and evaluate a parallel version of this algorithm. It opens up avenues for future research: among them are new heuristics for guiding the meta search towards the most promising orderings, different policies for generating subproblems, influence of the embedded subplanner and other parallelization strategies of the meta search.

1 Introduction

Automated Planning in Artificial Intelligence (Ghallab, Nau, and Traverso 2004) is a general problem solving framework which aims at finding solutions to combinatorial problems formulated with concepts such as actions, states of the world, and goals. Landmark-based analysis is actually among the most popular tools to build efficient planning systems, either optimal or suboptimal. Landmarks are facts that must be true at some point during the execution of any solution plan, and some of them can be found, as well as an ordering, in polynomial time (Hoffmann, Porteous, and Sebastia 2004; Keyder, Richter, and Helmert 2010). Landmarks have been used in two main ways. The most successful one is the design of heuristic functions to guide search algorithms, such as the landmark-counting heuristic used in the LAMA suboptimal planner (Richter, Helmert, and Westphal 2008) or the LM-Cut heuristic for optimal cost-based planning (Helmert and Domshlak 2009). An anterior method proposed in (Hoffmann, Porteous, and Sebastia 2004) is to divide a planning problem into successive subproblems whose goals are disjunctions of landmarks to be achieved in turn by any embedded planner. This method is not as efficient as using landmark-based heuristics: among the most prominent problems are its incompleteness and its lack of flexibility with respect to an initial ordering of the landmarks. STeLLa (Sebastia, Onaindia, and Marzal 2006) is another problem-splitting method which creates a set of subproblems using conjunctions of landmarks.

We aim in this paper to revisit these last methods, with the objective of devising a complete algorithm for subproblem splitting based on landmarks. Roughly speaking, our method consists in performing a best-first search algorithm in the space of landmark orderings, in which node expansion implies the search of a subproblem by an embedded planner, these orderings being explored in parallel. This search algorithm is performed at a meta level, the low level being the search made by the embedded planner that can itself use a best-first search algorithm. After giving some background about classical planning and landmark computation, we define the basic components later used to describe the landmark-based meta best-first search algorithm (LMBFS), along with several heuristics to guide the meta search. We then propose a parallel version of this algorithm and experimentally evaluate both sequential and parallel versions. We finally conclude and present some perspectives for future works.

2 Background on Classical Planning

2.1 STRIPS Model of Planning

The basic STRIPS (Fikes and Nilsson 1972) model of planning can be defined as follows. A *state* of the world is represented by a set of ground atoms. A *ground action* a built from a set of atoms A is a tuple $\langle pre(a), add(a), del(a)\rangle$ where $pre(a) \subseteq A$, $add(a) \subseteq A$ and $del(a) \subseteq A$ represent the preconditions, add and delete effects of a, respectively.

A *planning problem* is defined as a tuple $\Pi = \langle A, O, I, G\rangle$, where A is a finite set of *atoms*, O is a finite set of ground actions built from A, $I \subseteq A$ represents the *initial state*, and $G \subseteq A$ represents the *goal* of the problem. The *application* of an action a to a state s is possible if and only if $pre(a) \subseteq s$ and the resulting state is $s' = (s \setminus del(a)) \cup add(a)$. A *solution plan* is a se-

quence of actions $\langle a_1, \ldots, a_n \rangle$ such that for $s_0 = I$ and for all $i \in \{1, \ldots, n\}$, the intermediate states $s_i = (s_{i-1} \setminus del(a_i)) \cup add(a_i)$ are such that $pre(a_i) \subseteq s_{i-1}$ and $G \subseteq s_n$. $S(\Pi)$ denotes the set of all solution plans of Π.

We also denote $\circ$ the concatenation of two plans, i.e. $\langle a_1, \ldots, a_i \rangle \circ \langle a_j, \ldots, a_k \rangle = \langle a_1, \ldots, a_i, a_j, \ldots, a_k \rangle$.

2.2 Landmarks

Classical landmark definitions state that *landmarks* are facts that must be true at some point during the execution of any solution plan (Hoffmann, Porteous, and Sebastia 2004; Keyder, Richter, and Helmert 2010). We use the following definition of landmarks:

Definition 1 (Causal landmark)**.** *(Zhu and Givan 2003) Given a planning problem $\Pi = \langle A, O, I, G \rangle$, an atom l is a* causal landmark *for Π if either $l \in G$ or $\forall \rho \in S(\Pi), \exists a \in \rho : l \in pre(a)$.*

An intuitive precedence relation among landmarks and a graph based on this relation can be defined as follows:

Definition 2 (Precedence relation $<_{\mathcal{L}}$)**.** *$<_{\mathcal{L}}$ can be defined on a set of landmarks $\mathcal{L}$: $(\forall (l, l') \in \mathcal{L}^2)\ l <_{\mathcal{L}} l'$ if the first occurence of l is reached before the first occurence of l' by the execution of every solution plan.*

Definition 3 (Landmark graph Γ)**.** *Given a set of landmarks $\mathcal{L}$ and a precedence relation $<_{\mathcal{L}}$, we define $\Gamma = (\mathcal{V}, \mathcal{E})$, the corresponding landmark directed graph where the set of vertices $\mathcal{V} = \mathcal{L}$ and the set of edges $\mathcal{E}$ is the transitive reduction of the graph $(\mathcal{V}, \{(l, l') \in \mathcal{L}^2 \mid l <_{\mathcal{L}} l'\})$.*

Definition 4 (Relatives of a landmark l)**.** *Accordingly to the graph Γ, we denote $Pa_\Gamma(l)$ the set of parents of l, $Ch_\Gamma(l)$ the set of children of l, and $\mathcal{P}_\Gamma(l)$ the set of ancestors of l.*

We now introduce the following definitions that we will rely on. First, we denote *root landmarks* of a landmark graph all landmarks associated to vertices with no parents:

Definition 5 (Root landmark set)**.** *Let $\Gamma = (\mathcal{V}, \mathcal{E})$ be a landmark graph: $roots(\Gamma) = \{l \in \mathcal{V} \mid Pa_\Gamma(l) = \emptyset\}$.*

We now define the subgraph $\Gamma \setminus \mathcal{A}$ built by removing from the landmark graph Γ the vertices associated to landmarks in $\mathcal{A}$ and their corresponding edges:

Definition 6 (Landmark subgraph)**.** *Let $\Gamma = (\mathcal{V}, \mathcal{E})$ be a landmark graph and $\mathcal{A}$ be a set of landmarks: $\Gamma \setminus \mathcal{A} = (\mathcal{V} \setminus \mathcal{A},\ \{(v, v') \in \mathcal{E} \mid v \notin \mathcal{A} \wedge v' \notin \mathcal{A}\})$.*

Landmark Graph Generation Practicals methods proposed to produce landmark graphs (Hoffmann, Porteous, and Sebastia 2004; Zhu and Givan 2003) are based on a Relaxed Planning Graph (RPG) of Π. More complex types of landmarks might be considered (Keyder, Richter, and Helmert 2010). In this work, we choosed the method of (Zhu and Givan 2003) for its simplicity.

Related Works on Using Landmarks Previous approaches used landmarks in mainly two different ways. One approach is computing heuristics. For example, the LAMA heuristic (Richter, Helmert, and Westphal 2008) estimates a heuristic value of the states by counting unreached and required again landmarks. Another approach is to split a planning problem into subproblems. Disjunctive Search Control (DSC) (Hoffmann, Porteous, and Sebastia 2004) is a search control algorithm based on the landmark graph. It runs a subplanner on the problem Π whose goal is the disjunction of the roots of the landmark graph and G. If a valid plan is found, then the reached landmark is removed from the landmark graph and the algorithm iterates (the reached state is used as the new initial state) until the landmark graph is empty. Finally, the subplanner is called a last time with G as goal.

3 The Landmark-based Meta Best-First Search (LMBFS) Algorithm

Our approach is based on problem splitting with a flexible exploitation of the landmark graph: LMBFS performs a best-first search in a space of subproblems generated on-the-fly, based on possible landmark orderings.

More precisely, LMBFS builds a search tree where nodes represent planning problems that are subproblems of the original one. Solving subproblems along a branch of this tree leads to iteratively reach each landmark in a possible ordering, the initial state of each subproblem being the final state obtained by applying the plan found for the previous subproblem. We formally define in this section the metanodes and associated planning problems, as well as different ways of generating the next subproblems to solve from a metanode (the children of that metanode in the search tree). Both aspects heavily rely on the landmark graph and on the partial order it defines. We then give the complete algorithm, heuristics used and implementation details.

In the following, we consider a planning problem $\Pi = \langle A, O, I, G \rangle$, its corresponding set of landmarks $\mathcal{L}$, and Γ the landmark graph associated to Π.

3.1 Metanode and Associated Planning Problem

We first define metanodes and associated problems:

Definition 7 (Metanode)**.** *A* metanode *is a tuple $m = \langle s, h, \mathcal{A}, l, \rho \rangle$ where:*

- *s is a state of the planning problem Π;*
- *h is a heuristic evaluation of the node;*
- *$\mathcal{A}$ is a set of landmarks ($\mathcal{A} \subseteq \mathcal{L}$);*
- *l is a landmark ($l \in \mathcal{L}$);*
- *ρ is a plan yielding the state s from the initial state I.*

Definition 8 (Metanode-associated planning problem)**.** *The planning problem associated to a metanode $m = \langle s, h, \mathcal{A}, l, \rho \rangle$ is $\Pi_m = \langle A, ops_\Gamma(l, \mathcal{A}), s, \{l\} \rangle$ with $ops_\Gamma(l, \mathcal{A})$ a subset of O defined below.*

We consider the planning problem where s is the initial state, A is the set of ground atoms of Π, $\{l\}$ is the goal. In order to focus search on reaching l, we forbid the actions producing some other landmarks by defining $ops_\Gamma(l, \mathcal{A})$ as a subset of actions associated to a landmark subgraph:

Definition 9 (Landmark subgraph action restriction)**.** *Let* $m = \langle s, h, \mathcal{A}, l, \rho \rangle$ *be a metanode.* $ops_\Gamma(l, \mathcal{A}) = \{a \in O \mid (l \in add(a)) \vee (add(a) \cap roots(\Gamma \setminus \mathcal{A}) = \emptyset)\}$.

In other words, $ops_\Gamma(l, \mathcal{A})$ is the set of actions producing l and actions which does not produce any root landmarks of the subgraph $\Gamma \setminus \mathcal{A}$ (except if they also produce l). In our algorithm, $\mathcal{A}$ will be the set of already achieved landmarks.

3.2 Expansion of Metanodes

There are several ways to generate children of a metanode, or equivalently defining other subproblems to solve. Let us recall that a metanode $m = \langle s, h, \mathcal{A}, l, \rho \rangle$ defines a problem starting from s and focusing on achievement of landmark l by forbidding actions producing other landmarks of $roots(\Gamma \setminus \mathcal{A})$ (except if they also produce l). In the following, h' is the heuristic evaluation of the generated metanode, discussed in section 3.4.

Next Landmarks Metanode Generation This first metanode generator tries to follow the landmark graph Γ as closely as possible, exploring sequences from the roots to the leaves. The idea is the following: when the goal landmark of the metanode can be reached, generate children in order to reach other root landmarks in Γ. We thus define the *nextLM* operator as:

Definition 10 (Next landmarks metanode generation)**.** *Let* $m = \langle s, h, \mathcal{A}, l, \rho \rangle$ *be a metanode. If* Π_m *has a solution* ρ'*, then* $\mathrm{nextLM}(m) = \{\langle s', h', \mathcal{A} \cup \{l\}, l', (\rho \circ \rho') \rangle \mid l' \in roots(\Gamma \setminus (\mathcal{A} \cup \{l\}))\}$ *where* s' *is the state obtained by applying* ρ' *to* s*. If* Π_m *has no solution,* $\mathrm{nextLM}(m) = \emptyset$.

In other words, at a metanode m, we try to reach the landmark l. If there is a plan, the landmark l is added to the set of already achieved landmarks $\mathcal{A}$, and the partial plan is updated accordingly. Then, new metanodes are generated by looking at root landmarks in the restricted graph $\Gamma \setminus (\mathcal{A} \cup \{l\})$.

Remark. Using nextLM, we can explore every total order created from the precedence relation $<_\mathcal{L}$, which was our objective. Indeed, consider a metanode focusing on an initial root landmark of Γ. If we generate its children using the nextLM operator, then selecting one of them and iterating the process in a depth-first way, we will eventually empty the landmark graph Γ, achieving the exploration of a total order of all landmarks.

Unfortunately, even if the landmark graph Γ is sound and complete, using only nextLM makes the algorithm incomplete, as shown in the following counterexample. Let us consider the example in Figure 1 where circles are atoms, squares are actions, arrows mean precondition of an action or production of an atom and dashed arrows mean deletion of an atom. The initial state is $\{a, f, d\}$ and the goal set is $\{c\}$. As we can see, g and c are landmarks, and g has to be reached before c. The first metanode will have the landmark g as goal. The subplanner gives the plan $\langle \alpha \rangle$. The only generated metanode added to the open list is $m = \langle \{f, g\}, h, \{g\}, c, \langle \alpha \rangle \rangle$. The associated problem Π_m

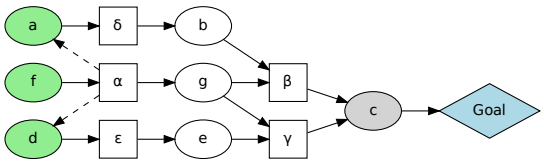


Figure 1: Planning Graph of "nextLM problem".

is unsolvable, as α deletes the preconditions of δ and ϵ which are mandatory actions that should be applied before α.

This led us to define new metanode generators, which, used in conjunction with nextLM, make the algorithm complete. These operators are based on landmark deletion from Γ, allowing for instance in the counterexample to try to reach the goal c without blindly focusing on g first.

Cut-parents Metanode Generation These operators remove the ancestors of a non-root landmark.

Definition 11 (Cut-parents metanode generations)**.** *Let* $m = \langle s, h, \mathcal{A}, l, \rho \rangle$ *be a metanode. If* Π_m *has a solution* ρ'*, then* $\mathrm{cutParent}(m) = \{\langle s', h', \mathcal{A} \cup \mathcal{P}_\Gamma(l'), l', (\rho \circ \rho') \rangle \mid l' \in Ch_\Gamma(l)\}$ *where* s' *is the state obtained by applying* ρ' *to* s*. If* Π_m *has no solution, then* $\mathrm{cutParent}(m) = \emptyset$.

Definition 12 (Restart cut-parents metanode generation)**.** $\mathrm{restartCutParent}(\langle s, h, \mathcal{A}, l, \rho \rangle) = \{\langle I, h', \mathcal{A} \cup \mathcal{P}_\Gamma(l'), l', \emptyset \rangle \mid l' \in Ch_\Gamma(l)\}$ *where* I *is the initial state of the original planning problem.*

The idea is that a total order constructed on the partial order defined by the landmark graph may be too restrictive, as in the counterexample. Using these two operators, some landmarks may be skipped by trying to reach deeper landmarks.

Delete Landmark Metanode Generation Finally, we introduce the very generic landmark deletion operator: metanodes are generated as if the deleted landmark did not exist:

Definition 13 (Delete landmark metanode generation)**.** $\mathrm{deleteLM}(\langle s, h, \mathcal{A}, l, \rho \rangle) = \{\langle s, h', \mathcal{A} \cup \{l\}, l', \rho \rangle \mid l' \in roots(\Gamma \setminus (\mathcal{A} \cup \{l\}))\}$.

This operator discards a landmark, and causes the search to try to achieve remaining root landmarks. Applying this operator enough times on the first metanode (that has I as initial state) empties the landmark graph, eventually giving a metanode associated to the original planning problem. Also, the cut-parents operators can be seen as shortcuts for several delete landmark operators applications, guided by $\mathcal{P}_\Gamma$.

3.3 Algorithm

LMBFS (Algorithm 1) is a best-first search algorithm on metanodes of definition 7, with deferred heuristic evaluation (Richter and Helmert 2009): new nodes are inserted into the open list with the heuristic value of their parent.

Algorithm 1: LMBFS.

input : STRIPS problem $\Pi = \langle A, O, I, G\rangle$, landmark graph Γ, metanode successor function succ
output : solution plan (or $\perp$ if there is no solution)

```
1  open ← ∅; closed ← ∅;
2  ∀l ∈ roots(Γ) : add ⟨I, h, ∅, l, ∅⟩ to open;
3  while open ≠ ∅ do
4      m ← arg min_{⟨s,h,A,l,ρ⟩∈open} h;
5      open ← open \ {m};
6      if m ∉ closed then
7          closed ← closed ∪ {m};
8          ρ' ← subplanner(Π_m);
9          if ρ' ≠⊥ then
10             s' ← result of executing ρ' in s;
11             if G ⊆ s' then
12                 return ρ ∘ ρ';
13             open ← open ∪ succ(m);
14 return ⊥
```

The algorithm is run on the problem $\Pi_g = \langle A \cup \{g\}, O \cup \{a_g\}, I, \{g\}\rangle$ where g is a dummy atom representing goal achievement, and a_g is a dummy action whose precondition is G and add effect is $\{g\}$. g is a landmark whose achievement implies that a solution to Π has been found.

First, the metanodes associated to each root landmark of Γ are added to the open list. Then, at each iteration, the best metanode m (according to a heuristic detailed in section 3.4) is extracted from the open list, and a subplanner is run on the associated problem Π_m. If the subplanner returns a plan, m is expanded by adding its *successors* to the open list. The algorithm iterates until the open list is empty or g is reached.

The function succ applied to the metanode m (Algorithm 1 line 13) computes the set obtained by an operator or the union set of several operators described in section 3.2. In our planner, we have implemented two successor functions:

- $\text{succDel}(m) = \text{nextLM}(m) \cup \text{deleteLM}(m)$
- $\text{succCut}(m) = \text{nextLM}(m) \cup \text{cutParent}(m) \cup \text{restartCutParent}(m)$

The operator nextLM is at the heart of our algorithm in order to focus on sequences of landmarks. However to ensure completeness, we have to use a combination of the other operators: as a net effect of applying these operators at each node expansion, the metanode $m = \langle I, h', \mathcal{L} \setminus \{g\}, g, \emptyset\rangle$ corresponding to the global problem Π_g will appear.

Theorem 1. *The LMBFS algorithm using* succCut *or* succDel *as* successor *function is sound and complete if the subplanner is sound and complete.*

Proof. (sketch) Soundness comes from: (1) the state in the first metanode is the initial state I of the problem, (2) all successor operators build plans that can be concatenated to form a solution of the global problem or search a new plan from I, and (3) if the final landmark g appears in a metanode, then achieving it solves the global problem goal. (2) is obtained by induction: if a metanode $m = \langle s, h, \mathcal{A}, l, \rho\rangle$ is such that s is reachable by applying ρ from I, then by definition of nextLM and cutParent, s' is the state obtained by applying ρ' to s and so s' is reachable by applying $\rho \circ \rho'$ from I (using subplanner soundness). deleteLM modifies neither s nor ρ, so the recursive property is ensured. restartCutParent produces a metanode $m = \langle I, h, \mathcal{A}, l, \emptyset\rangle$: a new search from I is started, giving a sound plan if the subplanner is sound.

Completeness comes from the fact that the operators restartCutParents (for the succCut successor function) and deleteLM (for the succDel successor function) are systematically used at each expansion of a metanode. So, the search graph contains a branch starting from the initial metanode consisting only of applications of restartCutParents or deleteLM, and both will have the effect to (1) keep I as associated state, (2) put all landmarks but g in the set of landmarks $\mathcal{A}$, and (3) produce a final metanode whose associated landmark is g. From definition 9, $ops_\Gamma(g, \mathcal{L} \setminus \{g\}) = O \cup \{a_g\}$: all actions of the original problem are used for solving this final metanode, which is the global problem, and so completeness of LMBFS derives from completeness of the subplanner. □

Lazy Metanode Generation The delete landmark metanode generation (section 3.2) can generate a considerable amount of metanodes for some instances, thus inducing a slow-down during the insertion of these metanodes in the open list. To overcome this issue, we generate metanodes with deleteLM only when the open list is empty. When a metanode is inserted in the closed list, it is also pushed into a secondary open list. When the main open list is empty, we simply pop a metanode m from the secondary open list, and generate its children using $deleteLM(m)$. The heuristics for ordering metanodes in the secondary open list are the same as the ones used for the main open list.

3.4 Heuristics for Metanode Selection

In order to improve the algorithm effectiveness, the most promising metanode from the open list has to be selected. For doing so, a metanode generated by nextLM is always preferred over others, in order to focus search on reaching landmarks in sequence. Thus, the expansion of other (degraded) metanodes is delayed until we have no other choice, in the spirit that search can be focused using preferred operators (Richter and Helmert 2009).

Three heuristic functions have been implemented. The first ones evaluate G from the starting state of the metanode, with the well-known heuristics h^{add} (Bonet and Geffner 2001) and h^{ff} (Hoffmann and Nebel 2001). The last one, inspired by the landmark-counting heuristic of LAMA (Richter, Helmert, and Westphal 2008), uses the landmark graph Γ and counts the remaining landmarks to be reached. The metanode with the least number of remaining landmarks is chosen, enforcing a depth-first search in the graph. We will refer to this heuristic as $h^{\mathcal{L}_{left}}$.

Definition 14 ($h^{\mathcal{L}_{left}}$)**.** *For a metanode $m = \langle s, h, \mathcal{A}, l, \rho\rangle$ and an associated landmark graph $\Gamma = (\mathcal{V}, \mathcal{E})$, the heuristic $h^{\mathcal{L}_{left}}$ is defined by $h^{\mathcal{L}_{left}}(m) = |\mathcal{V} \setminus \mathcal{A}|$.*

4 Parallel LMBFS

A well known parallelization scheme is the principle used in HDA* (Kishimoto, Fukunaga, and Botea 2009): the idea is to distribute search nodes among the processing units (PUs) based on a hash key computed from planning states. Doing so, the list of nodes to be expanded (the open list) owned by each PU are disjoint: computations made on a given state (applicable actions, heuristic function, etc.) are performed only once, only by the PU the node "belongs" to.

Another important aspect is that communication between PUs can be performed asynchronously: a PU expands nodes from its open list, sends sons to the PUs they belong to, and periodically checks its incoming messages to incorporate new nodes into its open list. The principle has been initially conceived for optimal planning with successful results (Kishimoto, Fukunaga, and Botea 2009). It has also been applied successfully to suboptimal planning (Vidal, Vernhes, and Infantes 2011).

4.1 Algorithm

In our case, metanodes are distributed among the PUs. To do so, we define a key for metanode $m = \langle s, h, \mathcal{A}, l, \rho \rangle$ using (1) the starting state s, (2) the set of already achieved landmarks $\mathcal{A}$ and (3) the landmarks goal l. This key is hashed into a natural number modulo the number of PUs, thus giving the single PU the metanode belongs to. So the main modification is to distribute metanodes when they are generated instead of adding them to the local open list. Each PU runs a slightly modified version of the sequential search described in section 3, with its own open and closed lists.

We now give some details on the induced modifications:

- line 2: initial metanodes are only created by the master PU, and distributed among the PUs;
- line 3: the main loop condition is modified in order to wait if the open list is empty (rather than exiting); the loop is stopped only if a PU has found a solution (a message is broadcasted to all the PUs when a solution is found);
- line 4: before choosing the next metanode to expand, the algorithm checks if there are incoming metanodes, and if it is the case, it incorporates them in its open list;
- line 13: after calls to the metanode generators, the hash keys of generated metanodes are computed and metanodes are send asynchronously to the corresponding PUs.

4.2 Implementation details

In order to save memory, the current partial plan ρ (definition 7) is not stored inside the metanode in our implementation; instead, a metanode stores the partial plan ρ' (definitions 10 and 11) and a pointer to its parent metanode. So, in order to retrieve the solution plan, when a PU reaches the goal, it becomes a master PU and asks for its parent metanode partial plan to the PU owning it, concatenates it to its plan, and iterates until an initial metanode has been reached.

Another issue that came up during the adaption of HDA* parallelization scheme to LMBFS is that sometimes, the sequential algorithm gets stuck for a long time in a call to the subplanner searching for a solution to a (hard or unfeasible) subproblem. If such a case appears in one of the PUs, the parallel algorithm cannot finish until this PU ends its current search. To overcome this issue, we modified the underlying planner to check if a stop message has been received (this check is performed right before YAHSP opens a new node).

One improvement pointed out in the work with transposition-table driven scheduling for parallel IDA* (Romein et al. 1999), is to overcome the issue of the communication overhead by packing multiple nodes with the same destination into a single message. This packing strategy has also been experimented in HDA* (Kishimoto, Fukunaga, and Botea 2009), and is also implemented in our algorithm.

The lazy metanode generation cannot be implemented as-is in parallel because the metanodes generated by a PU will not necessarily be expanded by this PU; which means that emptiness of a PU primary open list does imply that metanodes generated by other operators than $\mathrm{deleteLM}$ have all been expanded. So we cannot rely on such condition to defer generation of metanodes by $\mathrm{deleteLM}$ operator.

5 Experiments

5.1 Experimental Setup

We conducted a set of experiments on a selection of benchmarks from the 3rd to the 7th International Planning Competition (IPC) within a 10 minutes CPU time limit. The experiments were all run on an Intel X5670 processor running at 2.93Ghz with 24GB of RAM. In the next figures, each plot represents an IPC problem. Only results with the $\mathrm{succDel}$ operator are reported here, as it yielded better results than $\mathrm{succCut}$. However, actually, nodes generated by nextLM are always preferred over nodes generated by these two operators, and we think that relevant heuristics for interleaving both kind of nodes might give a different picture.

Subplanner Embedded in LMBFS For subproblem resolution, we use YAHSP (Vidal 2004; 2011) for two reasons.

Firstly, we do not want to use a subplanner that also uses landmarks internally (especially if non-negligible preprocessing time is required), as our objective is to evaluate a new use of landmarks without benefiting from them in any other way.

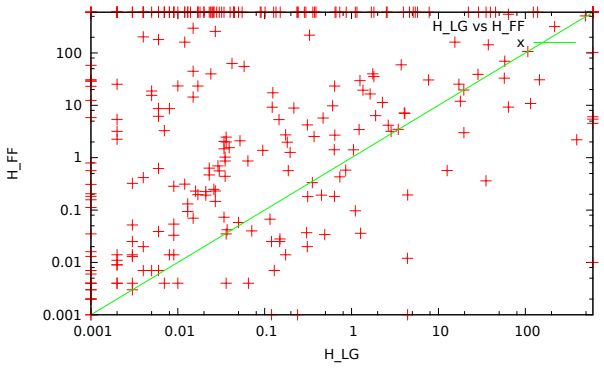
Secondly, because the successive subproblems solved during metanode expansion should be, and generally are, easy to solve with very few lookaheads computed in YAHSP. Moreover, directly embedded in the form of a C library, YAHSP does not require any preprocessing when faced with a new subproblem. It can thus generally answer very fast. It has also already been embedded with some success in another planner based on evolutionary algorithms (Bibaï et al. 2010) for solving different kinds of subproblem sequences.

Selected Domains To come up with a test suite, we ran preliminary tests in all STRIPS domains from the IPC. We selected some domains which YAHSP does not solve too easily (using few lookaheads), and also included some domains it solved easily to exhibit the possible slow-down required by the pre-computation of the landmark graph and

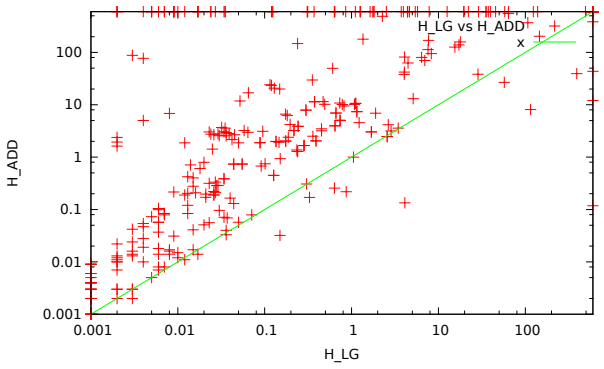
the splitting into smaller instances of already easy solvable problems. Thus, we selected 14 domains[1] (390 problems), which we believe could represent a classic set of domains for the IPC.

5.2 Sequential evaluation

Landmark Graph Generation For most problems, the computation time of the landmark graph is low. It takes less than $0.1s$ for 86% of the instances, and less than $1s$ for 97% of the instances (on sequential setup described below). Even if the computational time of the landmark graph on the initial state is acceptable, we consider it too long to be processed at each metanode during search. Recomputing landmarks could be more informative for search but, as LMBFS is designed for speed, we did not investigate such an option. Another reason is that adding new landmarks in the graph would break the current algorithm's completeness, which is based on emptying the initial landmark graph.



(a) h^{ff} vs. $h^{\mathcal{L}_{left}}$ (denoted H_LG)



(b) h^{add} vs. $h^{\mathcal{L}_{left}}$ (denoted H_LG)

Figure 2: Runtimes of LMBFS with several heuristics (in seconds).

Efficiency of the Different Heuristics Figure 2(a) shows a comparison of the runtime of LMBFS with $h^{\mathcal{L}_{left}}$ (x-axis) and h^{ff} (y-axis). As we can see, most of the plots are above $y = x$, meaning most instances have been solved faster using $h^{\mathcal{L}_{left}}$. Figure 2(b) shows a comparison between $h^{\mathcal{L}_{left}}$ (x-axis) and h^{add} (y-axis). The results are again in favor of $h^{\mathcal{L}_{left}}$ which outperforms h^{add} in most of the problems. Table 1 summarizes the runs performed with LMBFS using several heuristics. It shows that LMBFS with the $h^{\mathcal{L}_{left}}$ heuristic outperforms the two other (with h^{add} or h^{ff}).

Heuristic	solved $< 1s$	solved $< 10s$	solved
h^{add}	45.64%	63.85%	75.13%
h^{ff}	34.62%	45.38%	57.95%
$h^{\mathcal{L}_{left}}$	74.36%	85.90%	92.31%

Table 1: Coverage of LMBFS using different heuristics (10 minutes timeout).

Lazy Metanode Generation Table 2 compares the speed-up obtained by using the lazy metanode generation described in section 3.3. We can see that there is a strictly positive speed-up for 51.03% of the problems, and a speed-up superior to 2 for 23.74% of them. Even if there is a noticeable slow-down for 4.36% of the problems (heuristically equivalent nodes may be ordered differently in the main and secondary open lists due to implementation details), it still is a nice improvement for the overall test suite.

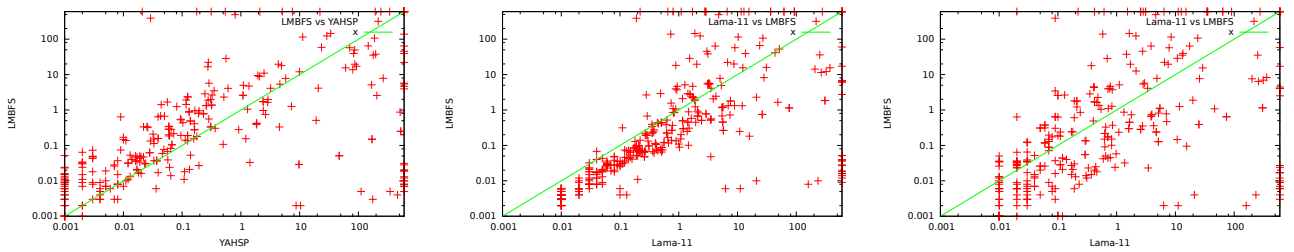
		Instances where the speed-up/slow-down is		
	Average	> 1	> 1.05	> 2
Speed-up	8.11	51.03%	36.41%	22.74%
Slow-down	2.24	14.10%	4.36%	1.81%

Table 2: Speed-up of lazy metanode generation.

LMBFS versus sub-planner (YAHSP) Table 3 summarizes the coverage of runs performed with LMBFS using the $h^{\mathcal{L}_{left}}$ heuristic in comparison with YAHSP and some state-of-the-art planners, also shown as a curve in function of the timeout in Figure 4. The comparison between LMBFS and YAHSP is also depicted in Figure 3(a). It shows that many problems are solved within 1s and most of the problems quickly solved by YAHSP (under 0.1s) are solved by LMBFS nearly as fast. On harder instances we can also see that LMBFS shows its benefits in terms of running time. Finally the total number of solved instances is slightly in favor of LMBFS.

A major drawback that has been pointed out for the DSC algorithm and the SteLLa planner is the length of computed plans which can be significantly higher. Compared to the plans computed by YAHSP, the average plan length computed by LMBFS is 8% shorter. In 43% instances LMBFS computes strictly shorter plan than YAHSP and in 19% instances the plans are strictly longer.

[1] ipc3-driverlog, ipc3-freecell, ipc3-satellite, ipc4-pipesworld_tankage, ipc4-psr_small, ipc[56]-openstacks, ipc5-pathways, ipc[67]-transport, ipc[67]-scanalyzer, ipc7-barman, ipc7-floortile



(a) LMBFS vs. YASHP. (b) LMBFS vs. LAMA (w/ preprocessing). (c) LMBFS vs LAMA (w/o preprocessing).

Figure 3: Runtimes of LMBFS with $h^{\mathcal{L}_{left}}$ versus YASHP and LAMA-11 (in seconds).

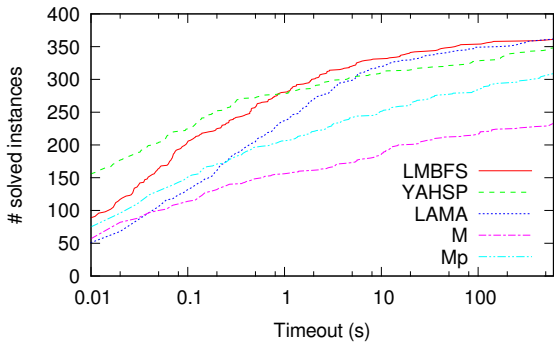


Figure 4: Coverage of LMBFS with $h^{\mathcal{L}_{left}}$ versus state-of-the-art planners as a function of the timeout (in seconds).

Planner	solved $< 1s$	solved $< 10s$	solved
with preprocessing time			
LMBFS	71.79%	84.87%	92.31%
YAHSP	71.28%	79.74%	88.97%
LAMA-11	60.51%	81.79%	92.05%
M	40.00%	47.44%	58.72%
Mp	52.82%	64.36%	78.46%
without preprocessing time			
LMBFS	74.36%	85.90%	92.31%
LAMA-11	69.74%	82.82%	92.56%

Table 3: Coverage of LMBFS with $h^{\mathcal{L}_{left}}$ versus state-of-the-art planners (10 minutes timeout), with and without preprocessing time for LMBFS and LAMA-11.

LMBFS versus State-of-the-Art Planners Table 3 and Figure 4 compare LMBFS to the LAMA-11 planner (based on landmark analysis but in a different way), as well as to the SAT-based planners M and Mp (**?**).

Figures 3(b) and 3(c) are scatter plots comparing the runtimes of LMBFS versus LAMA-11. Figure 3(b) shows runtimes including preprocessing within the 10 minutes timeout, and Figure 3(c) without taking into account the preprocessing time, as LAMA-11 can spend a lot of time during this stage.

These evaluations show that on our selection of domains, LMBFS is competitive with the state-of-the-art. It clearly outperforms M, Mp and YAHSP (for problems that require at least 1 second). It also outperforms LAMA-11, although the overall performance for a 10 minutes timeout depends on whether or not the preprocessing time is included.

LMBFS versus YAHSP (quality) A major drawback that has been pointed out for the DSC and STeLLA algorithms is the length of computed plans which can be significantly higher. Compared to the plans computed by YAHSP, the average plan length computed by LMBFS is 8% shorter. In 43% instances LMBFS computes strictly shorter plan than YAHSP and in 19% instances the plans are strictly longer.

5.3 Parallel version evaluation

To implement the parallel version, we used the Message Passing Interface (MPI) to spawn the different processes and pass messages between them. We conducted a set of experiments on a cluster of 4 servers. Each one embeds two 6-core CPUs (Intel X5670) running at 2.93Ghz with 24GB of RAM. They are connected via a gigabit network.

The parallel version of LMBFS in a HDA* fashion is denoted MPI $a \times b$ where a is the number of servers used and b is the number of processes per server.

For comparison purposes, a parallel non-cooperative version has also been developed: it consists in several sequential versions of LMBFS with no communication between the processes. The only difference in these sequential versions is that the order of the nodes which have the same heuristic value is randomized (in the open list) for both LMBFS and the underlying planner YAHSP. This random version is simply denoted Random and is run with 48 processes in parallel (with different seeds for the random number generator). For this last version, we consider only the statistics of the process which found first the solution (if any).

Number of expanded nodes We can first take a look at the growth of the number of expanded metanode per instances: Figure 5. The number of expanded nodes seems to scale well when the number of PUs increase. The random version mostly expands the same number of nodes than the classic sequential version (we consider only one process).

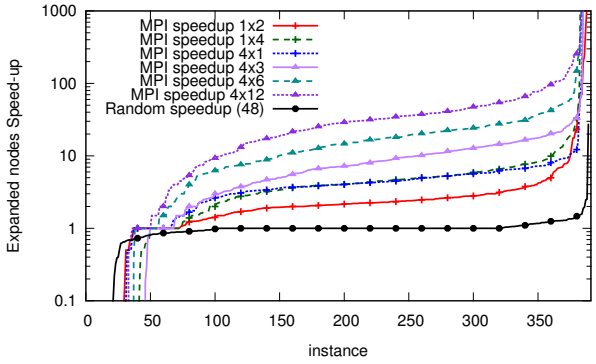
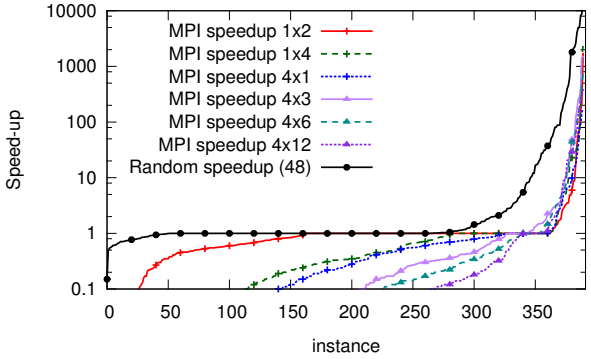
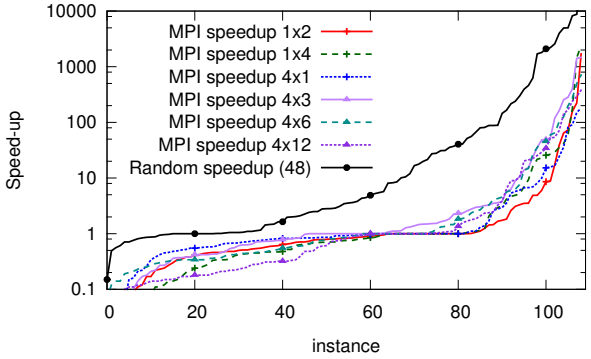


Figure 5: Expanded node speedup of parallel vs. sequential version (instances sorted by increasing speedup)



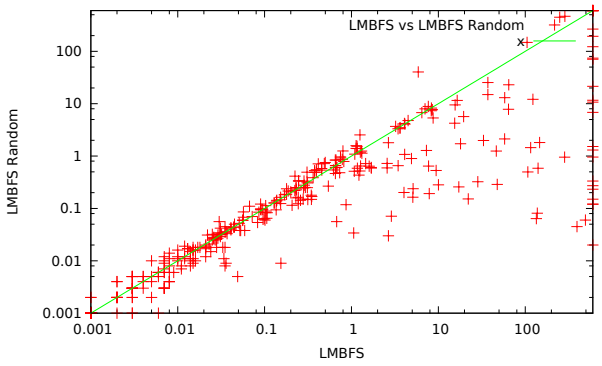
(a) Wall-clock time speedup of parallel vs sequential version



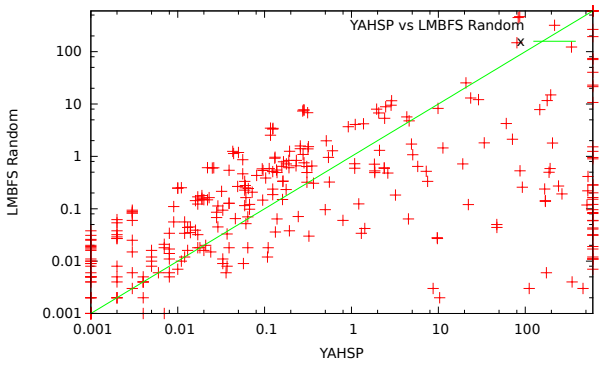
(b) Wall-clock time speedup of parallel vs sequential version (where the sequential version takes more than $1s$)

Figure 6: Solving time speedups (sorted by speedup)

Solving performance Figure 6(a) shows however, that even if the number of expanded nodes has increased a lot, the MPI algorithm does not scale up at all; it performs even worse for some instances. MPI takes some pre-processing time to boot up (around one second); but even if we just select instances that are solved in more than one second (Figure 6(b)), we can see that in most cases parallel versions take the same time as the sequential version (or even more, as we



(a) LMBFS runtimes: sequential version vs. Random (48)



(b) Runtimes of YAHSP and LMBFS Random (48)

Figure 7: Results for Random version (in seconds)

increase the number of processes). Finally, there are linear or super-linear speedups for only a small subset of instances.

The coverage is not improved either, but it increases with the number of processes: it goes from 87.69% for the MPI 1×4 version to 92.05% for the MPI 4×12 version (the only exception is the MPI 1×2 version with 90.26%). Clearly, the breadth induced by this parallelization is not efficient.

Open List Randomization On previous figures, the random version is always better than the MPI version. As shown in Figures 7(a) and 7(b), the random version is also better than both sequential LMBFS and YAHSP, especially for hard instances. It also increases the coverage to 97.69%. Clearly, a promising lead to improve LMBFS is to find a way to bend the large plateau induced by the $h^{\mathcal{L}_{left}}$ heuristic.

Discussion This parallel algorithm expands many uninteresting metanodes generated by $\mathrm{deleteLM}$. Associated problems are usually hard to solve (because this generation of subproblem is less guided by the landmark graph), and can get a subplanner stuck whereas the PU should ideally explore more interesting metanodes received in the meantime. In other words, these expansions seem to lead the algorithm into uninteresting or dead-end branches of the search graph, which asks the question of the quality of the heuristic.

This also can be seen if recalling that LMBFS is based on exploration of landmarks orders. LMBFS tries, in a depth first search way, all possible total orderings before emptying the landmark graph (this is particularly enforced by the $h^{\mathcal{L}_{left}}$ heuristic and the lazy generation of section 3.3). But in the parallel version, this is not the case, as too early exploration of metanodes expanded by $\mathrm{deleteLM}$ occurs.

6 Conclusion and Future Works

This paper presents several contributions towards a new landmark-based planning algorithm. First, we propose a sound framework for a (meta)search based on the order of landmarks, given a landmark graph. We formalize the link between so-called metanodes and subproblems of the original planning problem, including restrictions on the allowed actions themselves. We give several operators that allow to explore different orders for using landmarks as subgoals, including skipping some. We also propose a first approach for evaluating heuristic values of such metanodes, or equivalently giving priorities to subproblems. We put everything together in a (deferred) best-first search algorithm, leading to a complete algorithm. We also propose a first parallelization scheme based on asynchronous distribution of open nodes among processing units. Last but not least, we implemented it and give some promising results.

From now on, several leads will be followed.

A key point for performance is the heuristic evaluation of metanodes, linked to the operators used for generation. For instance, nodes generated with nextLM are always expanded before other metanodes, which is not necessarily the best solution. Furthermore, the comparison of one LMBFS instance against several non-cooperative randomized instances showed that there is room for heuristic improvement. A first lead to obtain a better heuristic would be to also take into account the landmark subgoal but preliminary experiments showed us that this will not be straightforward.

Another point is the operators used. While $\mathrm{deleteLM}$ is very general, $\mathrm{cutParent}$ can be seen as a special case (a shortcut for a given sequence of $\mathrm{deleteLM}$, or said differently, a lookahead in the landmark graph itself), and other special cases may be very useful.

Another next step will focus on smarter parallelization schemes. As experiments showed, while we are able to scale up the number of explored nodes, this does not lead to finding a solution in less time. It seems that the processing units are "saturated" very quickly with hard problems coming from $\mathrm{deleteLM}$ (especially because lazy generation is not implemented in parallel algorithm), and so, with this simple parallelization scheme, the whole spirit of LMBFS, which is to try to follow as closely as possible the landmark graph while being complete, is no more enforced. There are some possibilities to do a lazy-like generation for the parallel version, but it would be intrinsically different from the lazy generation of sequential version. Another interesting lead is to select a few total orders at the beginning of the search, and try to find plans following these orders. If no solution can be found, we might be able to extract some data from these searches to create other interesting total orders, and so on.

References

Bibaï, J.; Savéant, P.; Schoenauer, M.; and Vidal, V. 2010. An evolutionary metaheuristic based on state decomposition for domain-independent satisficing planning. In *Proc. of ICAPS*, 18–25.

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

Fikes, R. E., and Nilsson, N. J. 1972. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2(3-4):189–208.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning, theory and practice*. Morgan-Kaufmann.

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

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.

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

Kishimoto, A.; Fukunaga, A.; and Botea, A. 2009. Scalable, parallel best-first search for optimal sequential planning. In *Proc. of ICAPS*.

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

Richter, S.; Helmert, M.; and Westphal, M. 2008. Landmarks revisited. In *Proc. of the 23rd AAAI Conference on Artificial Intelligence*, 975–982.

Romein, J. W.; Plaat, A.; Bal, H. E.; and Schaeffer, J. 1999. Transposition table driven work scheduling in distributed search. In *Proc. AAAI*.

Sebastia, L.; Onaindia, E.; and Marzal, E. 2006. Decomposition of planning problems. *AI Communications* 19:49–81.

Vidal, V.; Vernhes, S.; and Infantes, G. 2011. Parallel AI planning on the SCC. In *Proc. of the 4th Symposium of the Many-core Applications Research Community (MARC)*.

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

Vidal, V. 2011. YAHSP2: Keep it simple, stupid. In *Proc. of IPC'11*.

Zhu, L., and Givan, R. 2003. Landmark extraction via planning graph propagation. In *ICAPS Doctoral Consortium*, 156–160.

Generalization of the Landmark Graph as a Planning Problem

Vidal Alcázar
Universidad Carlos III de Madrid
Avenida de la Universidad, 30
28911 Leganés, Spain
valcazar@inf.uc3m.es

Abstract

Landmarks are logical formulæ over sets of propositions or actions that must be satisfied at some point in a planning task. The landmark graph, a proposed representation of the set of landmarks and their interactions, is built using the landmarks of the task and ordering relations between them. A formalization of the landmark graph in terms of a planning task has not been proposed yet, which makes it difficult to determine the significance of the landmark graph with respect to the original planning problem. In this work we propose a generalization of the landmark graph as an abstraction of the original problem and analyze its characteristics.

Introduction

Landmarks are currently one of the most important lines of research in automated planning, exemplified for instance by the success of landmark-based planners like LAMA (Richter and Westphal 2010). They were initially defined as disjunctive sets of propositions that had to be made true at some time in every solution plan to a problem (Hoffmann, Porteous, and Sebastia 2004), and later on its definition was extended to include both action landmarks (Richter and Westphal 2010) and conjunctive sets of propositions (Keyder, Richter, and Helmert 2010). Landmarks were also formalized in a framework that relates them to abstractions and critical paths (Helmert and Domshlak 2009), giving them a stronger theoretical background.

Although finding the complete set of landmarks is PSPACE-complete (Hoffmann, Porteous, and Sebastia 2004) and thus not tractable in most cases, current methods can efficiently compute a subset of the landmarks using a delete-relaxation representation of the problem. Partial orders between proposition landmarks can be obtained with these techniques too, which can be used to build the landmark graph. The landmark graph is an important part of many of the techniques that exploit landmarks, as the interactions and orders between landmarks are often as important as the landmarks themselves.

Despite the landmark graph providing important information, it still has some limitations in its current form. First, an informative total order may not be straightforwardly deduced from the set of partial orders, which is critical in applications like factored planning; second, landmarks may have to be achieved several times in every solution plan, a fact that is not taken into account due to the lack of negative interactions between landmarks other than the causal orderings (Hoffmann, Porteous, and Sebastia 2004); third, the exploitation of the cycles in the graph is still unclear and so current techniques usually just remove them.

To address these shortcomings, in this work a generalization of the landmark graph as a planning problem is proposed. The main motivation is creating an automatic landmark-based abstraction whose solution can capture the causal structure of the problem in a more accurate way than the landmark graph. The idea presented in this work is using the landmarks to build the set of propositions of the problem and transforming the achievers of those landmarks into the actions of the abstraction, adding information from landmark orderings and mutexes. This abstraction is a planning task itself, so it can be solved just like any other planning problem. Besides, it has several properties of its own that can be exploited when solving the original task.

Background

Automated planning is the task of finding a solution plan, composed by an ordered sequence of actions, so a set of goals can be achieved by applying the actions in the plan from the initial state. In this work only propositional planning is considered. A planning problem is a tuple *P=(S,A,I,G)* where *S* is a set of propositions, *A* is the set of grounded actions instantiated from the operators of the domain, $I \subseteq S$ is the initial state and $G \subseteq S$ the set of goal propositions. Actions are applicable when their preconditions are satisfied. The effects of an action make propositions true or false, which is commonly known as *adding* and *deleting* the propositions respectively. Thus, an action is defined as a triple $\{$*pre(a), add(a), del(a)*$\}$ in which $a \in A$ and *pre(a), add(a), del(a)* $\subseteq S$. Actions can have an associated cost; in this work only non-negative cost actions are used.

A landmark is a logical formula over either *S* (proposition landmark) or *A* (action landmark). Every solution plan must satisfy every action landmark. For each proposition landmark, at least one state in every sequence of states generated by a solution plan must satisfy it. *L* is the set of discovered landmarks of the problem.

The achiever of a proposition landmark $l \in S$ is an action $a \in A$ such that $l \in add(a)$. Informally, an achiever is an action that makes a given landmark true, or *adds* it. A late

achiever $a \in A$ is a regular achiever of a landmark $l \in S$ with the special condition that, in every plan executed from I, it is not possible for a to appear before l has been achieved at least once by some other action $a' \in A$.

Orderings between landmarks are relations between two proposition landmarks that represent the partial order in which they must be achieved. The landmark graph is a directed graph composed by the proposition landmarks of a problem and the orderings between them (Hoffmann, Porteous, and Sebastia 2004). There are the following orderings:

- Natural ordering: A proposition *a is naturally ordered before b* ($a <_{nat} b$) if *a* must be true at some time before *b* is achieved
- Necessary ordering: A proposition *a is necessarily ordered before b* ($a <_{n} b$) if *a* must be true one step before *b* is achieved
- Greedy-necessary ordering: A proposition *a is greedy-necessarily ordered before b* ($a <_{gn} b$) if *a* must be true one step before *b* when *b* is first achieved

There are also reasonable orderings between landmarks, but these orderings are unsound, that is, they do not have to be respected in every solution plan. Therefore, reasonable orderings will not be considered in this work.

Landmark Counting Heuristics

The admissible landmark counting heuristic h_{LA} formulated by Karpas and Domshlak (2009) is an admissible estimation of the distance to the goal computed by adding the cost of achieving the propositional (disjunctive) landmarks that are still needed. These landmarks may be landmarks that have not been achieved yet or that are required again. There are two versions of h_{LA}. The simplest version splits uniformly the cost of every achiever amongst the landmarks that they achieve (h^{uni}_{LA}), so the cost of each landmark is the minimum of these "split costs". The more complex version solves a Linear Programing problem per state that yields the maximum cost of achieving the required landmarks by selecting which achiever contributes to the cost of which landmark and by how much while keeping the estimate admissible (h^{opt}_{LA}).

h_{LA} depends on which landmarks are true in a given state $s \in S$ and on which landmarks have already been achieved. The latter depends on the path or paths that led from I to s, so this information must also be encoded. This makes h_{LA} a *path-dependent* heuristic, which means that h_{LA} is inconsistent and may yield different values for the same state s.

Generalization of the Landmark Graph as a Planning Problem

In this section the process of generating a new problem from the original problem using the information of the landmark graph is described.

Definition of the New Problem P_{abs}

In this section we will take into account only single proposition landmarks; other cases will be analyzed in a subsequent subsection. The generalization of the landmark graph as a planning problem is a tuple $P_{abs} = (S_{abs}, A_{abs}, I_{abs}, G_{abs})$ where S_{abs} is a set of propositions derived from the discovered proposition landmarks, A_{abs} is a set of grounded actions derived from the achievers of the propositions contained in S_{abs}, $I_{abs} \subseteq S_{abs}$ is the value of S_{abs} in the current state and $G_{abs} \subseteq S_{abs}$ the set of goal propositions.

The set of propositions S_{abs} is formed by two propositions per landmark $l \in L$, one proposition l_{abs}indicating whether the landmark is true or not and another proposition $achieved(l_{abs})$ indicating whether the landmark l_{abs} has been previously achieved.[1] All the propositions of the original problem $S \setminus L$ that are not landmarks are discarded. The set of actions A_{abs} are the actions in A that are landmark achievers, that is, at least one landmark proposition appears in its *add* effects. Formally, if $a \in A$ and $add(a) \cup L \neq \emptyset$, then a is used to derive new actions in A_{abs}. A priori, a single new action per achiever in A is created. However, due to the possibility of having disjunctive preconditions in the new actions, some actions in A_{abs} may have to be split into several ones. All the actions in A that do not add at least one landmark proposition are ignored. The goal propositions G_{abs} are the goal propositions G of the original problem, since a goal proposition is a landmark by definition. Additionally, the propositions that encode whether the original goal propositions have been achieved can also be added to G_{abs}, since a proposition being true necessarily implies that it has been achieved. Every goal state s must ensure that $\forall g \in G_{abs} : s(g) = \{true\}$.

This tuple is analogous to that of a regular planning task, which means that the new problem P_{abs} has all the properties of a regular planning problem. Besides, the resulting P_{abs} is an abstraction of the original problem P. The function α that maps a state $s \subseteq S$ to $\alpha(s) \subseteq S_{abs}$ consists of determining which landmarks are true and which landmarks have been previously achieved in s. Note that this depends not only on s but also on the path that led from I to s, so the information about which landmarks have already been achieved must be stored in a similar fashion as when computing h_{LA}. The transitions are defined by the relationship between A_{abs} and A, described in the following subsections.

Preconditions of the New Actions

Actions belonging to A_{abs} are applicable when the landmarks appearing in their preconditions have the required value. Three different cases define the preconditions of the actions:

- Preconditions derived from the orderings between the landmarks.

[1] A more concise representation using multi-valued state variables could be obtained in two ways: first, the variables could take the values of *true*, *false* or *false but previously achieved* instead of using two different variables to avoid redundancy, although this can lead to actions with disjunctive preconditions; second, landmarks belonging to the same variable in the original problem or that are otherwise mutually exclusive could be grouped in a single variable to indicate whether they are true or not.

- Preconditions of the achievers in the original problem that are landmarks.
- Preconditions obtained from actions labeled as *late achievers*.

Causal relationships between the landmarks are encoded by the orderings of the landmark graph. To represent this fact in the new task P_{abs}, new preconditions are added to the actions in A_{abs}. These preconditions enforce the partial orders between landmarks without hindering the computation of a valid total order. Each type of ordering implies a different set of new preconditions. In particular, the correspondence is as follows:

- Natural ordering: Every achiever of a given landmark l has a precondition for every natural ordering which represents that the landmarks naturally ordered before l must have been previously achieved. Formally, if $l \in add(a)$, $\forall l' <_{nat} l : achieved(l') \in pre(a)$.
- Necessary ordering: Every achiever of a given landmark l has a precondition for every necessary ordering which represents that the landmarks necessarily ordered before l must be true. Formally, if $l \in add(a)$, $\forall l' <_{n} l : l' \in pre(a)$.
- Greedy-necessary ordering: Every achiever of a given landmark l has a precondition for every necessary ordering which represents that the landmarks greedy necessarily ordered before l must be true or that l has been previously achieved. Formally, if $l \in add(a)$, $\forall l' <_{nat} l : (achieved(l) \vee l') \in pre(a)$.

Greedy-necessary orderings add a disjunctive precondition. Most planners do not support disjunctive preconditions, so actions with preconditions derived from greedy-necessary orderings must be split into several actions. In theory this can lead to having 2^n new actions per action that achieves landmarks with greedy-necessary orders, where n is the number of greedy-necessary orderings. However, due to the dominance that often appears between actions (described in a subsequent subsection), the actual number of new actions is at most $2^{n'}$ actions, where n' is the number of landmarks with greedy-necessary orderings achieved by the action. Although still exponential, in practice it seldom happens that an action achieves more than 2 such landmarks, which explains why there are no exponential blow-ups of the size of P_{abs} in the current benchmarks.

Preconditions can also be derived from the landmarks that appeared in the preconditions of the actions in A. Landmarks in the original problem P are kept as part of the new problem P_{abs}. Because of this, occurrences of landmarks in the actions of A must be taken into account when generating the actions in A_{abs}. Regarding the preconditions of every action in A_{abs}, this means that the preconditions of the original actions in A must also appear in the actions in A_{abs} if they are landmarks: if $a \in A$ was used to create $a' \in A_{abs}$, then $pre(a) \setminus L \subseteq pre(a')$. This overlaps with the preconditions obtained from greedy-necessary orderings and strictly dominates those obtained from necessary orderings, hence making the computation of the latter not necessary. This is so is because both types of orders are computed from common preconditions of the achievers of the landmark, already taken into account this way.

On the other hand, not all achievers are considered equal. There are two kinds of achievers: *first achievers*, which can achieve some landmark when it has never been achieved before, and *late achievers*, which can achieve a landmark only if it had been already achieved at some time before. Hence, a late achiever $a' \in A_{abs}$ derived from action $a \in A$ has an additional precondition $achieved(l) \in pre(a')$ per landmark $l \in add(a) \setminus L$ if a is a late achiever of l.

Effects of the New Actions

Effects of the original achievers and relations of mutual exclusivity between propositions (Haslum and Geffner 2000), typically called mutexes, determine the effects of the new actions. The positive effects of the actions, also known as *add* effects, make the propositions true. Similar to the propositions that are landmark in the preconditions of the actions in A, the *add* effects of the actions in A can be straightforwardly encoded in the new actions belonging to A_{abs}. Thus, every landmark proposition added by an action in A appears also as an *add* effect in the actions created from it in A_{abs}, whereas non-landmark propositions added by the action are ignored. Similarly, for every regular *add* effect of a landmark l, an *add* of $achieved(l)$ must be included. In summary, if $a \in A$ was used to derive $a' \in A_{abs}$ and landmark $l \in add(a) \setminus L$ then $l \wedge achieved(l) \in pre(a')$.

Before describing how the delete effects of the new actions are computed, the definition of mutex is given:

Definition 1 *A relation of mutual exclusivity between propositions (mutex) is a set of propositions $M = \{p_1, \ldots, p_m\}$ such that there is no reachable state $s \subseteq S$ such that all the propositions $p \in M$ are true in s.*

Negative effects, also known as *delete* effects, are linked to the notion of *e-delete* (Vidal and Geffner 2005).

Definition 2 *An action e-deletes a proposition if the proposition p must be false in every state after the action is executed.*

This means that if an action *e-deletes* a given proposition *p*, either it explicitly deletes *p* or it is only applicable in states in which *p* is false. There are three cases in which an action *e-deletes* a given proposition *p*: it deletes *p*; it has a set of preconditions mutex with *p* and does not add *p*; and it adds a set of propositions mutex with *p*.

Computing sets of mutexes of size $m > 2$ is usually impractical (Haslum and Geffner 2000), so in this work only sets of size $m = 2$ will be considered. Hence, to fulfill the second and third condition a single proposition mutex with *p* is enough. This way, every landmark that is *e-deleted* by an action $a \in A$ is added to the new action in A_{abs} as a *delete* effect. There is an exception to this rule, though: if an action $a \in A$ *e-deletes p* because it has a precondition mutex with *p* and does not add it (second condition) and that precondition is a landmark, it means that the new action in A_{abs} is only applicable *iff p is false*. In this case *p* will be always false after the execution of the new action in A_{abs}, which means there is no need to explicitly delete it.

Dominance Between the New Actions

Due to the abstraction of non-landmark propositions and the splitting of disjunctive preconditions, it is possible to have actions in A_{abs} that are equivalent to or dominated by other actions. In particular, an action with the same effects and with a cost greater or equal than another action needs not to be included in the definition of the problem as long as its preconditions are a superset of the preconditions of the dominating action. In terms of the formal definition of the planning task P_{abs}, $a \in A_{abs}$ dominates $a' \in A_{abs}$ if $a \neq a' \wedge add(a) = add(a') \wedge del(a) = del(a') \wedge pre(a) \subseteq pre(a') \wedge cost(a) \leq cost(a')$. This can be done after generating all the actions to avoid redundancy and obtain a smaller instance of P_{abs}.

Dominance between actions also explains why splitting actions when using greedy-necessary orderings leads to having fewer additional actions than expected. For example, let's assume that an action $a \in A$ adds a landmark $l \in S$ which is greedy-necessarily ordered after n landmark propositions $p_i \in S$. This means that the new action $a' \in A_{abs}$ will have n disjunctive preconditions of the form $\{achieved(l) \vee p_i\}$. If a' is split in two for every disjunctive precondition, a total of 2^n new actions will be created. However, every time an action is split, either $achieved(l)$ or p_i will be added to the preconditions; if $achieved(l)$ is added and it is already present, no new action is needed, and if p_i is added and $achieved(l)$ is already a precondition, it will be forcibly dominated by another action with $achieved(l)$ as precondition. In the end, only 2 actions are generated per added landmark with greedy-necessary orders, one with $achieved(l)$ as additional precondition and another one with $p_1 \wedge ... \wedge p_n$ as additional preconditions.

Conjunctive Landmarks, Disjunctive Landmarks and Action Landmarks

Landmarks are not restricted to the single proposition case. When generalizing the landmark graph this has to be taken into account. Particular cases are treated in the following way:

- Action landmarks: They can be safely ignored, as their preconditions and effects are landmarks themselves. A possible optimization is allowing only the action landmark as achiever if the landmarks derived from the effects of the action are not ordered after any landmark other than the preconditions of the action landmark.
- Conjunctive landmarks: All the propositions that form the set can be treated as independent landmarks. The achievers of landmarks ordered after a conjunctive landmark will have all the propositions in the set as preconditions.
- Disjunctive landmarks: All the propositions that form the set can be treated as independent landmarks. Only one of the propositions that form the set must have the required value for an action $a \in A_{abs}$ that has the disjunctive landmark as precondition to be applicable. This is done by splitting every such action into several actions, one per proposition in the set, and including that proposition as precondition. This can lead to an exponential number of actions, so a more efficient way of dealing with disjunctive landmarks could improve the compactness of the problem.

Properties of the New Problem

The new planning problem P_{abs} has all the properties of a regular planning problem in a propositional representation. Besides, due to its relationship with the original problem P, P_{abs} has several additional characteristics:

- All the propositions being true and having been achieved are landmarks, unless they were generated from a disjunctive landmark. If this is the case, they will be part of a disjunctive landmark if a landmark discovery method that can find disjunctive landmarks of size equal or greater than the original disjunctive landmarks is used.
- The optimal cost to the goal in the new problem h^*_{abs} is an admissible estimation of the cost to the goal in the original problem h^*, since P_{abs} is an abstraction of P. The proof is as follows: P_{abs} is a projection of P except for the added propositions of the type $achieved(l)$ for $l \in L$. If a problem p' is a projection of p, then $h^*_{p'} \leq h^*_p$. Hence, the only source of inadmissibility can be the propositions of the type $achieved(l)$. However, these propositions are added to enforce the orderings of the landmark graph. These orderings are respected by any optimal solution of P, so the presence of these propositions cannot make that $h^*_{p'} > h^*_p$.
- h^*_{abs}, when being used as a heuristic in a forward search algorithm, is *not* necessarily a consistent estimation of h^*, as I_{abs} depends on the path that led to the current state in P. That is, h^*_{abs} as an admissible estimation of the cost to the goal is a path-dependent heuristic. For the proof, we refer the reader to (Karpas and Domshlak 2009).
- h^*_{abs} dominates the admissible landmark counting heuristic h_{LA} (Karpas and Domshlak 2009) and is not bounded by h^+, the optimal cost of the delete-relaxation version of the original problem. The proof is straightforward: on one hand, if $S = L \subset S_{abs}$ and at least one landmark must be reachieved after being deleted in every optimal plan, then $h^*_{abs} = h^*$ and $h^*_{abs} > h^+ \geq h_{LA}$; on the other hand, if P_{abs} is relaxed by removing orderings and *deletes*, then $h^*_{abs} = h^{opt}_{LA}$. There is no further relaxation other than projecting landmarks away that may make h^*_{abs} lower, so h_{LA} is a lower bound of h^*_{abs} if all the landmarks are considered when building P_{abs}.
- The optimal cost of the delete-relaxation version of the new problem h^+_{abs} still dominates h_{LA}, even with optimal cost partitioning (h^{opt}_{LA}). This is because landmark preconditions of the achievers add additional constraints that may make the cost of the optimal plan go beyond h_{LA}'s value. An example is shown in the delete-free problem appearing in Figure 1: if $I = \{l\}$ and $G = \{l', l''\}$, h_{LA} would assume a cost of 1 for both l' and l'' for a total cost of 2; h^+_{abs} however takes into account that to achieve l' with a cost of 1 l'' must be achieved first, so h^+_{abs} would yield a value of 3.

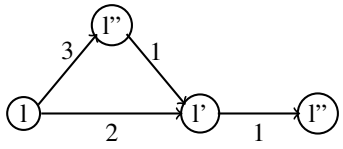


Figure 1: Example in which h^+_{abs} yields a higher value than h_{LA}.

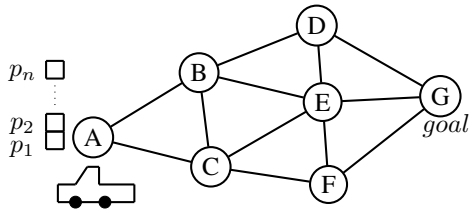


Figure 2: Initial state of the Logistics instance

Example

In this section we show an example of how P_{abs} is constructed. The example is taken from the Logistics domain, although in this case trucks can only carry one package at a time, encoded by the predicate *(empty ?t - truck)*. The initial state is shown in Figure 2. There is a single truck that can move through a graph of different locations. The truck is initially at location *A*, where there is an arbitrary number of packages. The packages can be loaded into the truck if the truck is empty and dropped at another location. The goal is carrying all the packages to the goal location *G*. The single proposition landmarks of this problem are:

- The truck at the initial location: *(at truck A)*
- The truck at the final location: *(at truck G)*
- The truck empty: *(empty truck)*
- Every package p_i at the initial location: *(at p_i A)*
- Every package p_i in the truck: *(in p_i truck)*
- Every package p_i at the final location: *(at p_i G)*

The actions of the planning problem would be the following (note that every time a landmark is made true, the proposition that encodes whether it has been achieved before or not is unconditionally made true as well):

- Move the truck to the initial location *A*: *pre*={*achieved(at truck A)*}, *add*={*(at truck A)*}, *del*={*(at truck G)*}
- Move the truck to the final location *G*: *pre*={*achieved(at truck A)*}, *add*={*(at truck G)*}, *del*={*(at truck A)*}
- Load a package p_i at the initial location *A*: *pre*={*(at truck A),(at p_i A),(empty truck)*}, *add*={*(in p_i truck)*}, *del*={*(at p_i A),(empty truck)*}
- Load a package p_i at the final location *G*: *pre*={*(at truck G),(at p_i G),(empty truck)*}, *add*={*(in p_i truck)*}, *del*={*(at p_i G),(empty truck)*}
- Drop a package p_i at the initial location *A*: *pre*={*(at truck A),(in p_i truck)*}, *add*={*(at p_i A),(empty truck)*}, *del*={*(in p_i truck)*}
- Drop a package p_i at the final location *G*: *pre*={*(at truck G),(in p_i truck)*}, *add*={*(at p_i G),(empty truck)*}, *del*={*(in p_i truck)*}

In this example all the preconditions are taken from the landmark preconditions of the original actions in A with the exception of the *move* actions. Similarly, apart from the *move* actions all the actions in A_{abs} can only be generated from a single action in A. The *move* action that achieves *(at truck A)* can be generated from two actions in A; however, both actions have the same preconditions and effects, so one arbitrarily dominates the other, which explains why there is only one action in A_{abs} that achieves *(at truck A)*. This action has *achieved(at truck A)* as precondition because the original actions in A are *late achievers* of *(at truck A)*. No other precondition appears as no precondition of the original actions is a landmark and *(at truck A)* is not ordered after any other landmark. The same dominance occurs with the action that achieves *(at truck G)*, although in this case the precondition is generated from the natural ordering existing between *(at truck A)* and *(at truck G)*. As a side note, every landmark of the form *achieved(s)* that is true in I_{abs} is a static fact, so it can be safely discarded, as with *achieved(at truck A)* in the example.

The key feature that regular delete-relaxation approaches do not capture in this case is achieving and deleting *(at truck A)* and *(at truck G)* alternatively. In this case, if *n* encodes the number of packages, the cost of the optimal solution in the original problem is $h^* = 8(n-1)+5$ and in the abstraction is $h^*_{abs} = 4(n-1)+3$. On the other hand, the value of both h^+ and h^{lmcut} (Helmert and Domshlak 2009) in the original problem is $h^+ = h^{lmcut} = 2n+3$ and the value of the admissible landmark heuristic with optimal cost partitioning h^{opt}_{LA} is in both the original and the delete-relaxation problem $h^{opt}_{LA} = 2n+1$.

Potential Applications of P_{abs}

Although beyond the scope of this work, it is interesting to analyze the possible applications of the resulting P_{abs} in regards to task solving. The main consideration when exploiting the information provided by P_{abs} is that P_{abs} is a planning problem itself and so it is PSPACE-complete (Bylander 1994). This means that in many cases solving it (optimally or otherwise) may not be tractable. However, the size of P_{abs} is in most cases smaller than the original task P, so if a planner is not able to solve P_{abs} it is highly unlikely that it would be able to solve P.

One of the first applications of landmarks, factored planning (Hoffmann, Porteous, and Sebastia 2004), seems to be a technique that would greatly benefit from this approach. As opposed to using the first layer of unachieved or required again landmarks as a disjunctive intermediate goal, suboptimally solving P_{abs} can be used to obtain a total order of single landmarks, which can be used as subgoals to partition P. Apart from being able to exploit cycles between landmarks

and offering a finer partitioning, this approach also has the advantage of not being subject to arbitrary decisions when dealing with disjunctive subgoals (for instance, when partitioning the Sussman's anomaly, a layered partitioning can lead to either finding the solution with the minimum number of expansions or to exploring the whole search space depending on how ties are broken).

Inspired by the potential of the serialization of the landmark graph, the recent planner PROBE(Lipovetzky and Geffner 2011) builds probes that try to achieve the goal with little to no search by estimating a total order of the landmarks and trying to achieve them in a greedy way. However, the way the probes are built is not based on any theoretical scheme and it has the additional problem of not allowing actions that delete a landmark to be used if that that landmark is a goal or is still needed (which is often the case when cycles between landmarks appear). Solving P_{abs} suboptimally yields a total order that may be more informative and that may capture stronger interactions between landmarks.

The use of P_{abs} to derive heuristics to be used in P has already been mentioned. Nevertheless, a naive approximation like solving P_{abs} at every state to obtain the heuristic estimate would be intractable in most cases, so more synergistic techniques are required. Since P_{abs} is an abstraction of P, hierarchical search algorithms like Hierarchical A* (Holte et al. 1996) can be an interesting alternative. When using HA*, P_{abs} is solved from I_{abs} to obtain a heuristic estimate and the expanded nodes are kept hoping that subsequent queries can be cached, which allows to obtain $h(s)$ with no search. If cache hits happen often enough, there will be a trade-off between the time spent solving P_{abs} and the time saved from computing $h(s)$, which may result in an increased efficiency. Furthermore, P_{abs} can be solved both optimally, superseding the admissible h_{LA}, or suboptimally, superseding LAMA's heuristic (Richter and Westphal 2010).

Lastly, the total order obtained from solving P_{abs} can also be used in combination with other search paradigms. For instance, local search planners like LPG (Gerevini and Serina 2002) are often ill-suited to solving highly sequential domains, particularly if they involve cycles. In this case, the solution to P_{abs} can be used as a seed so a skeleton of the plan is provided and the local search only has to fill the "gaps" between the landmarks.

Experimentation

As any other landmark-based technique, the proposed generalization of the landmark graph is highly dependent on the number and relevance of the landmarks found with current methods. For instance, in some domains only a few landmarks other than the propositions true in I and the goals in G are found, in which case landmark-based techniques perform poorly. Furthermore, one of the main advantages of P_{abs} over most ways of exploiting landmarks resides in the fact that it is able to account for negative interactions such as delete effects and mutexes. In particular, P_{abs} appears to capture the structure of the problem best when cycles are a key part of the domain. If such negative interactions are not representative of the planning task, using P_{abs} may not be more advantageous than using already existing landmark techniques. Because of this, the performance of any technique based on P_{abs} will depend on the number of landmarks and the features of the domain they represent.

Because of the aforementioned reasons, three different cases arise in problems with meaningful landmarks:

- Almost all the propositions are landmarks: the abstraction is very similar to the original problem and thus it is hard to obtain a balanced trade-off between the information it provides and the difficulty of solving P_{abs}. In this case directly solving P is the simplest alternative, though solving P_{abs} may not be necessarily worse, since the information obtained from P_{abs} probably would allow solving P with almost no search.
- There are few necessary orderings and/or landmark preconditions of the achievers: when solving the abstraction the landmarks do not need to be reachieved, so only a rather arbitrary total order of the landmark graph is obtained. Hence, in most cases the cost of the solution is equal or only slightly higher than h_{LA} with optimal cost partitioning. Besides, partitionings of P derived from the obtained total order will not offer much more information than the regular landmark partitioning using disjunctive subgoals.
- There are fairly numerous necessary orderings and/or landmark preconditions of the achievers: in these cases an informative plan is obtained with a cost higher than h_{LA} while still being solvable. As described before this is often the case in which resources or similarly "consumed" landmarks enforce cycles in the landmark graph, cases in which other techniques often do not fare that well.

Knowing this, it may be relatively simple to predict whether P_{abs} will be useful in the actual resolution of the planning task or not. This work is mainly theoretical, and hence P_{abs} was not used as a way of improving the efficiency of a given planner. Nevertheless, in order to assess its usefulness in practice, some experimentation has been done. First, P_{abs} was generated in a broad range of domains from the International Planning Competition so the relative size of P_{abs} compared to P could be estimated. The planner used to ground the instances in both cases was Fast Downward (Helmert 2006), and the chosen measure of size is the number of actions and fluents of each instance after preprocessing, that is, the cardinality of A, A_{abs}, S and S_{abs} in every instance. The reason why the number of actions was chosen is because it gives an idea of the size of the tasks and because generating P_{abs} often requires action splitting, whose impact can be measured by counting the final number of actions. The number of propositions $|S_{abs}|$ can be known a priori just by computing the landmarks of P, although a significant subset of the propositions derived from the landmarks of P may be static in P_{abs}, which means that $|S_{abs}|$ is often smaller than twice the number of landmarks. Table 1 shows the sum of the cardinality of A, A_{abs}, S and S_{abs} of every instance in each domain. All the orderings were used and dominance between actions was enabled. Only propositional landmarks were used, since disjunctive landmarks may easily lead to an exponential increase in the number of actions.

Domain	$\|A\|$	$\|A_{abs}\|$	$\|S\|$	$\|S_{abs}\|$	h^{lmcut}	h^{opt}_{LA}	h^{uni}_{LA}
airport (50)	144963	125067	157592	46706	0.92	1.00	1.00
barman-opt11 (20)	13264	11004	4604	1200	0.64	1.97	2.50
blocks (35)	7490	7434	4826	2549	1.44	1.44	1.44
depot (22)	68894	51392	9423	1934	0.46	1.00	1.00
driverlog (20)	53494	5926	6007	542	0.46	1.00	1.00
elevators-opt08 (30)	18520	7574	3360	607	0.51	1.07	1.07
elevators-opt11 (20)	11450	4619	2097	571	0.50	1.09	1.09
floortile-opt11 (20)	9188	6036	3578	1008	0.76	1.06	1.06
freecell (80)	1071066	663857	23419	13286	2.42	1.03	1.03
grid (5)	38808	18010	3373	185	0.88	1.01	1.01
gripper (20)	3720	1880	2380	960	0.52	1.00	1.00
logistics00 (28)	6972	2380	3429	2409	1.09	1.09	1.09
logistics98 (35)	501186	13491	82687	3203	1.09	1.09	1.09
miconic (150)	189100	125128	13950	19964	1.03	1.03	1.03
mprime (35)	567960	2977	17796	139	0.32	1.00	1.00
mystery (30)	217800	3634	13066	164	0.36	1.01	1.01
nomystery-opt11 (20)	72522	59865	4434	1008	1.15	1.09	1.09
openstacks (30)	213470	212544	10634	11593	1.69	1.00	1.00
parcprinter-opt08 (30)	9066	2933	6139	4159	0.91	1.00	1.00
parcprinter-opt11 (20)	5096	1732	3993	2591	0.91	1.00	1.00
pathways-noneg (30)	40595	7126	13119	2596	0.40	1.00	1.00
pegsol-opt08 (30)	5346	5346	2920	2003	1.00	1.41	1.41
pipesworld-notankage (50)	187388	73935	44594	1649	1.03	1.21	1.21
pipesworld-tankage (50)	1135917	742542	28027	1649	1.06	1.20	1.20
psr-small (50)	14546	11751	2158	572	1.65	1.65	1.65
rovers (40)	231653	45064	29324	2705	0.51	1.04	1.04
satellite (36)	3709130	34163	30479	6192	0.55	1.01	1.01
scanalyzer-opt08 (30)	1145836	733474	4680	1691	1.05	1.07	1.23
scanalyzer-opt11 (20)	631288	420820	3088	1010	1.05	1.07	1.18
sokoban-opt08 (30)	12674	5657	8518	1129	0.59	1.05	1.09
sokoban-opt11 (20)	7166	3332	5306	706	0.56	1.08	1.11
tidybot-opt11 (20)	384018	114737	11476	662	0.59	1.09	1.09
tpp (30)	281351	59833	18807	1564	0.57	1.02	1.02
transport-opt08 (30)	105888	4200	6800	390	0.02	1.00	1.00
transport-opt11 (20)	35216	2360	2886	250	0.02	1.00	1.00
trucks (30)	442262	33896	6964	2065	0.76	1.04	1.04
visitall-opt11 (20)	3520	2688	2516	2259	1.25	1.00	1.00
woodworking-opt08 (30)	27835	22058	5677	2729	0.87	1.02	1.03
woodworking-opt11 (20)	18175	14267	3805	1837	0.90	1.03	1.04
zenotravel (20)	140433	9138	4518	428	0.45	1.01	1.01

Table 1: Comparison between P and P_{abs}: task size and heuristics.

Results show that the relative cardinality of A_{abs} varies greatly from domain to domain. At one extreme, in *pegsol* $|A_{abs}|$ has the same value as $|A|$; at the other, in some transportation domains like *transport* or *zenotravel* $|A_{abs}|$ is comparatively rather small because of the small number of single proposition landmarks found in those domains. This is confirmed by the low value of S_{abs} in those domains in which $|A_{abs}|$ is significantly lower than $|A|$. There are no domains in which $|A_{abs}| > |A|$, although in two domains (*miconic* or *openstacks*) $|S_{abs}| > |S|$ because of the additional propositions added to represent when a landmark has been achieved.

Additionally, the geometric mean of the ratio between h^*_{abs} and $h^{lmcut}/h^{opt}_{LA}/h^{uni}_{LA}$ is shown for every domain. Only instances in which P_{abs} could be solved optimally under a time limit of 300 seconds were included. A ratio between h^*_{abs} and h^{opt}_{LA} close to 1.00 means that using P_{abs} probably has little potential in those domains; a higher ratio, as in *barman*, *blocks*, *pegsol*, *pipesworld* (both versions) and *psr-small* means that using P_{abs} may be promising. Also, note that disjunctive landmarks were not used; if disjunctive landmarks had been used, the ratio could have been higher as well in domains with symmetric resources, like *Gripper* and *Logistics*. The comparison with h^{lmcut} shows that although h^{lmcut} is on average closer to h^*, it varies substantially from domain to domain.

Related Work

Using landmarks in factored planning (Hoffmann, Porteous, and Sebastia 2004) was the first attempt to exploit the information contained in the landmark graph. However, it did not consider the fact that landmarks can be deleted nor the negative interactions between them. A subsequent work that employed mutexes to build layers of conjunctive landmarks addressed the latter (Hoffmann, Porteous, and Sebastia 2004). Although more informed than the original partitioning in disjunctive subgoals, computing the new layer was sometimes too inefficient, apart from offering no insight in terms of the formal properties of the problem.

Another work exploited inconsistencies between landmarks and local interactions between achievers and other landmarks to compute the minimum number of states required to satisfy given sets of landmarks (Porteous and Cresswell 2002), which in turn could be used to obtain a lower bound on the number of times a landmark must be achieved. This information captures the fact that in a sequential plan landmarks may have to be deleted even if they are needed again later and can be used in a cost-partition scheme to derive admissible heuristics with properties similar to the ones mentioned in this work. However, this approach is difficult to define formally due to its procedural nature and offers potentially less information than the generalization of the landmark graph.

Another closely related work proposed a SAT compilation of the landmark graph (Alcázar and Veloso 2011). The compilation generates a SAT problem which is solved using a regular SAT solver in order to obtain a more informative version of the landmark graph. Orderings of the landmark graph and londexes between landmarks (Chen, Xing, and Zhang 2007) are used to create the constraints of the problem. The new version of the landmark graph obtained from the solution of the SAT problem encodes the time steps in which landmarks may be needed, which implicitly captures the fact that some landmarks may have to be achieved several times and which gives a possible total order of the set of landmarks. However, this approach works with a parallel scheme and does not take into account the achievers of the landmarks, so it cannot consider costs nor offer reliable information about the possible total orders.

Finally, encoding achieved landmarks as new variables in P was also proposed (Domshlak, Katz, and Lefler 2012). This work focused on obtaining more informative abstractions to derive admissible heuristics, whereas here we make a preliminary study of the potential of the landmark abstraction in isolation. Both works have important points in common, so a deeper comparison between both approaches (like checking how close a projection of the enriched P is to P_{abs}) may be fundamental for future developments.

Conclusions and Future Work

In this work a generalization of the landmark graph as a planning problem has been presented. So far the main contribution of the discussed approach is a theoretical one, although several ways of exploiting the obtained abstraction have been proposed. This generalization bridges the gap towards the integration of landmarks in a common framework along with abstractions and heuristics in automated planning. Furthermore, the formal properties that relate it to the original problem have been analyzed.

It remains an open question whether there are additional ways of exploiting or improving the abstraction. On the one hand, it is unclear whether some characteristics of the abstraction, like symmetries or solvability features, can be extrapolated to the original problem. On the other hand, the costs encoded in the generalization could be improved by using cost-partitioning schemes such as the ones used in additive admissible heuristics. For example, the costs of the actions could be substituted by lower bounds on the cost of achieving a given landmark. This is actually already possible in a simple way: if, through the path to a landmark, there are no positive interactions, the cost of the achieved landmark can be substituted by the cost of the path from the closest landmark to the achieved landmark. This is trivial when using actions whose preconditions and effects affect a single invariant, like the *move* operator in the example presented in this work. In this case, the cost of the *move* actions can be the cost of getting to the achieved landmark from the closest landmark, which interestingly enough would make that $h^*_{abs} = h^*$. Other approaches could extend to more general cases, so this line of research may be an interesting follow up of this work.

Acknowledgments

The author would like to thank the whole research group for Foundations of Artificial Intelligence at Freiburg University and in particular Malte Helmert, without whom this

work would not have been possible. The author would like to thank Michael Katz too for his insightful comments about the related work. This work has been partially supported by the Spanish government through MICINN projects TIN2008-06701-C03-03 (as a FPI grant) and TIN2011-27652-C03-02.

References

Alcázar, V., and Veloso, M. 2011. A SAT compilation of the landmark graph. In *COPLAS*, 47–54.

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

Chen, Y.; Xing, Z.; and Zhang, W. 2007. Long-distance mutual exclusion for propositional planning. In *IJCAI*, 1840–1845.

Domshlak, C.; Katz, M.; and Lefler, S. 2012. Landmark-enhanced abstraction heuristics. *Artificial Intelligence* 189(0):48 – 68.

Gerevini, A., and Serina, I. 2002. LPG: A planner based on local search for planning graphs with action costs. In *AIPS*, 13–22. AAAI.

Haslum, P., and Geffner, H. 2000. Admissible heuristics for optimal planning. In *AIPS*, 140–149.

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

Helmert, M. 2006. The Fast Downward planning system. *J. Artif. Intell. Res. (JAIR)* 26:191–246.

Hoffmann, J.; Porteous, J.; and Sebastia, L. 2004. Ordered landmarks in planning. *J. Artif. Intell. Res. (JAIR)* 22:215–278.

Holte, R. C.; Perez, M. B.; Zimmer, R. M.; and MacDonald, A. J. 1996. Hierarchical A*: Searching abstraction hierarchies efficiently. In *AAAI/IAAI, Vol. 1*, 530–535.

Karpas, E., and Domshlak, C. 2009. Cost-optimal planning with landmarks. In Boutilier, C., ed., *IJCAI*, 1728–1733.

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

Lipovetzky, N., and Geffner, H. 2011. Searching for plans with carefully designed probes. In Bacchus, F.; Domshlak, C.; Edelkamp, S.; and Helmert, M., eds., *ICAPS*, 154–161. AAAI.

Porteous, J., and Cresswell, S. 2002. Extending landmarks analysis to reason about resources and repetition. In *PLANSIG*.

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

Vidal, V., and Geffner, H. 2005. Solving simple planning problems with more inference and no search. In *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming (CP'05)*, volume 3709 of *LNCS*, 682–696. Sitges, Spain: Springer.

An Admissible Heuristic for SAS$^+$ Planning Obtained from the State Equation

Blai Bonet
Departamento de Computación
Universidad Simón Bolívar
Caracas, Venezuela
bonet@ldc.usb.ve

Abstract

Domain-independent optimal planning has seen important breakthroughs in recent years with the development of tractable and informative admissible heuristics, suitable for planners based on forward state-space search. These heuristics allow planners to optimally solve an important number of benchmark problems, including problems that are quite involved and difficult for the layman. In this paper we present a new admissible heuristic that is obtained from the state equation associated to the Petri-net representation of the planning problem. The new heuristic, that does not fall into one of the four standard classes, can be computed in polynomial time and is competitive with the current state of the art for optimal planning, as empirically demonstrated over a large number of problems, mainly because it often shows an improved quality-to-cost ratio. The new heuristic applies to SAS$^+$ planning tasks with arbitrary non-negative action costs.

1 Introduction

Domain-independent planning deals with the development of planners for solving unknown input problems that are specified in a high-level description language. Domain-independent means that the planner has not other information about the problem than the one it can infer from its description. The general interest is on building "satisficing" planners whose task is to compute a valid solution, while "optimal" planners are required to output solutions of minimum cost.

Recent years have witnessed a remarkably progress in optimal planning in terms of the type and size of problems that can be dealt with. Current state-of-the-art optimal planners perform forward search in state space using A* with an admissible heuristic in order to meet the optimality requirement, and thus the basic difference between optimal planners is the heuristics that are used to guide the search.[1]

[1]More recently, best optimal planners correspond to systems that use a 'portfolio' of heuristics that are scheduled according to features of the input problem. However, a portfolio is a collection of base heuristics and hence its intrinsic limitations is a function of the heuristics in the portfolio.

Helmert and Domshlak [2009] observed that most of the well-known heuristics for optimal (and also for satisficing) planning fall in one of four categories: delete-relaxation heuristics that try to estimate the optimal cost h^+ of the delete-relaxed problem [Bonet and Geffner, 2001; Hoffmann and Nebel, 2001; Coles *et al.*, 2008], abstraction heuristics that correspond to the optimal costs of a simplified yet informative abstraction of the problem [Edelkamp, 2001; Haslum *et al.*, 2007; Helmert *et al.*, 2007; Katz and Domshlak, 2008], heuristics based on critical paths such as the family h^m [Haslum and Geffner, 2000], and landmark heuristics that compute sets of facts or actions that every plan must achieve or execute from which the cost of an optimal plan can be estimated [Hoffmann *et al.*, 2004; Richter and Westphal, 2010; Karpas and Domshlak, 2009; Helmert and Domshlak, 2009; Bonet and Helmert, 2010]. Currently, the most successful heuristic is the LM-cut heuristic [Helmert and Domshlak, 2009] that approximate h^+ quite well on some domains, but that can also be thought as a cost-partitioning heuristic or as a landmark heuristic. LM-cut is always bounded by h^+ and hence, ineffective at assessing the need to apply a fixed action multiple times for reaching the goal from a given state. On the other hand, abstraction heuristics are not delete-relaxation heuristics and have the potential to overcome this and other limitations [Helmert and Domshlak, 2009], yet to this date, the full potential of such heuristics have not been realized in a domain-independent manner and thus do not stand out as the best general heuristics.

In this paper we define a new heuristic for optimal planning that targets problems specified in SAS$^+$ involving multi-valued variables, arbitrary action costs, but without conditional effects. This is a class of problems that subsumes STRIPS and is as general as the classes of problems handled by other state-of-the-art heuristics. The heuristic is simple to formulate and is related to the so-called state equation for the Petri-net that is obtained from the planning problem in a standard way. For computing it, though, a small and simple linear programming problem needs to be solved. Despite the overhead involved in this computation, the new heuristic is competitive with the best current heuristics, and in some domains, it is able to solve more problems than any other heuristic. This is due to an improved quality-to-cost ratio on some domains, as observed in our experiments. We also mention how the new heuristic can be improved in three different

and independent ways by either automatically reformulating the problem instance, by analyzing the resulting Petri-net of the problem, or by incorporating information from the landmarks of the problem into the linear program that defines the heuristic. The contribution of the paper thus sets a general framework for obtaining heuristics that appears to be quite promising.

The paper is organized as follows. In the next section, we present the planning framework and the concepts of Petri nets that are related to the heuristic. The new heuristic and its properties are then defined, while the experimental study and a discussion appear at the end of the paper.

2 Preliminaries

2.1 SAS$^+$ Planning

A SAS$^+$ problem with action costs is a tuple $P = \langle V, A, s_{init}, s_G, c\rangle$ consisting of a set of multi-valued variables V, where each variable $X \in V$ has a finite domain D_X, together with a finite set A of actions, an initial state s_{init}, a goal description s_G, and non-negative action costs $c : A \to \mathbb{N}$. A state in SAS$^+$ is a V-valuation while a partial state is a partial valuations on variables; for a partial valuation ν, we use $Vars(\nu)$ to denote the set of variables that ν assigns a value, and $\nu|_W$ to denote the partial valuation obtained by restricting ν to the set of variables $W \cap Vars(\nu)$. The initial state s_{init} is a complete V-valuation while s_G is a partial valuation that tells what are the requirements that any valid plan must achieve. Each action $a \in A$ consists of a precondition and a postcondition, both corresponding to partial states and denoted by pre(a) and post(a) respectively, that specify the conditions that must hold for the action to be executable and the conditions that will hold after the execution of the action.

The state that results after applying the action a in state s is denoted by $res(a, s)$ (always assuming that a is applicable at s), and the state that results after applying a sequence $\langle a_1, \ldots, a_n\rangle$ of actions in state s is denoted by $res(\langle a_1, \ldots, a_n\rangle, s)$ (always assuming that a_k is applicable at $res(\langle a_1, \ldots, a_{k-1}\rangle, s)$ for $1 \leq k < n$). We say that a partial valuation ν is reachable iff there is a sequence $\pi = \langle a_1, \ldots, a_n\rangle$ of actions applicable at s_{init} such that $res(\pi, s_{init})|_{Vars(\nu)} = \nu$. In particular, a state s is reachable if there is a sequence π such that $res(\pi, s_{init}) = s$.

A plan for problem P is a sequence π of actions that reaches s_G, while its cost is the sum of the costs of the actions in π. A plan π is optimal iff it is a plan of minimum cost.

2.2 Petri Nets

A Petri net (also called place/transition or P/T net) is a type of directed graph that represents a transition system in factored form, together with an initial state called the initial marking. Petri nets had been used extensively to model and reason about concurrent and distributed systems, verification, dataflows and workflows, formal languages, etc., and more recently also automated planning [Hickmott *et al.*, 2007; Hickmott and Sardiña, 2009]. Murata [1989] gives an excellent introduction to Petri nets and the most important techniques for analyzing them.

A P/T net is a directed bipartite graph between two types of nodes called places and transitions, while a marking assigns to each place a non-negative integer. If a marking assigns the integer $k \geq 0$ to a place p, we say that p is marked with k tokens. Transitions represent events that map markings into markings by moving tokens from one place to another. Formally, a P/T net is a tuple $PN = \langle P, T, F, W, M_0\rangle$ where $P = \{p_1, p_2, \ldots, p_m\}$ and $T = \{t_1, t_2, \ldots, t_n\}$ are the finite sets for places and transitions, $F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs that is often called the flow relation of the net, $W : F \to \mathbb{N}^*$ assigns positive weights to arcs in the flow, and $M_0 : P \to \mathbb{N}$ is the initial marking of the net. The structure of the net is the tuple $N = \langle P, T, F, W\rangle$ made of the first 4 elements and represent the transition system detached of the initial marking.

The initial marking M_0 of the net evolves as transitions becomes enabled and fired according to the following rules:

1. Marking M *enables* transition t if for each arc $(p, t) \in F$, the number of tokens consumed by t from place p is at most the number of tokens assigned by M at p; i.e., $W(p, t) \leq M(p)$.
2. *Firing* an enabled transition t in marking M yields a new marking M' that results of consuming (subtracting) $W(p, t)$ tokens from place p, for each $(p, t) \in F$, and producing (adding) $W(t, q)$ tokens to place q, for each $(t, q) \in F$.

If the transition t is enabled by marking M and yields marking M' when fired, we write $M[t\rangle M'$. In this case, M' is the marking that assigns $M'(p) = M(p) + W(t, p) - W(p, t)$ tokens at place p.[2] Inductively, we write $M_1[u_1u_2 \cdots u_k\rangle M_{k+1}$ when transition u_i is enabled by marking M_i and yields marking M_{i+1} given by $M_{i+1}(p) = M_i(p) + W(t, p) - W(p, t)$, for each place $p \in P$ and $1 \leq i \leq k$. An *ordinary* net is one in which $W(\cdot, \cdot)$ ranges over $\{0, 1\}$. Throughout the rest of the paper, we only consider ordinary nets.

If there is a sequence of transitions $\sigma = u_1 \cdots u_k$ such that $M[\sigma\rangle M'$, we say that M' is reachable from M through the *firing sequence* σ. A basic decision problem for P/T nets is to determine whether a given marking M is reachable from the initial marking M_0. This is the *reachability* problem for P/T nets. Another related problem is the *coverability* problem that asks to determine whether for a given marking M, there is a marking M' reachable from M_0 such that $M(p) \leq M'(p)$ for each place p; in the latter case, we just write $M \leq M'$.

If every reachable marking M from M_0 (including M_0 itself) assigns at most k tokens to every place, we say that the net is k-bounded. The case for $k = 1$ is frequent and 1-bounded is commonly referred to as 1-safe or just safe.

We now present the *state equation* that expresses the relation between two markings M and M', through a transition t when $M[t\rangle M'$ holds, with an algebraic equation over vectors. For this, let us define the *incidence matrix* $A = (a_{ij})$ for the flow relation; it is a matrix of dimension $n \times m$ (where n and

[2] Strictly speaking, this is an abuse of notation as W may be undefined on some pairs, yet this is fixed by defining $W(t, p) = 0$ (resp. $W(p, t) = 0$) when $(t, p) \notin F$ (resp. $(p, t) \notin F$).

m are the number of transitions and places) given by entries

$$a_{ij} = W(t_i, p_j) - W(p_j, t_i) \quad (1)$$

that quantify the net change on the number of tokens at place p_j caused by firing the transition t_i. If we associate with transition $t = t_i$, the control (column) vector u of dimension $m \times 1$ given by $u(j) = 1$ if $j = i$ and $u(j) = 0$ otherwise, then the relation $M[t\rangle M'$ holds for two markings M and M' iff $M(p) \geq W(p,t)$ for each place $p \in P$, and $M' = M + A^T u$. In particular, the latter condition is *necessary* for $M[t\rangle M'$ to hold, and it can be generalized over arbitrary firing sequences $\sigma = u_1 u_2 \cdots u_k$. Indeed, if we let u_i to also denote the control vector of the transition that it represents, then the condition becomes $M[\sigma\rangle M'$ only if

$$M' = M + A^T \sum_{i=1}^{k} u_i \,. \quad (2)$$

Thus, a necessary condition for M to be reachable from the initial marking M_0 is that the linear system $A^T x = \Delta M$, for $\Delta M = M - M_0$, has solution x over the non-negative integers. Such a vector x is called a *firing-count vector* and tells how many times each transition needs to fire in order to reach M from M_0. For coverability, the condition is that $A^T x \geq \Delta M$ must have solution x over the non-negative integers. Intuitively, the Equation 2 is a balance equation for flows that go through the places of the net, saying that the net change between two markings, one accessible from the other, must be an exact match of the difference between what is produced and consumed by the sequence, at each place.

2.3 Petri Nets for SAS$^+$ Planning

A SAS$^+$ planning problem $P = \langle V, A, s_{init}, s_G, c\rangle$ is readily mapped into a P/T net $PN = \langle P, T, F, W, M_0\rangle$ where the places are all atoms of the form '$X = x$' for variable X and value $x \in D_X$, and the transitions correspond to the actions a in A. The flow relation F includes the pairs (p, a) for atoms $p =$ '$X = x$' and actions a such that $X \in Vars(\text{pre}(a))$ and $\text{pre}(a)[X] = x$, and pairs (a, p) such that $X \in Vars(\text{post}(a))$ and $\text{post}(a)[X] = x$. The function W assigns unit weight to each pair in F, while M_0 is the marking $M_{s_{init}}$ associated with the initial state s_{init}. In general, the marking M_s associated with a state s is the marking that puts one token at each place $p =$ '$X = x$' whenever $s[X] = x$, and puts zero tokens at the other places. The idea is to obtain a P/T net whose reachable markings M encode the reachable states s in the planning problem in such a way that $M = M_s$. However, this straightforward mapping does not always result in a faithful encoding of the planning problem because the resulting P/T net may not be 1-safe, meaning that a reachable marking may put 2 or more tokens at a place thus destroying the 1-1 correspondence between markings and states. In particular, there may be reachable markings M that do not stand for any reachable state either because M puts more than one token at a place, or because M puts a non-zero number of tokens at two places $p =$ '$X = x$' and $q =$ '$X = x'$' with $x \neq x'$. However, the following result always hold:

Theorem 1. *Let $P = \langle V, A, s_{init}, s_G, c\rangle$ be a SAS$^+$ planning problem and $PN = \langle P, T, F, W, M_0\rangle$ be the P/T net associated to it. An action sequence π is applicable at the initial state s_{init} only if π is a firing sequence for the initial marking M_0. Moreover, if π reaches the state s from s_{init}, then the marking M reached by π* covers *the marking M_s; that is, M is such that $M_0[\pi\rangle M$ and $M_s \leq M$.*

Proof. Both claims are proved by induction on the length π. Along the induction, one shows (simultaneously) 1) if π is applicable at s_{init}, π is a firing sequence for M_0, and 2) if $s = res(\pi, s_{init})$ and $M_0[\pi\rangle M$, then $M_s \leq M$. □

This theorem is important as it will allow us to prove the admissibility of the new heuristic based on the state equation.

3 The SEQ Heuristic

Let π be an action sequence that achieves the state s from s_{init} with s satisfying the goal (i.e., $s[X] = s_G[X]$ for each variable $X \in Vars(s_G)$). The firing-count vector u_π for π is the n-dimensional column vector u_π in which $u_\pi(i)$ is the number of times that the i-th action in P appears in the sequence π. By the state equation and Theorem 1, π is a firing sequence for M_0 and

$$A^T u_\pi \;=\; M - M_0 \;\geq\; M_s - M_0 \;\geq\; M_{s_G} - M_0 \,, \quad (3)$$

where M is the marking generated by firing π on M_0, and M_s and M_{s_G} are the markings associated with s and s_G respectively. On the other hand, the cost of π can be expressed as $c^T u_\pi = \sum_{i=1}^{n} c(i) u_\pi(i)$ where c is the vector of action costs such that $c(i)$ is the cost of the i-th action.

Therefore, if x is a non-negative integer solution of the linear system $A^T x \geq M_{s_G} - M_0$ with minimum $c^T x$ cost, then $c^T x \leq c^T u_\pi$ as u_π is a non-negative integer solution of the same system. On the other hand, if $A^T x \geq M_{s_G} - M_0$ has no solution, then there is no sequence π that achieves the goal s_G from the initial state s_{init}. These two properties are sufficient to define an admissible heuristic h^{iSEQ} for SAS$^+$ planning that is defined for state s as the minimum cost $c^T x$ of any non-negative integer solution x for $A^T x \geq M_{s_G} - M_s$. Clearly, this heuristic is not an abstraction heuristic as it is defined in terms of the whole planning problem. Also, it is not a delete-relaxation heuristic as it considers the positive and negative effects of the actions (or postconditions for actions in SAS$^+$), and indeed, it may even infer that an action must be applied multiple times in order to reach the goal from a given state s. However, the heuristic h^{iSEQ} is not computable in polynomial time as it involves the solution of an integer linear program. Yet, we overcome this limitation by approximating h^{iSEQ} with the solution of the relaxed LP problem in which the integrality constraints are dropped. Hence,

Definition and Theorem 2. *The h^{SEQ} heuristic for SAS$^+$ planning is the function that assigns the state s the value $\lceil c^T x^* \rceil$, where x^* is the solution of the LP problem*

$$\begin{aligned} &\textit{Minimize} && c^T x \\ &\textit{subject to} && A^T x \;\geq\; M_{s_G} - M_s \\ & && x \;\geq\; 0 \,, \end{aligned}$$

if the LP has a feasible solution, and ∞ if the LP is not feasible; the case of unbounded solutions is not possible. Then, h^{SEQ} is an admissible heuristic for SAS$^+$ planning.

We now discuss improvements to the base h^{SEQ} heuristic that can be implemented in a domain-independent manner.

3.1 Improvements

We report the coverage and performance for the base h^{SEQ} heuristic in the next section. In summary, the heuristic is competitive with the best heuristics across several domains, but shows degraded performance on other domains. In this section we propose three different methods to improve the quality of the heuristic. The methods consist in either reformulating the planning problem, analyzing its safeness properties, or using the information contained in the action landmarks.

Reformulations

Problems with few goals generate goal markings M_{s_G} that have few non-zero entries. These markings result on weak constraints for the LP that yield solutions x^* that may be ineffective to guide the search. One way to increase the value of the heuristic is to add atoms to the goal that must hold along the other conditions originally specified. For example, if the goal contains the atom $X = x$, which is not true initially, and every action that makes $X = x$ true also makes $Y = y$ true, and no other action destroys $Y = y$, then the planning problem can be modified without loosing any solution by extending the goal with the atom $Y = y$.

This idea is illustrated in the experiments where a slight change in the `airport` instances (done manually) increases the number of problems solved by 72.7% from 22 to 38.

Analysis of Safeness

As mentioned before, a P/T net is safe if no reachable marking assigns two or more tokens to a single place. In such a case, the reachable markings of the net are in 1-1 correspondence with the reachable states of the planning problem, and some of the inequalities in the LP problem can be tightened up to equalities. Indeed, for every place p in the P/T net for which $M_{s_G}(p) = 1$ (i.e. p refers to an atom of the form $X = x$ such that $s_G[X] = x$), we can safely enforce the constraint $A_p^T\, x = M_{s_G}(p) - M_s(p)$ in the LP, where A_p^T is the row of the matrix A^T for place p and s is the state on which the heuristic is being computed. The proof that this constraint can be added without loss of admissibility becomes obvious once it is observed that for safe nets, the first inequality in (3) is indeed an equality, by the correspondence between states and markings, and so for the second inequality for the coordinate referring to atom p since $M_{s_G}(p) = 1$ and all reachable markings are 1-bounded.

As shown by Hickmott *et al.* [2007], a planning problem P can be reformulated in a domain-independent manner into an equivalent problem P' whose P/T net is safe. This is a possible method to tighten up the LP that defines the SEQ heuristic, but the method is exponential in the maximum of $|Vars(\text{post}(a)) \setminus Vars(\text{pre}(a))|$ when a ranges over all actions in the planning problem.

However, it is possible to define a restricted notion of safeness and then tighten up the inequalities using this notion. We say that a subset S of places is safe or 1-bounded for a P/T net PN if no reachable marking M puts more than one token at any place in S; i.e., $M(p) \leq 1$ for each $p \in S$. By applying the above argument to the places in S, we can safely change by equalities all the inequalities for places p that are in a safe subset S and for which $M_{s_G}(p) = 1$. Indeed,

Theorem 3. *Let P and PN be a planning problem and its corresponding P/T net. Let S be a subset of places that is safe and let s be a state for the planning problem. Then, the following LP gives an admissible estimate for the state s:*

$$
\begin{array}{lrcll}
\textit{Minimize} & c^T x & & & \\
\textit{subject to} & A_p^T x & \geq & M_{s_G} - M_s & \textit{for } p \notin S \cap V_G \\
& A_p^T x & = & M_{s_G} - M_s & \textit{for } p \in S \cap V_G \\
& x & \geq & 0 &
\end{array}
$$

where $V_G = Vars(s_G)$ is the set of variables mentioned in the goal. That is, if the LP is feasible and x^* is a solution, $\lceil c^T x^* \rceil$ is an admissible estimate for the cost of any plan for s; if the LP is unfeasible, there is no plan for s and the estimate ∞ is admissible; lastly, the LP cannot be unbounded.

Checking or finding safe sets S in a general P/T net is not an easy task. However, our P/T nets comes from SAS$^+$ problems and thus it is natural to consider the set S_X of places for variable X, defined by $S_X = \{X = x : x \in D_X\}$, as good candidates for safe sets. Moreover, the test to determine whether S_X is safe or not is performed on the planning problem and *not* on the P/T net: S_X is safe if every action that affects X has a precondition on X; i.e., there is no action a such that $X \in Vars(\text{post}(a)) \setminus Vars(\text{pre}(a))$.

Landmarks

h^{SEQ} estimates the costs of the actions needed to cause a net change of $M_{s_G}(p) - M_s(p)$ between the goal and the state s, for each place p. When the goal contains an atom $X = x$ that is true at the state s, the right-hand side of the constraint for the place $p =$ '$X = x$' is zero and the constraint plays a minor role in the LP (unless the atom $X = x$ is destroyed by another action enforced by the constraints in the LP). A similar situation appears when $X = x$ is a *prevail condition* of an action a since then the transition for a consumes and produces p causing a zero net effect on it. However, if we infer that the goal $X = x$ needs to be destroyed and re-established by any plan for state s, or that a prevail condition of such an action must be necessarily established by some other action, then we can add an additional constraint of the form

$$x(t_1) + x(t_2) + \cdots + x(t_k) \;\geq\; 1\,, \tag{4}$$

where the transitions $\{t_1, t_2, \ldots, t_k\}$ correspond to all the actions that have $X = x$ as an effect. In either case, such a set of transitions correspond to an *action landmark* for the planning problem. In general, an action landmark for state s is a set L of actions such that every plan for s must execute at least one action in L. Landmarks are the basis of the current state-of-the-art planners, either satisficing or optimal; they provide crucial information for the computation of heuristics and the serialization of goals and subgoals [Richter and Westphal, 2010], and there are many ways to compute them [Hoffmann *et al.*, 2004; Zhu and Givan, 2003].

If we are given a collection of landmarks for state s, we can add one constraint for each landmark L in the collection. The

Domain	$h^{\text{LM-cut}}$	$h^{\text{LM-cut}}_{\text{ours}}$	h^{LA}	$h^{\text{M\&S}}$	HSP^*_F	h^{SEQ}	$h^{\text{SEQ}}_{\text{safe}}$
Airport (50)	**38**	35	24	16	15	22	23
Blocks (35)	28	28	20	18	**30**	28	28
Depot (22)	**7**	**7**	**7**	**7**	4	6	6
Driverlog (20)	**14**	**14**	**14**	12	9	11	11
Freecell (80)	15	15	**28**	15	20	**30**	**30**
Grid (5)	**2**	**2**	**2**	**2**	0	**2**	**2**
Gripper (20)	6	6	6	**7**	6	**7**	**7**
Logistics-2000 (28)	**20**	**20**	**20**	16	16	16	16
Logistics-1998 (35)	**6**	**6**	5	4	3	3	3
Miconic-STRIPS (150)	**140**	**140**	**140**	54	45	50	50
MPrime (35)	25	**24**	21	21	8	21	21
Mystery (19)	**17**	**17**	15	14	9	15	15
Openstacks-STRIPS (30)	**7**	**7**	**7**	**7**	**7**	**7**	**7**
Pathways (30)	**5**	**5**	4	3	4	4	4
Pipesworld-no-tankage (50)	17	17	17	**20**	13	15	15
Pipesworld-tankage (50)	11	11	9	**13**	7	9	9
PSR-small (50)	49	49	48	**50**	**50**	**50**	**50**
Rovers (40)	**7**	**7**	6	6	6	6	6
Satellite (36)	8	**9**	7	6	5	6	6
TPP (30)	6	6	6	6	5	**8**	**8**
Trucks (30)	**10**	9	7	6	9	**10**	**10**
Zenotravel (20)	**12**	**12**	9	11	8	9	9
Airport-modified (50)	na	36	na	na	na	**38**	**38**
Total (w/o Airport-modified)	**450**	446	422	314	279	335	336

Table 1: Coverage for different heuristics and h^{SEQ}. Data taken from references except for $h^{\text{LM-cut}}_{\text{ours}}$ and the SEQ heuristics that were run by us. The last column refers to h^{SEQ} improved with safeness information. Gray cells highlights difference for the same heuristic.

Domain	$h^{\text{LM-cut}}_{\text{ours}}$	h^{SEQ}	$h^{\text{SEQ}}_{\text{safe}}$
Elevators-08-STRIPS (30)	**19**	9	9
Openstacks-08-STRIPS (30)	**19**	16	16
Parcprinter-08-STRIPS (30)	22	**28**	**28**
Pegsol-08-STRIPS (30)	**27**	26	**27**
Scanalyzer-08-STRIPS (30)	**15**	12	12
Sokoban-08-STRIPS (30)	**28**	17	17
Transport-08-STRIPS (30)	**11**	9	9
Woodworking-08-STRIPS (30)	**15**	12	12
Total	**156**	129	130

Table 2: Coverage for $h^{\text{LM-cut}}$ and h^{SEQ} over the problems of IPC-08 that involve actions of different costs. Gray cells highlights difference for the same heuristic.

resulting LP (and also ILP) is guaranteed to deliver admissible estimates for the cost of an optimal plan for s. Recently, landmarks had been used to strengthen abstraction heuristics in a similar way [Domshlak *et al.*, 2012b]. In the experiments below, we did not test the use of landmarks to improve the heuristic.

4 Experiments

We implemented h^{SEQ} within the Fast Downward system, that also implements the LM-cut heuristics, and performed experiments on Xeon 5140 'Woodcrest' CPUs running at 2.33GHz and with 2GB of memory, with a cutoff time of 30 minutes. The results for other heuristics were obtained under similar circumstances and are taken from [Helmert and Domshlak, 2009]. For solving LPs, we tried three publicly available solvers: LPSOLVE, the GNU LP Kit library, and CLP. All libraries produced similar results, yet there are some discrepancies on the efficiency of the solvers, with no one dominating the others across all the problems in the benchmark. The results reported here were obtained with the CLP library.

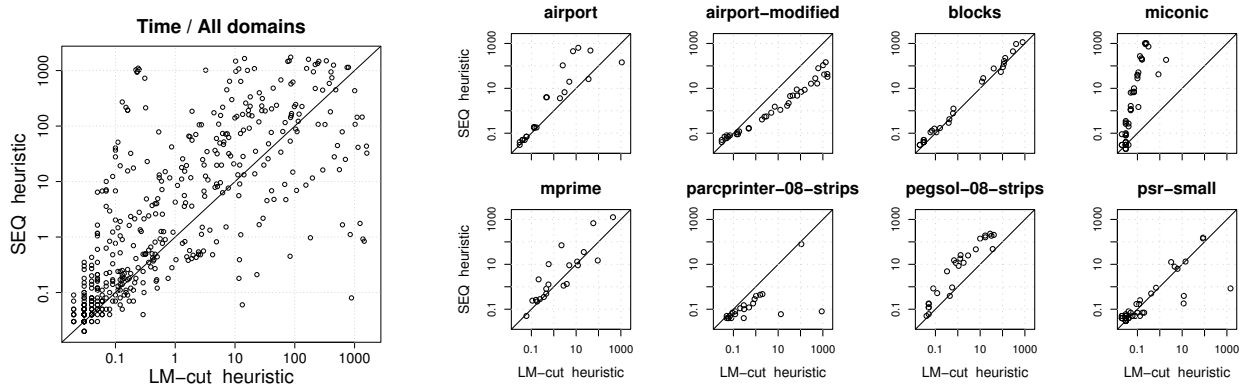
Table 1 shows coverage results per domain for state-of-the-art heuristics for optimal planning. The data for the two h^{SEQ} columns and for the $h^{\text{LM-cut}}_{\text{ours}}$ column was obtained by us. The last column refers to the heuristic h^{SEQ} improved with the safeness information that is automatically extracted from the planning problem as described above. Table 2 shows similar coverage results for the IPC-08 domains where action costs are not uniform. The SEQ heuristic is not dominated by any other heuristic, in terms of coverage for neither set of problems, and is competitive with the best heuristics (and more, if the 150 instances of Miconic are carefully re-considered). On the other hand, h^{SEQ} is able to solve two instances of `Freecell` and two of `TPP` that had no been reported before, and in the IPC-08 benchmarks, it performs quite well on `parcprinter`. The overall impact of the improvement provided by the safeness information is also shown in the tables; the difference is little because the improved LP very often provides the same estimates. To test the reformulation method for improving the heuristic, we *manually changed* the `airport` domain by simply extending the goal description with the fluent `(at-segment ?a ?s)` for each occurrence of the fluent `(is-parked ?a ?s)` in the goal, and with the fluent `(not_blocked ?s ?a)` for each occurrence of the fluent `(airborne ?a ?s)` in the goal.

In terms of running times, the panel on the left in Fig. 1 shows a comparison between the SEQ and LM-cut heuristics on the commonly solved instances across all the domains, while the panel on the right shows results within selected domains. As seen, there is no a clear dominance between heuristics on these instances, and interestingly, there seem to be some complementary behaviour for the heuristics, on a domain basis, which may be exploited by using portfolio systems, or the selective-max heuristic [Domshlak *et al.*, 2012a].

The final chart, in Fig. 2, compares the number of expanded nodes by both heuristics on the commonly solved instances. The total number of expansions favors LM-cut with respect to SEQ, yet there several instances in which SEQ expands far less nodes. If we compute the average number of expansions per second, by dividing the total number of expanded nodes by the total time to solve the problems, we find that LM-cut expands on average 713.02 nodes per second, while SEQ expands 2,211.02 nodes per second. These numbers quantify the quality-to-cost ratio of the heuristic since a better heuristic means less expanded nodes while a more efficient one means less total accumulated time.

5 Discussion

A few years ago a similar LP-based heuristic for optimal SAS^+ planning was put forward by van den Briel *et al.* [2007]. Their LP formulation is obtained directly from the planning problem and is also based on "flow" relations between fluents and actions as the SEQ heuristic. However, such formulation uses variables for actions and fluents (not only actions like SEQ) while the constraints are obtained from the domain transition and causal graphs of the problem. Both formulations are different, in terms of variables and constraints, while the van den Briel et al.'s seems to produce tighter bounds on some problems because the prevail



a) across all domains

b) over domains with at least 20 commonly solved instances

Figure 1: Running times for the LM-cut and SEQ heuristics. Panel (a) compares the heuristics over commonly solved instances across all the domains. Panel (b) is a comparison within a domain, for domains with at least 20 commonly solved instances. Points above the diagonal represent instances on which LM-cut is better, while points below the diagonal represent instances on which SEQ is better. Plots are in logscale and times are in seconds.

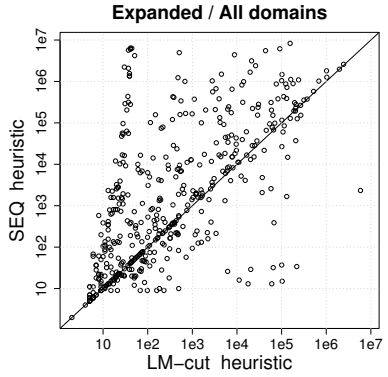


Figure 2: Comparisons of the number of expanded nodes for the SEQ and LM-cut heuristics for instances solved by both heuristics and across all domains. Points above the diagonal represent instances on which LM-cut is better, while points below it instances on which SEQ is better. Plots are in logscale.

conditions play a more active role. On the other hand, the resulting LPs are more difficult to solve to the point of being ineffective for guiding a search-based planner. This is clearly seen in their experiments in which only the value for initial states are computed.

h^{SEQ} approximates the ILP that defines h^{iSEQ}. In general, the LP for h^{SEQ} does not have integral solutions, but we have observed that it does so in many problems. A known fact between ILPs and their LP relaxations is that if the matrix A that define the constraints is *totally unimodular* (TU), then the relaxed LP has integral solutions for any cost vector and thus solve the ILP [Korte and Vygen, 2001]. A matrix A is TU when every minor (determinant of square submatrix) has value in $\{-1, 0, 1\}$. Furthermore, it is known that the incidence matrix of any directed graph is TU. The incidence matrix of a P/T net is not the incidence matrix of the net when viewed as a directed graph,[3] yet it is similar and related to it. One can define another heuristic in terms of the ILP defined by the TU matrix of the P/T net which can then be solved exactly in polytime by just dropping the integrality constraints. However, the new heuristic is not as informative as h^{SEQ}. We do not have yet a clear interpretation for the dual of the LP for h^{SEQ}. Certainly it is an important issue that may shed light on the strengths and weaknesses of h^{SEQ} and ways to improve it. We plan to study it in the near future.

We finish this discussion with a brief summary. The paper introduces a new admissible heuristic for optimal SAS$^+$ planning that is obtained from the state equation for the Petri-net associated with the problem. The heuristic is defined in terms of an LP problem that need to be solved for each node encountered during the search. Despite the overhead of this operation, the resulting heuristic is competitive with state-of-the-art heuristics in both number of problems solved and time to solve them.

Acknowledgements

The experiments were performed at the ORION cluster at the Universidad Pompeu Fabra, Barcelona, Spain.

References

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

[Bonet and Helmert, 2010] B. Bonet and M. Helmert. Strengthening landmark heuristics via hitting sets. In *Proc. ECAI*, pages 329–334, 2010.

[3] Indeed, the incidence matrix for a directed graph has rows associated with vertices and columns with edges so that the entry (i, j) is 1, -1, or 0 if the j-th edge enters, leaves, or is not incident at the i-th vertex. Thus, every column has exactly two non-zero entries, one being 1 and the other -1. This property does not hold for the incidence matrix of a P/T net.

[Coles *et al.*, 2008] A. Coles, M. Fox, D. Long, and A. Smith. Additive-disjunctive heuristics for optimal planning. In *Proc. ICAPS*, pages 44–51, 2008.

[Domshlak *et al.*, 2012a] C. Domshlak, E. Karpas, and S. Markovitch. Online speedup learning for optimal planning. *JAIR*, 44:709–755, 2012.

[Domshlak *et al.*, 2012b] C. Domshlak, M. Katz, and S. Lefler. Landmark-enhanced abstration heuristics. *AIJ*, 189:48–68, 2012.

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

[Haslum and Geffner, 2000] P. Haslum and H. Geffner. Admissible heuristic for optimal planning. In *Proc. AIPS*, pages 140–149, 2000.

[Haslum *et al.*, 2007] P. Haslum, A. Botea, M. Helmert, B. Bonet, and S. Koenig. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *Proc. AAAI*, pages 1007–1012, 2007.

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

[Helmert *et al.*, 2007] M. Helmert, P. Haslum, and J. Hoffmann. Flexible abstraction heuristics for optimal sequential planning. In *Proc. ICAPS*, pages 176–183, 2007.

[Hickmott and Sardiña, 2009] S. L. Hickmott and S. Sardiña. Optimality properties of planning via Petri net unfolding: A formal analysis. In *Proc. ICAPS*, pages 170–177, 2009.

[Hickmott *et al.*, 2007] S. Hickmott, J. Rintanen, S. Thiébaux, and L. White. Planning via Petri net unfolding. In *Proc. IJCAI*, pages 1904–1911, 2007.

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

[Hoffmann *et al.*, 2004] J. Hoffmann, J. Porteous, and L. Sebastia. Ordered landmarks in planning. *JAIR*, 22:215–278, 2004.

[Karpas and Domshlak, 2009] E. Karpas and C. Domshlak. Cost-optimal planning with landmarks. In *Proc. IJCAI*, pages 1728–1733, 2009.

[Katz and Domshlak, 2008] M. Katz and C. Domshlak. Structural patterns heuristics via fork decomposition. In *Proc. ICAPS*, pages 182–189, 2008.

[Korte and Vygen, 2001] B. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms (2nd Edition)*. Springer, Berlin, Germany, 2001.

[Murata, 1989] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[Richter and Westphal, 2010] S. Richter and M. Westphal. The LAMA planner: Guiding cost-based anytime planning with landmarks. *JAIR*, 39(1):127–177, 2010.

[van den Briel *et al.*, 2007] M. van den Briel, J. Benton, S. Kambhampati, and T. Vossen. An LP-based heuristic for optimal planning. In *Proc. CP*, pages 651–665, 2007.

[Zhu and Givan, 2003] L. Zhu and R. Givan. Landmark extraction via planning graph propagation. In *ICAPS Doctoral Consortium*, pages 156–160, 2003.

Pruning Methods For Optimal Delete-free Planning Using Domination-Free Reachability

Avitan Gefen and **Ronen I. Brafman**
Department of Computer Science
Ben-Gurion University of The Negev, Israel

Abstract

Delete-free planning (DFPlan) underlies many popular relaxation (h^+) based heuristics used in state-of-the-art planners, and a number of recent planning domains are naturally delete-free. This has led to increased interest in efficient methods for cost-optimal DFPlan. To aid in the solution of DFPlan problems, we introduce a new analysis technique, called domination-free reachability (DFR). DFR is an improved version of the well known notion of reachability in graphs that filters out nodes that are not useful for optimal planning. We explain how to compute DFR, and present three new pruning techniques that use it. Combined with a recent decomposition technique these pruning methods lead to effective pruning in delete-free planning.

Introduction

Heuristic search is currently the preferred method for solving satisficing and optimal planning problems. Among alternative methods for generating heuristic functions, relaxation-based heuristic functions are extremely popular, effective, and influential (Hoffmann and Nebel 2001; Helmert and Domshlak 2009). These methods estimate minimal distance-to-goal of a state in the delete-free problem generated from a given instance by removing all delete-effects. Computing this value, known as h^+, is NP-hard (Bylander 1994). Effective approximations exist (Hoffmann and Nebel 2001; Helmert and Domshlak 2009), yet efforts continue to improve these techniques in order to provide even better heuristic estimates (Pommerening and Helmert 2012; Haslum, Slaney, and Thiébaux 2012; Gefen and Brafman 2012), as well as to handle naturally delete-free domains more effectively (Gefen and Brafman 2011; Porco, Machado, and Bonet 2011; Gallo et al. 1993).

In this paper we investigate improved pruning techniques for delete-free planning (DFPlan). Our pruning techniques rely on a new construction, *domination-free reachability* (DFR) that seeks to improve the standard notion of reachability for optimal planning. Reachability and reachability analysis, whether forwards or backwards, underlies some fundamental techniques in planning. DFR attempts to provide a more refined notion of reachability, one that filters out "useless" nodes in the context of optimal planning. A node is useless in optimal planning if it is not part of some optimal plan from the initial state to the goal. Obviously, computing whether a node is part of an optimal plan is a difficult problem, and DFR which is a practical, polynomial time technique cannot compute this. What DFR provides is an approximation of this concept, whose practical utility is shown in our empirical evaluation.

The DFR technique operates on a faithful graphical depiction of a DFPlan problem, called the relaxed causal graph (Keyder, Richter, and Helmert 2010; Gefen and Brafman 2012). The nodes of this graph correspond to facts and actions. Given special source node (that corresponds to the initial state) and a target node (goal state) DFR seeks to find all nodes that are part of at least one minimal plan from the initial state to the goal. A plan π for G is *minimal* if no other plan for G is a strict subset of the set of actions in π. That is, we can not remove any action from π without loosing its legality or its ability to achieve G. Finding the set of all nodes which are part of a minimal plan is NP-hard, and the DFR is an over-approximation of it that is often much smaller than the set obtained using standard reachability.

We focus on the notion of minimal plans because of the results of (Gefen and Brafman 2012) (GB). GB show that, given a set of landmarks suitably ordered, an optimal plan can be generated by generating minimal plans between these landmarks. This implies that one can decompose the planning problem into multiple simpler search problems in each of which we seek to achieve a landmark, and where one *needs only consider minimal plans.* The DFR helps us make the search for each such minimal plan more efficient because it allows us to prune nodes that are guaranteed to not belong to any minimal plan.

In the first part of this paper, we define the notion of DFR, show how to compute it, and demonstrate its soundness with respect to minimal plans. In the second part of the paper, we show how to use the DFR to prune A* search for an optimal plan. We discuss three methods. The first, the simplest, the most obvious, and the most effective simply prunes actions that are not part of the (backwards) DFR of the goal. The second technique is a simpler variant of the notion of disjoint-path commitment introduced by GB. Intuitively, if a_1 and a_2 are two actions that achieve the goal then a minimal plan will include only one of them. If we can identify that our current plan is a prefix of a minimal plan targeting

a_1, due to minimality, we can commit to a_1, and prune actions that are only part of a minimal plan targeting a_2. The DFR algorithm, which provides a sound estimate of "being in a minimal plan" allows us to operationalize this intuition. Finally, in the last pruning method we describe, we use the DFR algorithm to provide an upper bound on the cost of a minimal plan because we know that the true minimal plan is a subset of the set computed by DFR. This bound can be used for standard bound-based pruning. We conclude the paper with an empirical evaluation of these pruning methods and a short discussion.

Delete-Free Planning Tasks

A *delete-free STRIPS planning task*, or *DF task* for short, is a 4-tuple (P, A, I, G). P is a finite set of *propositions*. A state s is represented by the set of propositions that are *true* in it. $I \subseteq P$ is the *initial state*. $G \subseteq P$ is the set of propositions that must be true at any *goal state*. A is the set of *actions*, where $a \in A$ has the form: $a = \langle pre(a), add(a) \rangle$ denoting its preconditions and add effects. An action a is applicable in a state $s \subseteq P$ iff $pre(a) \subseteq s$. Applying a in s transforms the system to the state $s \cup add(a)$. We use $a(s)$ to denote the resulting state. When a is not applicable in s then $a(s)$ is undefined. We do *not* consider actions with conditional effects. Therefore, there is always an optimal plan with only one instance of each action, as a second instance of an action has no effect on the state.

A *solution* to a DF task is a plan $\pi = (a_1, \ldots, a_k)$ such that $G \subseteq a_k(\cdots(a_1(I))\cdots)$. That is, it is a sequence of actions that transforms the initial state into a state satisfying the goal conditions.

Landmarks play an important role in the work of GB: they are used to decompose the problem. A *fact landmark* for a state s is a proposition that holds at some point in every plan from state s to the goal (Hoffmann, Porteous, and Sebastia 2004). An *action landmark* for a state s is an action that must be part of any plan from s to the goal. A *disjunctive action landmark* for a state s is a set of actions, at least one of which must be part of every plan from state s to the goal (Helmert and Domshlak 2009).

A plan π for G is *minimal* if no other plan for G contains a strict subset of the actions in π. Clearly, any optimal plan must be minimal, unless zero-cost actions exist, in which case it must have a sub-plan which is minimal. Therefore, when seeking an optimal plan for G, we can prune any plan that is not minimal without sacrificing optimality.

The following Lemma underlies the decomposition technique of GB.

Lemma 1 (Gefen & Brafman, 2012)**.** *Let L be a set of fact landmarks for a DFPlan problem $\Pi = (P, A, I, G)$, such that $G \subseteq L$. Then, if there is a solution to Π, there exists an ordering $l_1, \ldots, l_k$ of L and a minimal plan π for Π such that $\pi = \pi_1, \ldots, \pi_k$, where π_i is a minimal plan for $(P, A, \pi_{i-1}(\cdots(\pi_1(I))\cdots), l_i)$.*

Building on existing efficient algorithms for landmark detection (Keyder, Richter, and Helmert 2010), GB provide an efficient procedure for generating a landmark ordering as needed in Lemma 1. Thus, instead of planning for G, one can incrementally plan for each of the landmarks, keeping around minimal plans only. Although this method may prune some optimal plans, it is guaranteed to leave some optimal plans intact.

For the first part of the paper, we will focus on the task of finding one of the component minimal plans π_i. This could be a plan to reach l_1 from I, or a plan to reach l_k from $\pi_{k-1}(\cdots(\pi_1(I))\cdots)$. The reader can simply think of it as reaching G' from some initial state I'.

The Relaxed Causal And/Or graph

We can capture the structure of delete-free problems using a directed And/Or graph known as the *relaxed causal graph* (RCG) (Keyder, Richter, and Helmert 2010; Gefen and Brafman 2012). An And/Or graph is a directed graph with two types of nodes: *And* nodes and *Or* nodes. The RCG $\mathcal{G}$ associates an *And* node with each action and an *Or* node with each fact. There is an edge from (the node for) fact p to action a if $p \in pre(a)$, and an edge from a to p if $p \in add(a)$. In addition, there are two special *Or* nodes and two special *And* nodes: s the start node (Or), is attached to a special *And* node i via a special initial state edge, (s, i), and t (Or) is attached to a special *And* node g via a special goal edge, (g, t). There is an edge from s to every action with no preconditions. There is an edge from s to i and an edge from i to every initial state fact. There is an edge from every goal fact to g, and an edge from g to t. See Figure 1 for an example. To keep the generality of planning terms for the RCG we define the following: $pre(i) = s$, $pre(a) = s$ for every action a without a precondition, $add(i) = \{\text{initial state facts}\}$, $pre(g) = \{\text{goal facts}\}$, $add(g) = t$.

A subgraph $J = \langle V^J, E^J \rangle$ of $\mathcal{G}$ is a DF plan (aka, *justification subgraph*) iff the following holds:

1. It contains s and t.
2. For every *And* node $a \in V^J$, it contains all incoming edges and the *Or* nodes from which they originate.
3. For every *Or* node $p \in V^J \setminus \{s\}$, it contains at least one incoming edge and the node it originates from.
4. J is acyclic.

These conditions ensure that there is an action achieving each proposition (as well as t) and that each action has all its preconditions satisfied. The acyclicity ensures the plan is well founded.

Domination-Free Reachability

Forwards and backwards reachability analysis is one of the central methods used in planning. In particular, structured reachability analysis underlies popular techniques such as planning graphs (Blum and Furst 1997).

In this section, we define the notion of domination-free reachability (DFR) in an RCG. The DFR set for a node v, $dfr(v)$. Ideally, we would have liked $dfr(v)$ to contain only fact nodes n that are part of a minimal plan from s to v. Unfortunately, this would be too hard to compute:

Lemma 2. *Finding the set of all nodes which are part of a minimal plan is NP-hard.*

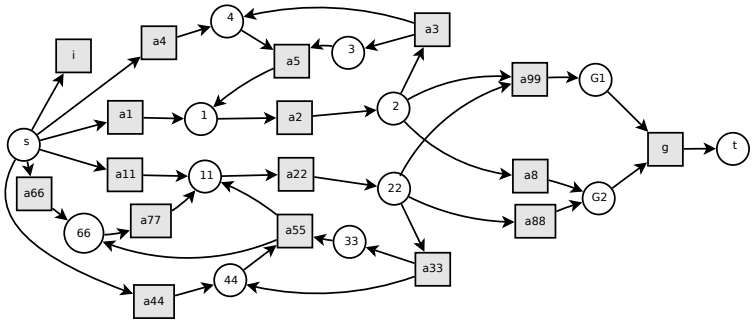


Figure 1: An RCG. Facts in white, actions in grey, G1 and G2 are the goal and hence fact landmarks, g is a special goal action. $Dom(2) = \{s, 1, 2\}$, $Dom(3) = \{s, 1, 2, 3\}$, $Dom(22) = \{s, 11, 22\}$, $Dom(33) = \{s, 11, 22, 33\}$. $dfr(s) = \{s\}$, $dfr(1) = \{s, 1\}$, $dfr(2) = \{s, 1, 2\}$, $dfr(3) = \{s, 1, 2, 3\}$, $dfr(4) = \{s, 1, 2, 3, 4\}$, $dfr(11) = \{s, 11, 44, 66\}$, $dfr(22) = \{s, 11, 22, 44, 66\}$, $dfr(33), dfr(44), dfr(66) = \{s, 11, 22, 33, 44, 66\}$

Proof. The hardness of this problem comes from the fact that a minimal plan is a generalization of the concept of a simple path in control flow graphs. (imagine a planning problem that can be represented using only a directed graph). The question of whether there exist a simple path from s to t with x as an obligatory node is equivalent to the 2-disjoint paths problem (Perl and Shiloach 1978) which is known to be NP-complete. □

Instead, $dfr(v)$ conservatively approximates (= is a superset of) the set of fact nodes that can be part of a minimal plan from the start node s to v. That is, all nodes in $dfr(v)$ lie on some path from s to v, but not all fact nodes on such paths necessarily belong to $dfr(v)$. That is, some nodes that would be obtained by standard reachability analysis for v are not part of $dfr(v)$. Hence, domination-free reachability is a *filtered* form of reachability that can prune certain nodes that are not part of a minimal plan.

To the best of our knowledge, it is the first effective variant of reachability geared towards optimal planning. We will define the notion of DFR, an algorithm to compute it, and demonstrate it through an example. We will show that it is an over-approximation, i.e., that it can contain nodes that are not part of a minimal plan, but that it can be much smaller than the standard set of reachable nodes.

Domination

Our procedure is based on the notion of *domination* used in *control flow* graphs: "Vertex v dominates vertex w in flow graph $(G; s)$ if $v \neq w$ and every path from s (start node) to w contains v" (Tarjan 1974). We adapt the idea of domination to And/Or graphs as follows: a fact v dominates a fact w in the RCG $\mathcal{G}$ if every DF plan from s to w contains v. Intuitively, one can understand this as saying that every plan from the current state that achieves w achieves v, as well. Slightly departing from the definition in control-flow graphs, our definition also implies that v dominates itself.

Landmarks are often used in planning to refer to goal dominators, but also, sometimes, to dominators in general. We will use the original "dominators" term because it comes with less baggage, it is more appropriate linguistically for our setting, and it is related to the graph-theoretic ideas that motivate us. We use $Dom(x)$ (called the *Dom set* of x) to denote all fact nodes that dominate fact x in the RCG, including x itself. This set could also be referred to as landmarks for x. For a set X, we define $Dom(X) = \bigcup_{x \in X} Dom(x)$. The process used to find landmarks in (Keyder, Richter, and Helmert 2010) computes the Dom sets naturally.

We use the notion of domination to refine the notion of reachability. To do so, we must formalize the notion of a path in the RCG. We borrow the following paths definitions from directed-hypergraphs (Gallo et al. 1993): A path $p = (v^0, a^1, v^1, a^2, v^2, \ldots, a^k, v^k)$ in the RCG is a sequence of facts and actions, where v^0 is the origin node, v^k is a target node and for every $1 \leq i \leq k$, $v^{i-1} \in pre(a^i)$ and $v^i \in add(a^i)$. A *simple path* is a path in which no action appears twice. A simple path will be *elementary* if no fact appears twice.

To better illustrate the advantage of DFR we define $reach(v)$, which captures the standard notion of reachability. $reach(v)$ is the set of action nodes that are part of some RCG path from the start node s to v. This set can be obtained by a backwards traversal of the RCG (treated as a regular directed graph) that starts at v and collects nodes along this traversal. Alternatively, one can do a forward traversal starting at s. These procedures can collect action nodes that are not part of a minimal plan to v.

For example, in Figure 1 $reach(1) = reach(2) = reach(3) = reach(4) = \{a_1, a_2, a_3, a_4, a_5\}$. In contrast to these sets, we can see that the only minimal plan from s to 2 will contain action nodes a_1, a_2 and fact nodes $s, 1, 2$. Indeed, node 3 is dominated by node 2.

If we can identify this fact, we can ignore node 3 when exploring the space of possible minimal plans from s to 2. This will be done (to some extent) by the DFR algorithm below.

DFR Algoritm

Algorithm 1, described above takes the RCG graph for the initial state ($\mathcal{G}(I)$) at its input, and its output are the DFR sets for each fact node. It iteratively propagates information forward, and can be understood as reachability analysis with filtration.

Algorithm 1 Domination-Free-Reachability

1: **DFR**($\mathcal{G}(I)$) {$\mathcal{G}(I)$ is initial state RCG}
2: **for each** $v \in V$ **do** $dfr(v) \leftarrow \{v\}$
3: $Q \leftarrow \{s\}$, $R \leftarrow \{s\}$ {s is the start node}
4: **while** $Q \neq \emptyset$ **do**
5: Select and remove $v \in Q$
6: **for all** $a \in consumers(v)$ **do**
7: **if** $pre(a) \subseteq R$ **then**
8: **for all** $ef \in add(a)$ **do**
9: $R \leftarrow R \cup \{ef\}$
10: **if** $ef \notin Dom(pre(a))$ **then**
11: **if** $add(a) \not\subseteq dfr(ef)$ **do** $Q \leftarrow Q \cup \{ef\}$
12: $dfr(ef) \leftarrow dfr(ef) \cup add(a)$
13: **for all** $pc \in pre(a)$ **do**
14: **for all** $r \in dfr(pc)$ **do**
15: **if** $ef \notin Dom(r)$ **and** $r \notin dfr(ef)$ **then**
16: $dfr(ef) \leftarrow dfr(ef) \cup \{r\}$
17: $Q \leftarrow Q \cup \{ef\}$
18: **end if**
19: **end for**
20: **end for**
21: **end if**
22: **end for**
23: **end if**
24: **end for**
25: **end while**

Let $dfrA(v) = \{a |\ pre(a) \cup add(a) \subseteq dfr(v)\}$ be the set of actions which their related fact nodes are in the $dfr(v)$. Then, for each $v \in V$, $dfrA(v) \subseteq reach(v)$. For example, in Figure 1 $dfr(1) = \{s, 1\}$ and $dfr(2) = \{s, 1, 2\}$, showing the superiority of the DFR sets over regular reachability. Unfortunately, a formal characterization of $dfr(v)$ is non-trivial – we discuss it in a longer version of this paper which is in preparation. Roughly speaking, $dfr(v)$ holds all facts which are part of a minimal plan to v. These are nodes that can reach v (or can be reached while achieving v) but which satisfy the additional constraint that they are not dominated by v. There are two exceptions: (1) v itself is dominated by v, and yet it is part of $dfr(v)$. (2) Nodes that appear together with v in the add effect of some action.

Technically, the algorithm works as follows: For every *Or* node v, we initialize $dfr(v)$ to v, since any node reaches itself as long as there is a plan that achieves it. Two other sets are Q – our propagation set, and R – the reachability set. Both are initialized with the initial state node s. Now, we start the iterative process (l. 4): To propagate information forward, we remove a fact node v from Q and go over all actions that can consume v (v is one of their precondition) (l. 6). For each such action a, if all of a's preconditions were already reached (l. 7), we can try to use a to propagate information to its effect nodes. Therefore, we iterate over a's effect nodes ef (l. 8) as follows: First, we update the reachability set R with ef – the node we just reached (l. 9). Now, our first filtration is based on the fact that information should be propagated to ef only if ef does not dominate a (i.e., any of its preconditions). Otherwise, any plan that achieves ef using a (ef is already true before applying a) is not minimal (l. 10). If ef does not dominate a, we would like to add $add(a)$ to $dfr(ef)$, since they can be achieved simultaneously with ef (l. 11,12). Finally, we can go over all DFR sets of the preconditions pc of a. For each node $r \in dfr(pc)$, as long as ef does not dominate r, we can propagate r to $dfr(ef)$ (l. 13–16). When we propagate new information we update Q so the new information can be propagated further (l. 11 & 17).

Let us examine an example using Figure 1. The RCG in Figure 1 holds two similar sub-graphs: one that is composed of nodes 1, 2, 3, 4 and the other 11, 22, 33, 44, 66. The fact landmarks are nodes 1, 2, 11, 22, G1, G2. A backward procedure that collect actions starting at fact node 1 will collect actions a_1, a_2, a_3, a_4, a_5. Two of these actions are applicable in the initial state (a_1, a_4). On the other hand $dfr(1) = \{s, 1\}$, and therefore only action a_1 is in $dfrA(1)$ (since node 1 is a fact landmark and a_1 is the only action that can achieve it in a minimal plan, we can conclude it is an action landmark). This outcome results from the fact that node 1 is a dominator of 3, and therefore a_5 cannot propagate information into node 1.

Unfortunately, the DFR algorithm is not perfect as can be seen using node 11. Action a_{55} cannot propagate information to node 11, but it will propagate information to node 66. Action a_{77} will propagate nodes 44,66 to node 11 and so $dfr(11) = \{s, 11, 44, 66\}$. With node 44 in $dfr(11)$ also actions a_{44}, a_{55} will be part of $dfr(11)$ although there is no minimal plan to 11 with those actions.

Correctness

The following Lemma ensures the correctness of DFR sets as a super-set of minimal plans for some fact node.

Lemma 3. *Let $dfr(G')$ be the DFR set computed using Algorithm 1 for DFPlan problem $\Pi = (P, A, I', G')$. If for some action a, $pre(a) \cup add(a) \not\subseteq dfr(G')$ then there is no minimal plan for G' that contains action a.*

Proof. Paths in the RCG are the channels for information propagation in Reachability/DFR algorithms. Every plan (justification graph) can be seen as a union (of actions) of, not necessarily disjoint, simple paths. The number of actions in each simple path that is contained in some plan π must be $\leq$ the length of the plan. Therefore, the length of simple paths can serve as a lower bound to the length of a minimal plan.

Notice, that if π is a minimal plan, then for any simple path p "extracted" from π that preserves the order of π, we have that for any actions $a_i, a_j \in p$ where a_j appear after a_i, a_j cannot dominate a_i. This follows from the definition of a minimal plan. We now prove by induction on the length of simple paths that the DFR sets include all fact nodes which are part of a minimal plan.

(i) Our induction assumption is that when we covered all paths of length i, if there is a minimal plan with length $\leq i$ to x that includes a, then $pre(a) \cup add(a) \subseteq dfr(x)$.

(ii) The base case, $i = 0$, is immediate since $dfr(s)$ includes s and only s.

(iii) Assuming the induction hypothesis holds for length i we show it holds at length $i + 1$. Let us look at some fact node x. Let us assume there is a minimal plan π_x of length $i + 1$

to x that ends with a_x. A minimal plan to x must end with an action where $x \in add(a_x)$, moreover it can't achieve x before. If we remove a_x from π_x we may get a plan which is not minimal but is composed of paths with length $\leq i$.
We shall now look at two scenarios, one in which some action e is not an achiever of x (x is not an effect of e), and the second one where it is:
(1) Let us look at some action e which is not an achiever of x (ef in line 8). If $pre(e), add(e) \in dfr(pc)$ where pc is a precondition of a_x (an achiever of x) and x is not a dominator of $pre(a_x)$ (line 10) then we would try to propagate $pre(e), add(e)$ ($r \in dfr(pc)$) to x (line 13–16). If x dominate $y \in pre(e)$ (line 15) then each plan that uses e must reach x before applying e and therefore it is not minimal. x can dominate $y \in add(e)$ without dominate the preconditions of e if it appears with y in all add effects of actions that achieve y (Lemma 4), but this contradicts the fact that e is not an achiever of x.
(2) If $e = a_x$ is an achiever of x and x does not dominate $pre(e)$ (ef in line 10), then in the DFR algorithm $add(e)$ will be added to $dfr(x)$ (line 12), and since each precondition of e will propagate itself to x, since they are not dominated by x (line 13–16) we get that $pre(e), add(e) \in dfr(x)$.

At the end of the run of algorithm DFR we will propagate information for all possible minimal plans (and possibly more). □

Lemma 4. *Let v, x be some facts in a DFPlan problem. Let $A_x \subseteq A$ be a set of actions that achieve fact x (i.e. if $a_x \in A_x$ then $x \in add(a_x)$). If v dominates x, then: (i) It also dominates some precondition of each $a_x \in A_x$, or (ii) In every minimal plan to x where v does not dominate some precondition of $a_x \in A_x$ (the action that achieves x in that plan) it appears with x in the add effect i.e. $x, v \in a_x$.*

Proof. If v dominates $x \in add(a_x)$, every plan that achieves x must also achieve v. If v must be achieved before x, i.e. before some action a_x that achieves x is being applied then v must dominate some precondition of a_x. If there are plans where v is not achieved before x, then it must be achieved simultaneously with x, i.e. $x, v \in add(a_x)$. □

Pruning Using DFR

What follows are three pruning methods that strongly utilize the DFR algorithm. These methods can be applied to any DFPlan problem, but they are even more effective when combined with the decomposition method of GB. The reason for this is that GB's decomposition yields multiple planning problems, each with a shorter solution, and our pruning methods are more effective the closer the goal is to the initial state.

Basic DFR Pruning

The simplest and empirically most powerful pruning method is based on Lemma 3. Given a planning problem with initial state I' and goal state G', prune any action a such that $pre(a) \cup add(a) \not\subseteq dfr(G')$. Lemma 3 guarantees that such actions are not part of a minimal plan to G'.

Path-Commitment using DFR

One of the two pruning methods introduced by GB is called disjoint-path commitment. There, the last action a_{last} applied during A* search towards some goal l, is used to prune actions that are not part of a minimal plan containing a_{last}. To use this method, one must be able to recognize that certain actions belong to different, disjoint minimal plans that lead to l. GB use the following sound approximation to apply this idea in practice: (i) At preprocess time, define a set of labels – one for each action achieving l. Using a backwards traversal from l, propagate these labels backwards. Intuitively, one now identifies different path to the goal with their last action. Let $lbl(a)$ be the set of labels reached to action a. Intuitively, these labels mark the different path to l to which a belongs. (ii) Then, during search, as long as a_{last} did not achieve a fact landmark, one can safely prune any action a where $lbl(a) \cap lbl(a_{last}) = \emptyset$ – that is, the labels of that action are disjoint from the labels of the last action applied.

We will show how to use the DFR sets to obtain an improved version of GB's disjoint path commitment. It replaces GB backwards label propagation – essentially backwards reachability – with the use of DFR sets. As these sets provide a better form of reachability for minimal plans, the resulting method is more powerful. The basic idea is simple: First, associate every action in $dfr(G')$ with one or more preconditions of the actions that achieve the current goal, or landmark. We do so by "reverse" computing DFR sets of these actions.

Next, imagine we have only two actions a_p, a_q with preconditions p, q respectively, which achieve G'. If the action a_{last} that was last applied belongs to the DFR set of precondition p, but not of q, then any minimal plan containing a_{last} must achieve the goal via a_p. It would be redundant (non-minimal) for such a plan to contain a_q as well. Moreover, we can ignore any other action that is not part of a minimal plan containing a_p after applying a_{last}.

Note that the above intuitions are correct provided that the last action applied, a_{last} does not achieve some landmark l. Our discussion below is under this assumption.

We start formulating the ideas above using the following two observations: Let $\pi_i = (a_1, \ldots, a_k)$ be a minimal sub-path to l, where a_k achieves l. (1) We know that $a_1, \ldots, a_k \in dfrA(l)$ because all actions of a minimal plan must be in $dfrA(l)$. (2) We know that $a_1, \ldots, a_{k-1} \in dfrA(x_1) \vee \cdots \vee dfrA(x_t), x_i \in pre(a_k)$, because the first $k-1$ actions must be in the dfrA sets of one of a_k's preconditions.

Notice, that if we knew the identity of a_k, we could prune all actions that are not in the DFR sets of one of a_k's preconditions. We usually cannot know who is a_k – it must belong to the set of achievers of l – but using a_{last} we can sometimes find a smaller set than $ach(l)$. Since we assumed that a_{last} does not achieve a fact landmark, it is one of the actions $a_1, \ldots, a_{k-1}$, as in the observations above. Let us assume the index of a_{last} is a_j ($j \leq k-1$). We can use a_{last} to identify which actions in $ach(l)$ could be in a minimal plan with a_{last}. That is, we can find which actions can act as a_k. Since we know what the possible a_k's are, we can

prune actions that cannot be a_{j+1} in such a plan.

To use these ideas we do the following: (i) Let $dfrb(v) = \{x|v \in dfr(x)\}$. That is, it is the set of facts x such that v can appear in a minimal plan that reaches them. (ii) Let $dfrb(a) = \bigcap_{v \in pre(a) \cup eff(a)} dfrb(v)$. $dfrb(a)$ is the set of facts such that a can be part of a minimal plan that reaches them. (iii) Let $PRE(l) = \{f|f \in pre(a) \wedge a \in ach(l)\}$, i.e., all preconditions of actions that achieve l. (iv) We can now define $cmt(a_{last}, l) = dfrb(a_{last}) \cap PRE(l)$. This is a set of fact commitments for l given a_{last}. At least one of these facts must be achieved in any minimal plan for l that includes a_{last}. That is, it is similar to a disjunctive landmark focused on minimal plans for l that include a_{last}.

Let us examine Figure 1 again, and assume our next landmark to achieve is G1 and that $a_{last} = a_1.dfrb(a_1) = \{s, 1, 2, 3, 4, G1, G2, t\}$, $PRE(G1) = \{2, 22\}$. Therefore $cmt(a_1, G1) = \{2\}$. We can notice three things: (1) We eventually must be able to apply action a_{99} (2) We must achieve node 2 (which is in $cmt(a_1, G1)$) (3) We also need to achieve node 22 (which is not in $cmt(a_1, G1)$). Since, we must achieve node 2, we can focus our search on this node until it is achieved and then turn our focus to node 22. We now formulate this observation

During A* search we divide the commitment set $cmt(a_{last}, l)$ into two sets. The first set $cmt_1(a_{last}, l)$ composed of the facts in the commitment set still unachieved. $cmt_2(a_{last}, l)$ is composed of achieved facts (facts in the state). The second set must be strengthened as follows: $cmt_2^*(a_{last}, l) = \{x|x \in pre(a) \wedge pre(a) \cap cmt_2 \neq \emptyset \wedge a \in ach(l)\}$. That is, we add to this set all preconditions of actions that uses facts from cmt_2 and achieve l. The reason this strengthening is necessary is because that cmt_2 could hold only part of the preconditions of some action in $ach(l)$ (as in the example above). If that precondition is already in the current state (like node 2) we will still need to achieve the rest of the preconditions (node 22). Now, during search, if a_{last} did not achieve a fact landmark, and action $a \notin ach(l)$, we can safely prune a, if there is no fact y in cmt_1 or cmt_2^* where $a \in dfrA(y)$. For example, if now $a_{last} = a_2$, then cmt_1 will be empty, $cmt_2 = \{2\}$ and $cmt_2^* = \{2, 22\}$, forcing us to turn our focus to node 22.

Let us now look at another example from Figure 1. If we will remove from the RCG in Figure 1 action a_{99} and fact node G1, we will get an RCG with only one fact landmark G2 (besides s). Now (using the mentioned RCG), let us assume the last action taken was a_1 which did not achieve a fact landmark. $dfrb(a_1) = \{s, 1, 2, 3, 4, G2, t\}$, $PRE(G2) = \{2, 22\}$. Therefore $cmt(a_1, G2) = \{2\}$. So, we can prune any action that is not part of $dfrA(2)$. Notice, that if a_8 would have another precondition besides node 2, we could miss it based on $dfrb(a_1)$ alone. In that case, We could safely prune until node 2 is in state, then we would have to strengthen cmt_2 to "see" the other precondition.

Sub-goal Bounds

Our third and last pruning technique uses the DFR differently. Let us assume we have some upper bound u on the cost of the entire plan to G. We'd like to get an upper bound on the cost of achieving G', our current landmark. We do so by first estimating the state $s(G')$ reached when G' is achieved. Then using an admissible heuristic to obtain a lower bound d on the cost of achieving the final goal G from $s(G')$. Now, $u - d$ is an upper bound on the cost of achieving G'. We use the $dfr(G')$ as our estimate of the state $s(G')$. This is justified because we know that all facts achieved on route to G' using some minimal plan are included in $dfr(G')$. This gives us a "quick and dirty" method of generating an upper bound on the cost of achieving G' which requires only a single computation of the heuristic function (very cheap) and a single computation of $dfr(G')$, which is computed anyway during preprocessing.

An upper bound u on plan cost provides a simple (well known) method for pruning during A^* search. Any state s such that $f(s) = g(s) + h(s) > u$ can be pruned (assuming an admissible h value). Our main observation here is that this idea can be used at each stage implemented in the GB framework, since their decomposition approach reduced the problem of searching for a single plan to one of searching for many (simpler) plans (see Lemma 1).

Unlike the first two pruning methods discussed so far, which would work for any I', G', the bounds generation method takes a slightly more global perspective, and will generate the upper bounds for all the relevant subproblems obtained in GB's decomposition at preprocessing time.

First, we get an upper bound on the cost of the entire plan. To get an upper bound on plan cost, one can use h^{add}, h^{max} (Bonet and Geffner 2001) to guide the selection for a satisficing plan, much like in FF (Hoffmann and Nebel 2001). A stronger technique will be to use algorithms described in (Keyder and Geffner 2009). The cost of this plan is an upper bound u. Next, recall that GB's decomposition method generates a sequence $L = (l_1, l_2, \ldots, l_k)$ of ordered landmarks (see Lemma 1). Using this sequence, we create a sequence of partial assignments that an optimal plan must reach. These partial assignments are supersets for minimal plans, specifically, the plan to l_1 must reside in $dfr(l_1)$.

Since in DFPlan propositions achieved remain true forever, as long as we keep the landmark ordering, a minimal plan to $\{l_1, l_2\}$ must lead to a state in which the achieved propositions are a subset of $dfr(l_1) \cup dfr(l_2)$. More generally, the superset of any state achieving l_j is $dfr(l_1) \cup \cdots \cup dfr(l_j)$.

Using an admissible heuristic function h, we now get the following respective bounds on the cost of reaching each of the landmarks: Let $B = (u - h(dfr(l_1)), u - h(dfr(l_1) \cup dfr(l_2)), \ldots, u - h(dfr(l_1) \cup \cdots \cup dfr(l_k)))$, where $h(dfr(v))$ is an admissible heuristic of state $s = \{x|x \in dfr(v)\}$ to the goal. This computation needs to be carried out once only, before search commences.

During search, for each current state s, each applicable action a, and next landmark to achieve l_i, we can ask: is $g(s) + cost(a) \leq B(i)$? If the answer is *no*, we can prune a. If all actions applicable at s are pruned, s is a dead-end for optimal planning.

Empirical Results

We implemented the ideas described in the previous sections in the Fast Downward framework (Helmert 2006). The DFR

algorithm was implemented to run once at preprocess time. The Q set in the algorithm was implemented as a queue. Pruning itself is done while in search, with respect to the next fact landmark to achieve. We implemented two variants: DFR1 uses the first pruning method only (basic DFR pruning). DFR2 uses all three pruning methods. We evaluated their performance in the context of heuristic search (using A^*) using LM-Cut and compare to the results in the GB paper in Table 1,2. Domains used are delete-free versions of IPC problems. The time limit is 0.5/5/30 minutes per problem. We compare LM-Cut alone to GB + LM-Cut, DFR1 + LM-Cut and DFR2 + LM-Cut. The overall performance (coverage, time scores, expansion scores) shows that pruning in general has an advantage over LM-Cut alone in almost all of the domains. There are some domains where the GB method is better than the basic pruning (DFR1), like logistics98 and satellite. For the satellite domain this can be the result of the path-commitment based pruning, since, DFR2 is better than the GB method in this domain. For logistics98, the GB method is better than DFR2 so this is probably due to minimization of disjunctive action landmark that is done in the GB method for which we have no parallel technique (although it is theoretically possible). We show DFR2 with all three pruning methods because our experiments showed that the contribution of sub-goal bounds is minor. Still, we believe it to be an interesting general technique that could be enhanced using stronger heuristics.

We also tried our pruning procedure on the seed-set problems with zero-cost actions. Zero-cost actions are challenging to existing solvers which cannot solve any instance of this problem (Gefen and Brafman 2011). Pruning, in this case, is not sufficient to overcome this method. However, when we augment pruning with a simple procedure that applies all applicable zero-cost actions, then pruning+blind-search+zero-cost-procedure solves all the seed-set problems.

Domain	LM-Cut			GB method			DFR1 + LM-Cut			DFR2 + LM-Cut			
Seconds	30	300	1800	30	300	1800	30	300	1800	10	30	300	1800
airport(50)	26	33	38	40	48	49	29	42	49	23	28	43	48
blocks(35)	35	35	35	35	35	35	35	35	35	35	35	35	35
depot (22)	4	6	7	10	10	12	9	10	10	9	9	10	10
driverlog (20)	13	14	14	13	14	15	14	15	15	13	14	15	15
elevators-opt08-strips (30)	5	7	7	6	7	9	5	7	8	4	5	6	8
freecell (80)	1	4	6	0	1	1	1	4	6	1	1	4	6
grid (5)	1	1	2	1	1	2	1	2	2	1	1	2	2
gripper (20)	20	20	20	20	20	20	20	20	20	20	20	20	20
logistics00 (28)	23	23	23	28	28	28	28	28	28	28	28	28	28
logistics98 (35)	7	8	8	10	17	17	12	15	16	11	13	16	16
miconic (150)	150	150	150	128	150	150	150	150	150	150	150	150	150
mprime (35)	19	24	27	17	21	24	19	24	27	17	20	25	28
mystery (30)	23	25	26	20	25	26	22	26	26	17	23	26	26
openstacks-opt08-strips (30)	5	6	7	5	7	8	4	5	7	30	30	30	30
openstacks-strips (30)	5	5	5	5	5	5	5	5	5	5	5	5	5
parcprinter-08-strips (30)	23	23	23	29	29	30	30	30	30	30	30	30	30
pathways--noneng (30)	5	5	5	5	7	8	5	5	8	5	5	6	8
pegsol-08-strips (30)	24	27	27	24	26	29	25	27	27	18	25	27	27
pipesworld-notankage (50)	12	15	17	10	13	17	12	15	17	10	12	16	17
pipesworld-tankage (50)	7	10	10	6	7	9	7	10	10	6	7	10	10
psr-small (50)	50	50	50	50	50	50	50	50	50	50	50	50	50
rovers (40)	8	12	12	17	20	21	18	20	21	20	21	25	28
satellite (36)	6	6	7	6	8	10	6	7	7	6	7	9	11
scanalyzer-08-strips 30)	13	14	15	8	10	14	10	14	15	9	10	14	15
sokoban-08-strips 30)	19	21	25	22	24	26	20	22	26	17	20	22	26
tpp (30)	9	12	13	14	15	16	17	21	21	19	19	21	21
transport-opt08-strips (30)	9	10	12	9	12	12	9	10	12	9	10	10	10
trucks-strips (30)	6	7	9	24	30	30	30	30	30	30	30	30	30
woodworking-opt08-strips 30)	17	18	19	27	27	28	27	27	29	27	27	27	29
zenotravel (20)	11	13	13	13	13	13	13	13	13	13	13	13	13
Total (1116)	556	604	632	602	680	714	633	689	720	633	668	725	752

Table 1: Coverage per domain. time limit per problem: 0.5 (30s), 5 (300s), 30 (1800s) minutes (sec.). We add a DFR2 column of 10 seconds to see where it is tied with 30 minutes of LM-cut alone.

Summary and Related Work

Using the graph-theoretic notion of domination, we were able to construct a new simple domination-free reachability method for delete-free planning and use it within the decomposition framework of GB to obtain better performance than the more complicated pruning techniques of (Gefen and Brafman 2012). In addition, we leveraged the decomposition to introduce a bounds-based pruning method. Our method was shown to be more effective in solving delete-free planning problems. The increased coverage is not large – we solve 16 more problems than GB – yet in the context of optimal planning, this is considered non-trivial progress. Perhaps more importantly, our work is based on a new construction that improves upon one of the more fundamental tools in the analysis of planning problems – that of reachability – and we are not aware of any similar such construct in the literature.

The DFR algorithm shares some similarity with the *first achiever analysis* (Haslum, Slaney, and Thiébaux 2012; Pommerening and Helmert 2012) which removes all operators that are not *first achievers* of any variable (Richter, Helmert, and Westphal 2008). Such actions can not be part a minimal plan. In fact, Line 10 in the DFR algorithm prevents the propagation of information through actions which are not first achievers, treating them as if they were not in the RCG. The DFR algorithm then continues to detect actions that cannot be part of a minimal plan but still would not be removed by the first achiever analysis. For example, in Figure 2 the DFR set of nodes $1, G_1$ does not contain actions a_2, a_3 which will not be part of a minimal plan even though they are first achievers.

A similar notion to path-commitment in regular planning was recently introduced by (Karpas and Domshlak 2012). In this paper causal links are used to infer constraints that must be satisfied by an optimal plan having some known prefix. The constraints are used to enhance a heuristic evaluation. Therefore, the notion of optimal plan plays a similar role as the minimal plan in the path-commitment method. It could be the case that using the notion of optimal plan is too restrictive, and using minimal plan instead would simplify this work.

To our knowledge, the work of GB and this paper are the only work in the literature that focuses on pruning in delete-free problems using A* search. There are, however, some other recent works on delete-free planning: (Pommeren-

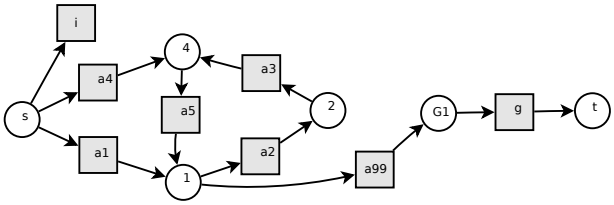


Figure 2: An RCG where first achiever analysis would not remove actions (but i).

planner	Time score	Expansion score
LM-cut	46.58	45.75
GB method + LM-cut	53.05	57.07
DFR1 + LM-cut	54.82	55.59
DFR2 + LM-cut	58.36	58.89

Table 2: time limit per problem: 30 minutes. Scores are average of domain score values for each planner. Scores based on (Richter and Helmert 2009). Logarithmically scaled scores between 0 ($\geq 300s$ and ≥ 1000000 expansions resp.) and 100 ($\leq 1s$ and ≤ 100 expansions resp.).

ing and Helmert 2012) share some similarities regarding upper-bounds, although they don't use A* search. (Haslum, Slaney, and Thiébaux 2012) try to solve delete-free problems by generating minimal disjunctive action landmarks and then applying minimal-hitting set procedure to find an optimal plan. There has been also quite a few recent papers dealing with action pruning: Stratified Planning (SP) (Chen, Xu, and Yao 2009), Expansion core (EC) (Chen and Yao 2009), Bounded intention Planning (BIP) (Wolfe and Russell 2011), Symmetries (Coles and Coles 2010; Fox and Long 1999; Pochter, Zohar, and Rosenschein 2011). Of these, the only work we are aware of that is landmark based is the SAC algorithm (Xu et al. 2011) and its use of disjunctive action landmarks is closer in spirit to GB.

The DFR sets in this paper are generated at preprocessing time giving them an advantage over the GB method which is repeatedly applied at run-time. This can be seen in the fact that even though GB use an extra minimization step to generate minimal disjunctive action landmark, the basic DFR method (DFR1) has, in general, a better average time score, while DFR2 has also a better average expansion scores. These observations supports our belief that the DFR set is a strong theoretical concept, and we believe that it can be used by other methods – any method that can benefit from the notion of minimal plans rather than (any) plan could potentially benefit from the use of DFR sets.

Acknowledgements: We are grateful for the extreme detailed and useful comments made by the anonymous reviewers and also to Malte Helmert and Florian Pommerening for their help in experiments. The authors were partly supported by the Paul Ivanier Center for Robotics Research and Production Management, and the Lynn and William Frankel Center for Computer Science.

References

Bacchus, F.; Domshlak, C.; Edelkamp, S.; and Helmert, M., eds. 2011. *Proceedings of the 21st International Conference on Automated Planning and Scheduling, ICAPS 2011, Freiburg, Germany June 11-16, 2011*. AAAI.

Blum, A., and Furst, M. L. 1997. Fast planning through planning graph analysis. *Artif. Intell.* 90(1-2):281–300.

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

Bylander, T. 1994. The computational complexity of propositional strips planning. *Artif. Intell.* 69(1-2):165–204.

Chen, Y., and Yao, G. 2009. Completeness and optimality preserving reduction for planning. In *IJCAI*, 1659–1664.

Chen, Y.; Xu, Y.; and Yao, G. 2009. Stratified planning. In *IJCAI*, 1665–1670.

Coelho, H.; Studer, R.; and Wooldridge, M., eds. 2010. *ECAI 2010 - 19th European Conference on Artificial Intelligence, Lisbon, Portugal, August 16-20, 2010, Proceedings*, volume 215 of *Frontiers in Artificial Intelligence and Applications*. IOS Press.

Coles, A. J., and Coles, A. 2010. Completeness-preserving pruning for optimal planning. In Coelho et al. (2010), 965–966.

Fox, M., and Long, D. 1999. The detection and exploitation of symmetry in planning problems. In Dean, T., ed., *IJCAI*, 956–961. Morgan Kaufmann.

Gallo, G.; Longo, G.; Pallottino, S.; and Nguyen, S. 1993. Directed hypergraphs and applications. *Discrete Applied Mathematics* 42(2-3):177–201.

Gefen, A., and Brafman, R. I. 2011. The minimal seed set problem. In Bacchus et al. (2011).

Gefen, A., and Brafman, R. I. 2012. Pruning methods for optimal delete-free planning. In McCluskey et al. (2012).

Gerevini, A.; Howe, A. E.; Cesta, A.; and Refanidis, I., eds. 2009. *Proceedings of the 19th International Conference on Automated Planning and Scheduling, ICAPS 2009, Thessaloniki, Greece, September 19-23, 2009*. AAAI.

Haslum, P.; Slaney, J. K.; and Thiébaux, S. 2012. Minimal landmarks for optimal delete-free planning. In McCluskey et al. (2012).

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In Gerevini et al. (2009).

Helmert, M. 2006. The fast downward planning system. *J. Artif. Intell. Res. (JAIR)* 26:191–246.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: fast plan generation through heuristic search. *J. Artif. Int. Res.* 14(1):253–302.

Hoffmann, J.; Porteous, J.; and Sebastia, L. 2004. Ordered landmarks in planning. *J. Artif. Intell. Res. (JAIR)* 22:215–278.

Karpas, E., and Domshlak, C. 2012. Optimal search with inadmissible heuristics. In McCluskey et al. (2012).

Keyder, E., and Geffner, H. 2009. Trees of shortest paths vs. steiner trees: Understanding and improving delete relaxation heuristics. In Boutilier, C., ed., *IJCAI*, 1734–1739.

Keyder, E.; Richter, S.; and Helmert, M. 2010. Sound and complete landmarks for and/or graphs. In Coelho et al. (2010), 335–340.

McCluskey, L.; Williams, B.; Silva, J. R.; and Bonet, B., eds. 2012. *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012, Atibaia, São Paulo, Brazil, June 25-19, 2012*. AAAI.

Perl, Y., and Shiloach, Y. 1978. Finding two disjoint paths between two pairs of vertices in a graph. *J. ACM* 25(1):1–9.

Pochter, N.; Zohar, A.; and Rosenschein, J. S. 2011. Exploiting problem symmetries in state-based planners. In Burgard, W., and Roth, D., eds., *AAAI*. AAAI Press.

Pommerening, F., and Helmert, M. 2012. Optimal planning for delete-free tasks with incremental lm-cut. In McCluskey et al. (2012).

Porco, A.; Machado, A.; and Bonet, B. 2011. Automatic polytime reductions of np problems into a fragment of strips. In Bacchus et al. (2011).

Richter, S., and Helmert, M. 2009. Preferred operators and deferred evaluation in satisficing planning. In Gerevini et al. (2009).

Richter, S.; Helmert, M.; and Westphal, M. 2008. Landmarks revisited. In *AAAI'08: Proceedings of the 23rd national conference on Artificial intelligence*, 975–982. AAAI Press.

Tarjan, R. E. 1974. Testing flow graph reducibility. *Journal of Computer and System Sciences* 9(3):355–365.

Wolfe, J., and Russell, S. J. 2011. Bounded intention planning. In Walsh, T., ed., *IJCAI*, 2039–2045. IJCAI/AAAI.

Xu, Y.; Chen, Y.; Lu, Q.; and Huang, R. 2011. Theory and algorithms for partial order based reduction in planning. *CoRR* abs/1106.5427.

A Tale of t Metrics

Mark Roberts and **Adele E. Howe** and **Indrajit Ray**
Computer Science Dept., Colorado State University
Fort Collins, CO 80524, USA
email: {mroberts,howe,indrajit}@cs.colostate.edu

Abstract

Classical planning systems tend to focus on producing one best solution according to a single objective function or metric. In domains with metrics that can be linearly combined, action costs are a suitable choice for modeling the metric(s). In some domains, however, metrics can interact in subtle ways that may be inappropriate for a weighted objective function. Existing benchmark domains explore a small subset of potential metric interactions, typically examining one or two non-temporal metrics that rarely interact within the same action. We create a synthetic domain that allows us to systematically vary two non-temporal metrics with respect to each other and to plan length. From this domain, we make observations about the search behavior of bounded suboptimal search. We find that search performance (a) is the poorest when the two metrics are either collinear or uncorrelated regardless of their strength of correlation with plan length; (b) is best when the metrics correlate with each other in a simple curvilinear fashion that is strongly correlated with plan length; (c) degrades for curvilinear functions as the functions become weakly correlated with plan length, and (d) degrades as the metrics interact with each other or plan length in subtle or uncorrelated ways.

Domains with multiple, competing objectives (e.g., minimizing time versus money) are common but have been simplified in planning research to avoid explicitly reasoning about the trade-offs. The net-benefit track in the 2006 planning competition (Gerevini and Long 2005) and the focus on plan quality in recent International Planning Competitions (IPCs) using action costs have taken steps in this direction, capturing the trade-offs via a weighted evaluation function.

Temporal and resource reasoning lies at the heart of many significant problems in planning research. A wealth of literature exists for producing (diverse) temporal plans using a single plan metric (e.g., SAPA (Do and Kambhampati 2003), LPG-td (Gerevini, Saetti, and Serina 2006), COLIN (Coles et al. 2012)). In most cases, these temporal+metric planners embed deep reasoning to solve specific resource and time constraints that lie at the junction of planning and scheduling. It is difficult to assess the contribution of the non-temporal metrics when they are combined with temporal concerns in a weighted objective function.

We focus on numeric-only metric domains rather than examining temporal+metric planning. Our approach is to isolate the *non-temporal* metrics with the hope of assessing search behavior for these metrics. The findings we present complement the temporal+metric planning literature by probing deeper into the non-temporal metrics. We discuss how metrics are used in existing IPC benchmarks and show that these domains rarely explore interactions between the metrics *within the same action*. We then highlight some unique qualities of the domain that motivates our study: cybersecurity for the personal home computer user. The security domain does not require deep temporal or resource reasoning, but it does have multiple non-temporal metrics by which plans are evaluated.

To study metric interaction in isolation, we create a synthetic domain that allows us to vary the interactions of two metrics, x and y, with each other and with plan length. The interactions include uniform random plus three structured interactions (and their "mirrored inverses"): linear, sigmoidal, and polynomial. We examine search cost and the ability to find minimal solutions for A^*_ϵ (Pearl and Kim 1982), as implemented in MetricFF (Hoffmann 2003). We find that this implementation of A^*_ϵ has the most difficulty finding minimal solutions for collinear and random interactions, while it is most successful for simple curvilinear interactions that are strongly correlated with plan length. Uniformly scaling the metrics so that they less strongly correlate with plan length decreases the ability of A^*_ϵ to find minimal solutions. Weighting one metric more heavily than the other degrades search performance as well. These findings, though limited, suggest directions for future work in characterizing search behavior when metrics interact.

Multi-metric Domains

Our work is motivated by a non-temporal domain: identifying potential breaches for a personal computer system (Roberts et al. 2011). We are extending this domain to use four metrics not readily combined into a single objective function. The *likelihood of attack* and *cost of attack* characterize the risk associated with doing nothing about a potential threat, while *utility to the user* and *cost of intervening* characterize the reward of better securing the system while minimizing user irritation. Searching for breaches in the security domain is unique in that a property called *monotonicity* (Ammann, Wijesekera, and Kaushik 2002) eliminates cy-

cles and bounds the length of plans, but the interacting metrics create a challenge for search because they may interact in subtle ways. For example, phishing attacks have a high privacy violation cost because a novice user may provide financial or personal details to a malicious third party, thus subjecting themselves to unpleasant consequences. Since such attacks can only happen through email, a low cost intervention might be to disallow email. However, this is unlikely to be a desirable intervention from the user's perspective given the ubiquity of email use for communication. A higher-cost intervention could be educating the user, but failing to do this well may increase the risk and/or cause the user to be more confident in their ability than they actually are. The agent of this security application must reason about a set of alternative plans which balance the complex trade-offs of how likely an attack will lead to a compromised system, of the costs associated with intervening against such attacks, and of the repair costs for a (potentially) compromised system. In short, there is no single, best solution.

Existing multi-metric benchmark domains provide some inspiration in how to approach the multi-metric security domain. We highlight six non-temporal benchmark domains from IPC-2002 (Long and Fox 2003) that are represented in PDDL 2.1, Level 2 (Fox and Long 2003): Depots, DriverLog, Satellite, Settlers, Rovers, ZenoTravel. These domains were a significant advancement toward metric planning and include at least two unique metrics in addition to plan length. Radzi (2011) shows that the single, weighted-sum metric in most of these domains interacts with plan length in ways that make the problem *straightforward* to solve[1]. Only Settlers has metrics that interact with plan length in an interesting way that is not correlated with plan length. Radzi further shows that, for most domains, the metrics interact within a set of repeatable action sequences (e.g., in satellite, turning the camera to take an image) or the metrics interact within a producer/consumer model where one action increases the availability of a resource that is later consumed by another action. While these benchmarks have a variety of metrics that do interact in constrained ways, the metrics rarely contradict plan length or interact *within* the same action.

In other multi-metric applications, the metrics may have competing or subtle interactions that occur within an action. The security application is one example. Another example is an autonomous system that needs to balance the overall objectives of a mission while assessing metrics such as power consumption, remaining time, risk and safety, level of required coordination, communications delay, etc. Yet another example is a human travel agent using a mixed-initiative planning system to book an itinerary that balances cost, total travel time, airport preferences, choice of seating, etc.

[1]More specifically, Radzi (2011) distinguishes metric straightforwardness into a spectrum of strictly straightforward (plan length is a proxy for the metric), straightforward (plan length may be a proxy), semi-straightforward (plan length and the metric may diverge but the metric is monotonic in the plan length), and expressive (the metric may contradict plan length). Radzi examines several new domains for the semi-straightforward category and leaves to future work the set of expressive domains, of which we believe the security domain may be one case.

Controlling for Metric Interactions with a Synthetic Domain

To study the impact of metric interactions on search in a controlled way, we create a synthetic domain. All solutions in the domain must have the same plan length and the metrics must be systematically varied across the operators of the domain. Figure 1 presents a pictorial view of the synthetic domain, which is essentially a fully connected planning graph. Nodes in this graph are states and arcs are transitions (operators). An $m \times n$ graph has m layers with each layer containing n states. In planning terms, m determines the plan length of any valid solution, while n determines the granularity of the metrics as discussed below.

States are labeled $s_{ij} \in \mathcal{S}$, where $i = \{0, 1, .., m\}$ indicates the layer, and $j = \{0, 1, .., (n-1)\}$ indicates one of the states within a layer. The initial ($i = 0$) and goal ($i = m$) layers have only one state; these are respectively labeled 'Init' and 'Goal' in Figure 1.

Transitions between states are ordered pairs $(s_{ij}, s_{i'j'}), i < i', j < j'$ and labeled $a_{(ij,i'j')} \in \mathcal{A}$. The middle layers of the graph are fully connected with the next layer, so there is a path from any state in level i to all states at the next level ($i' = i + 1, j' = \{0, 1, .., (n-1)\}$). There are $(m-2)n^2 + 2m$ transitions. Each arc in the graph represents a single operator, where the precondition links to a state at level i and the postcondition links to a state at level $i + 1$. This version of the synthetic domain is lacking some key features of many applications: there are no negative effects and no cycles[2]. We plan to add these features into future studies of the synthetic domain.

Adding Metrics To The Synthetic Domain

Table 1 shows the weights of the transition matrix for the 3×3 problem. The weight of an arc is set to the value of the node it leads into (shown as a small integer at the top of the states in Figure 1). The arcs leading into the left-most states ($a_{i0} \forall i$) are all set to 0. Arcs leading into the right-most states ($s_{i(n-1)} \forall i$) are set to 1000. Arcs in the middle interpolate the distance between 0 and 1000. Formally, the values depend on the jth column: $a_{ij} = j * 1000/(n-1)$, where $\forall ij, 0 < i < m, 0 \leq j < n$. Thus, the number of states in each layer, n, determines the granularity of the metrics across the graph.

The three metrics x, y and z are functions of the transition matrix; we only set the value of x or y if a transition exists. The ranges of all metrics are integers $[1, 1000]$ to ensure some numeric effect in every action. Values of x are obtained: $x_{(ij,i'j')} = max(1, w_{(ij,i'j')})$. For brevity, we drop the subscripts and say $x = max(1, w)$. To obtain y, we apply functions that control the interaction between x and y. These functions are listed in Figure 2 and plotted in Figure 4. The random (ran) function is a no-interaction baseline. The other functions vary the interaction with functions (and their "mirrored inverses"): linear, lin (nil), sigmoidal, sig (gis), and polynomial, pol (lop). To aid in reading, note that the letters are reversed for each inverse function. The

[2]The monotonicity property (Ammann, Wijesekera, and Kaushik 2002) of the security domain motivates this restriction

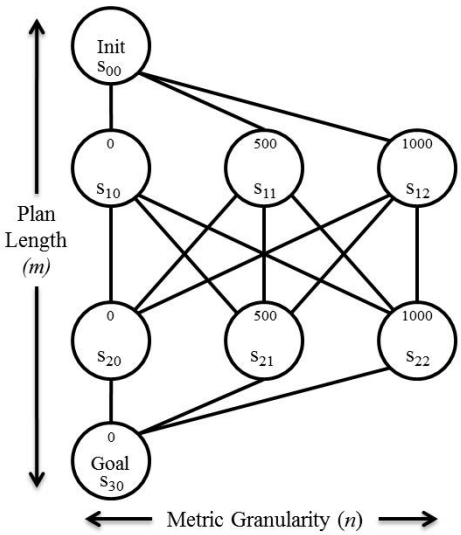


Figure 1: A graph of the 3×3 synthetic domain. Transition weights are at the top of the node an arc leads into.

$$x = max(1, w) + \epsilon$$

$$ran(w) = c \cdot uniform(1, 1000) + \epsilon$$

$$lin(w) = x = max(1, w) + \epsilon$$
$$nil(w) = max(1, 1000 - w) + \epsilon$$
$$sig(w) = round(1000 * (1/(1 + e^{-(w-500)/110}))) + \epsilon$$

$$gis(w) = round(1000 * (1/(1 + e^{(w-500)/110}))) + \epsilon$$
$$pol(w) = round(1050 * (1/exp(w/10)^{0.05})) + \epsilon$$
$$lop(w) = round(1000 * (1000^3 - w^3/1000^3)) + \epsilon$$

Figure 2: The functions used to vary metric interaction. See Figure 4 for a plot of each function.

polynomial function is rotated to provide a more interesting optimization metric. The fractional component of any function is truncated, and random noise $\epsilon = round(\mathcal{N}(0, 10))$ is added to all functions to generate randomized problems. If the function or the random noise cause a negative value, it is set to 1.

We further partition the functions into two sets. The "easy" functions $E = \{ran, lin, nil, sig\}$ show no real trade off between x and y or have many minimal solutions (as in nil). The "difficult" functions $D = \{gis, pol, lop\}$ exhibit a trade-off with a small number of minimal solutions, although multiple symmetric solutions may exist because we discretize the functions. The ordering of the functions in Figure 2 and in later plots maintains the relative order of the easy and difficult partitions.

We apply the transition matrix and the values of x and y to create PDDL problems for a specific $m \times n$ size; this

	1,0	1,1	1,2	2,0	2,1	2,2	3,0
0,0	0	500	1000				
1,0				0	500	1000	
1,1				0	500	1000	
1,2				0	500	1000	
2,0							0
2,1							500
2,2							1000

Table 1: The transition matrix for 3×3.

```
(define (domain synthetic-4-3-nil)
  (:requirements :equality :typing :fluents)
  (:types State)  (:predicates (state-active ?s - State))
  (:functions (x) (y) )
  (:action Apply-00-10  :parameters (?state-00 - State)
   :precondition (and (state-active ?state-00)
                      (= ?state-00 State-00) )
   :effect (and (state-active State-10 )
                (increase (x) 1)
                (increase (y) 1000) ) )
   ...)

(define (problem synthetic-4-3-nil-x)
  (:domain synthetic-4-3-nil)
  (:objects State-00 - State
            ...
            State-40 - State)
  (:init (state-active State-00) (= (x) 0) (= (y) 0))
  (:goal (state-active State-40))
  (:metric minimize (x) ) )
```

Figure 3: Partial PDDL for the domain and problem $4 \times 3^x_{nil}$.

yields one domain file and 21 problem variants (i.e., seven functions, three metrics). Metrics are applied in the operator effects with `(increase (f) value)`. The problems are labeled $m \times n^{\text{metric}}_{\text{function}}$, where the function is one of $\{ran, lin, nil, sig, gis, pol, lop\}$ and the metric is one of $\{x, y, z\}$. Figure 3 shows the PDDL for $4 \times 3^x_{nil}$.

Method

We used the A^*_ϵ algorithm as implemented in the newest version of MetricFF (Hoffmann 2003), a planner that directly supports the full complement of PDDL 2.1, Level 2. Instead of popping the best solution ($s = min(q)$) from the top of the priority queue at each iteration, A^*_ϵ (Pearl and Kim 1982) selects a solution from the top K solutions (i.e., a focus list) on the top of the queue. A solution s' is in K if $f(s') <= 5f(s)$ and $h(s') = h(s)$. We only use MetricFF with cost optimization enabled. This implementation does not use helpful actions nor the initial enforced hill climbing stage. We use $m \times n = 15 \times 29$ because it was as large as MetricFF could still handle.

We show box-and-whisker plots of CPU time to a completed plan. For plan cost, A^*_ϵ can produce suboptimal so-

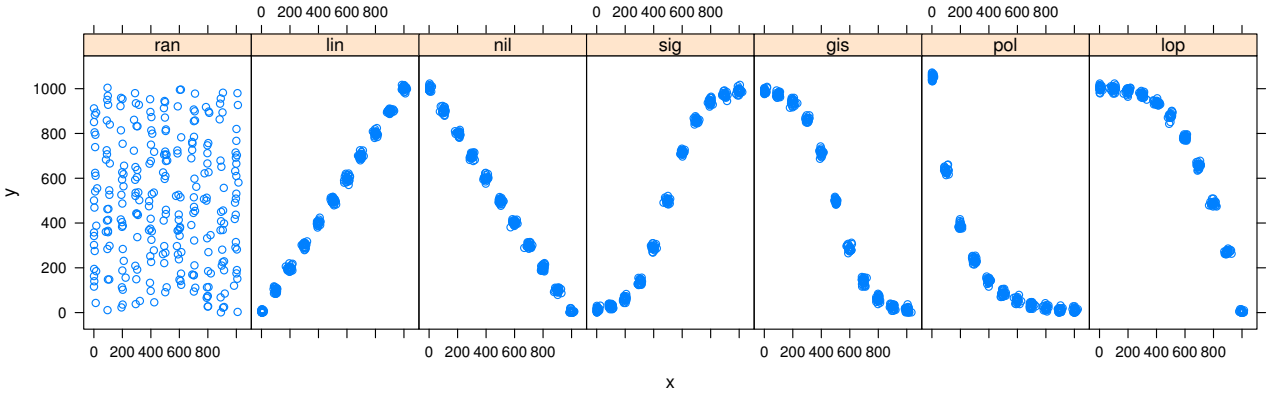


Figure 4: Examples of the interaction between metrics x and y.

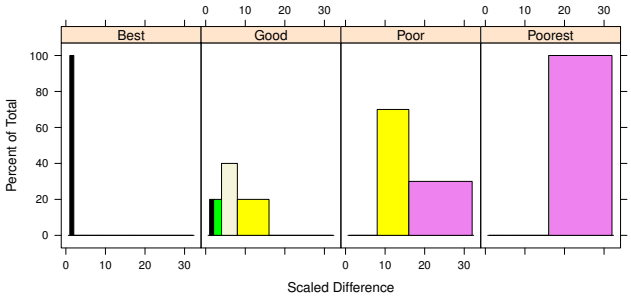


Figure 5: An example histogram explaining how to read later plots. Thinner bars to the left are best. Wider bars to the right indicate poorer performance. The y-axis is the percent of total problems (out of 30) that fall into each bin.

lutions. So we state A^*_ϵ ϵ-minimizes a function if most of its solutions fall within 8 times the optimal. We plan to explore more deeply the weight value within A^*_ϵ to determine the best setting. To plot how well A^*_ϵ minimizes the plan, each plan π is first scaled, $cost(\pi)/cost*$, where $cost*$ is the minimal solution found using a simple dynamic programming algorithm. A minimal plan receives a 1; other solutions are factors of how much worse they are from the minimal solution. We then histogram these scaled values for the 30 problems from each combination of function and metric using bin sizes of $1, 2, 4, 8, 16,$ and $>= 32$. Figure 5 shows a comparative example of what we want to see for optimal and worsening solution quality. Thicker bars indicate higher variance, while the right-most bars indicate the poorest relative performance. Note that the y-axis in these plots is the percent of total problems, which for each function/metric combination is 30 problems.

All experiments are run on a 2.7Ghz Quad Core i7 with a time limit of 15 minutes and memory limit of 1GB.

The Impact of Metric Interaction

Many planners are designed to minimize a single objective function. So we expect to observe little performance difference when minimizing either x or y alone.

Hypothesis 1: *For all functions, (a) A^*_ϵ will ϵ-minimize both x and y (b) with insignificant differences in CPU time between the two metrics.*

Table 2 (top) – this and all subsequent tables and figures are at the end of the paper – shows the runtimes of x and y along with the p-values for a pairwise t-test between x and y. These are also plotted on a log scale in Figure 6 (top). All the runtimes between the two metrics were significant ($p < 0.0073$) except the linear functions[3]. Figure 7 (left) shows a histogram of the scaled differences for how well A^*_ϵ minimizes x and y. A^*_ϵ has trouble finding the ϵ-minimal solutions.

There is a marked difference between minimizing x and minimizing y for some problems. Both metrics have a range that is represented equally in the actions at each layer of the graph. However, x is sampled at specific points that are interpolated between $[1, 1000]$ while y is sampled uniformly randomly in that same range. This sampling bias is evidenced in the vertical 'bands' for x seen in Figure 4 that are absent for y. This leads to more potential values for y than x, which appears to be more challenging for search. We plan to confirm this explanation in future work.

So we can conclude that, for the most part, hypothesis 1(a), is not validated as ϵ-minimal solutions are not found. We also conclude that hypothesis 1(b) is not supported, as there are significant differences in CPU time between minimizing x and y.

Comparing Easy and Difficult Functions

For minimizing z, we expect to see quality and performance degrade as the xy trade off becomes more challenging. In

[3]The Bonferroni adjustment controls the experiment-wise error of 7 pairwise comparisons at $\alpha = 0.05$, so the critical value for p is $0.0073 = 1 - (1 - 0.05^{1/7})$.

the following two hypotheses, we examine planner performance on the easy and difficult metric interactions. The intuition behind this hypothesis is that collinear functions are easy but non-collinear functions are difficult. So, finding the minimum will take longer and happen less frequently under difficult interactions.

Hypotheses 2: *For the easy functions (ran, lin, nil, sig), (a) A^*_ϵ will ϵ-minimize z and (b) with insignificant differences between the CPU time of minimizing x, y, and z.*

Figure 7 (left) shows that A^*_ϵ successfully finds ϵ-minimal solutions in z for nil, gis, pol, lop. In terms of runtime, Figure 6 (top) shows there is often a significant cost (10 to 100 times more) to minimizing z except in the simplest collinear functions lin and sig. The runtime for minimizing z is significantly different except for lin.

Hypothesis 3: *For the difficult functions (gis,pol,lop), (a) A^*_ϵ will not find the global minimums in z, and (b) the runtime will be significantly different between D and E. Further, (c) the runtime will be insignificant between pol and lop, but (d) will be significantly different between gis and $\{pol, lop\}$.*

Figure 6 (top) shows that the runtimes between the E and D problems are distinct, with the D problems usually taking more runtime. The runtimes between pol and lop, while overlapping, are statistically different; a Tukey HSD test run on the runtimes of both the E and D does not group any of the functions together. Figure 7 shows that A^*_ϵ successfully finds the minimal solutions for gis, pol, lop.

So we conclude that CPU time increases for the difficult functions but there is not otherwise similar performance with the easy/difficult groupings. A^*_ϵ finds minimal solutions for all three functions.

The Impact of Plan-length Correlation (PLC)

Due to using the planning graph as a heuristic function, many planners may implicitly rely on plan length to guide search. If this is true then we should expect to observe significant differences between the solutions and the runtime behavior of solving plan length correlated problems versus uncorrelated problems.

Unfortunately, we cannot completely test these intuitions with the current version of MetricFF, which only supports actions that move the metric in the opposite direction from the objective (i.e., when minimizing and given a $value$ and function f for the action, we can only use `(increase (f) value)` and cannot use `(decrease (f) value)` or `(increase (f) -value)`. This limits the kind of evidence we can collect for the behavior of best first search w.r.t. controlling PLC. So we discuss two additional experiments that assess how sensitive A^*_ϵ is to scaling x or y.

Recent research shows that cost-based search is sensitive to the ratio of the operator costs (i.e., (Wilt and Ruml 2011), (Cushing, Benton, and Kambhampati 2011)). Sroka and Long (2012) assess the metric sensitivity of planners and show that MetricFF (and other planners) can generate more diverse solutions by varying the constraintedness of resources in a logistics domain. We are interested in how the search behavior changes when metric interactions vary in addition to the scaling.

Uniformly Scaling x and y

We create a set of under-correlated problems that simply scale the original metric values in actions by 0.3. Figure 6 (bottom) and Table 2 show the runtime distributions for the under-correlated problems. The results parallel those of the correlated problems except for a more significant difference in gis, which can be explained by an outlier present in the original runs.

Figure 7 (right) shows histograms of the scaled difference from minimal solutions for the under-correlated problems. When minimizing x or y, A^*_ϵ finds equal or better solutions for the under-correlated problems, except for ran. In contrast, when minimizing z, A^*_ϵ finds worse solutions except for lin and sig.

Scaling either x or y in z

Another way we can control for PLC is to scale either x or y when minimizing z. This leads to a skewed evaluation function that favors one axis over another. We introduce functions that vary the weight of x and y (we only show the functions for x, but y is similar).

$$z_{2x} = 2x + y$$
$$z_{5x} = 5x + y$$
$$z_{10x} = 10x + y$$
$$z_{25x} = 25x + y$$
$$z_{50x} = 50x + y$$

Figure 8 shows the demonstrates that solutions tend to get worse as the scale increases (to the right). This trend is less pronounced (or absent) in the random and collinear functions (lin and sig) while very evident in the remaining functions. We do not show the plots for scaling y, but the trend is the same.

Summary and Future Work

We examine a synthetic domain that allows us to vary the interaction of two metrics, x and y, in a weighted objective function, z, while partially controlling for plan-length correlation. One of the more surprising findings is that A^*_ϵ search performs quite poorly when minimizing collinear functions ($y = x$ and $y = sigmoid(x)$), which suggests that researchers should avoid combining x and y when they are (nearly) collinear. A^*_ϵ works well for curvilinear functions such as polynomials and an "inverted" sigmoid. However, we discovered that scaling the metrics dramatically reduced search effectiveness. Poorer performance occurred when the metrics were uniformly scaled to control, at least in part, for plan-length correlation. Search performance also degraded as one metric was weighted more heavily.

There are limitations of the work that should be addressed in future work. We started with easier bi-criteria functions (linear and curvilinear) and plan to extend to more complex interactions and more metrics. We also need to control for

the discretization of x that led to better results than y. Uniformly selecting x should solve this problem. We identified at least some evidence for plan-length correlation and intend to explore this further. Our choice of MetricFF, A^*_ϵ and metric planning limits the findings to a single approach; we need to generalize our results to preference and cost-based planners as well as other metric planners. It is unclear how much the weight of A^*_ϵ or the heuristic accuracy could impact the search results, which needs to be addressed with a deeper study of the parameters for A^*_ϵ. It may also be fruitful to compare the plans found with worst cost plans rather than the optimal cost.

Ultimately we are interested in combining this work with search for diverse alternatives plans as in the work of (Nguyen et al. 2012; Coman and Munoz-Avila 2011; Roberts et al. 2012; Khouadjia et al. 2013; Radzi 2011; Sroka and Long 2012). In addition to examining non-temporal metrics, we want to focus on generating multiple alternative plans rather than a single plan. We intend to extend our set of domains to those found in Radzi (2011) and Sroka (2012) . We also hope to extend the findings of this paper to the security domain that motivates our initial interest in this topic.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. 0905232. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. We also thank the reviewers of this paper for their insightful comments that helped improve the paper.

References

Ammann, P.; Wijesekera, D.; and Kaushik, S. 2002. Scalable, graph-based network vulnerability analysis. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, 217–224.

Coles, A. J.; Coles, A. I.; Fox, M.; and Long, D. 2012. Colin: Planning with continuous linear numeric change. *Journal of Artificial Intelligence Research* 44:1–96.

Coman, A., and Munoz-Avila, H. 2011. Generating diverse plans using quantitative and qualitative plan distance metrics. In *Proc. of the 25th AAAI Conference on Artificial Intelligence*, 946–951.

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

Do, M. B., and Kambhampati, S. 2003. SAPA: A multi-objective metric temporal planner. *Journal of Artificial Intelligence Research* 20:155–194.

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

Gerevini, A., and Long, D. 2005. Plan constraints and preferences in PDDL3. Technical Report RT 2005-08-47, Dept. of Electronics for Automation, University of Brescia, Italy.

Gerevini, A.; Saetti, A.; and Serina, I. 2006. An approach to temporal planning and scheduling in domains with predicatable exogenous events. *Journal of Artificial Intelligence Research* 25:187–231.

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

Khouadjia, M.-R.; Schoenauer, M.; Vidal, V.; Dréo, J.; and Savéant, P. 2013. Multi-objective AI planning: Evaluating DAEyahsp on a tunable benchmark. In *Proceedings of the 7th International Conference on Evolutionary Multi-Criterion Optimization (EMO-2013)*, LNCS. Sheffield, UK: Springer.

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

Nguyen, T.; Do, M.; Gerevini, A.; Serina, I.; Srivastava, B.; and Kambhampati, S. 2012. Generating diverse plans to handle unknown and partially known user preferences. *Artificial Intelligence* 190:1–31.

Pearl, J., and Kim, J. H. 1982. Studies in semi-admissible heuristics. *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 4(4):392–399.

Radzi, N. H. M. 2011. *Multi-objective planning using linear programming*. Ph.D. Dissertation, The University of Strathclyde.

Roberts, M.; Howe, A.; Ray, I.; Urbanska, M.; Byrne, Z. S.; and Weidert, J. M. 2011. Personalized vulnerability analysis through automated planning. In *Workshop on Security and Artificial Intelligence (SecArt-11) in Working Notes of IJCAI-11. Barcelona, Catalonia, Spain. July 16-22*.

Roberts, M.; Howe, A. E.; Ray, I.; and Urbanska, M. 2012. Using planning for a personalized security agent. In *Workshop on Problem Solving using Classical Planners in Working Notes of the 26th AAAI Conference on Artificial Intelligence*. AAAI Press.

Sroka, M., and Long, D. 2012. Exploring metric sensitivity of planners for generation of pareto frontiers. In *Starting AI Researchers (STAIRS 2012)*, volume 241 of *Frontiers in Artificial Intelligence and Applications*, 306–317.

Wilt, C., and Ruml, W. 2011. Cost-based heuristic search is sensitive to the ratio of operator costs. In *Fourth Annual Symposium on Combinatorial Search (SoCS)*.

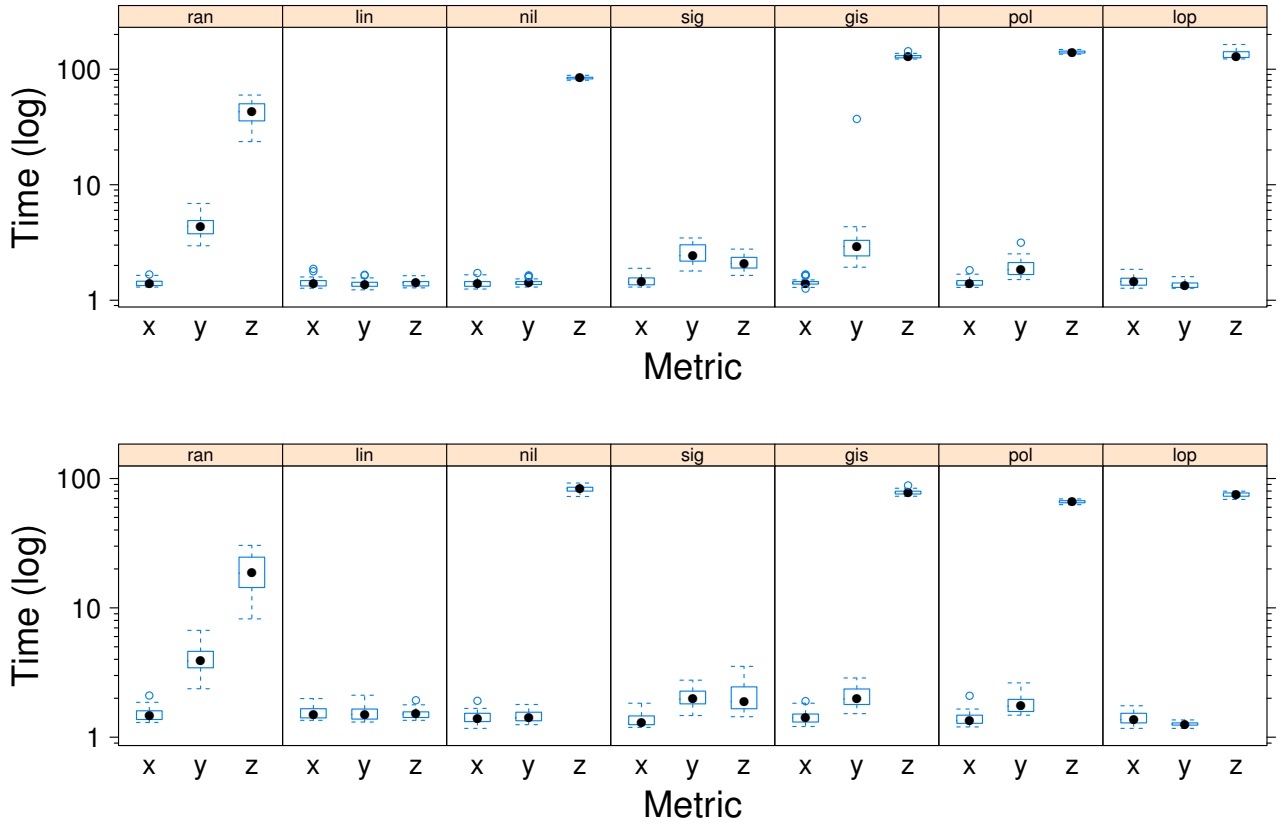


Figure 6: Runtime distributions of all functions and metrics for the original problems (top) and the under-correlated problems (bottom).

			x		y	
	Sig.	p-val	Avg. time	σ	Avg. time	σ
ran	***	0	1.41	0.101	4.48	0.958
lin		0.248	1.42	0.139	1.38	0.107
nil		0.669	1.41	0.109	1.42	0.088
sig	***	0	1.48	0.152	2.52	0.476
gis	*	0.034	1.43	0.104	3.99	6.28
pol	***	0	1.43	0.115	1.94	0.372
lop	***	0.001	1.47	0.148	1.36	0.083
ran	***	0	1.52	0.189	4.1	0.953
lin		0.984	1.54	0.169	1.54	0.203
nil		0.751	1.44	0.162	1.45	0.153
sig	***	0	1.37	0.174	2.03	0.297
gis	***	0	1.43	0.162	2.09	0.338
pol	***	0	1.4	0.185	1.82	0.332
lop	***	0	1.42	0.16	1.27	0.048

Table 2: Significance, p-value, and statistics (average time and standard deviation) for x and y on the original problems (top) and under-correlated problems (bottom).

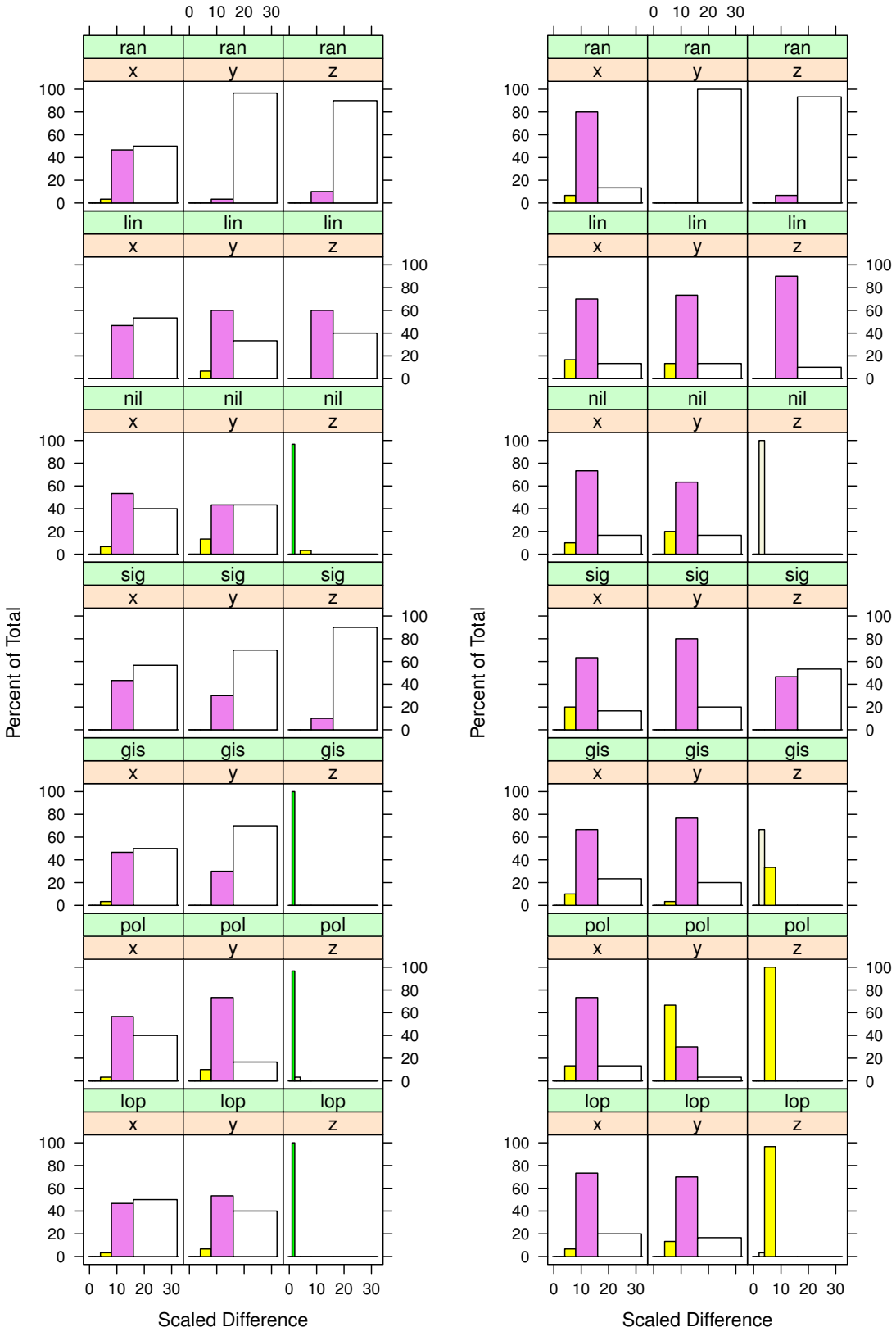


Figure 7: Log-Histograms of the scaled difference from the minimum for the original problems (left) and the under-correlated problems (right). Bins for the bottom axis are set at $\{0, 1, 2, 4, 8, 16, \geq 32\}$ to provide a visual representation of how well an algorithm does. Better performance is indicated by thinner and taller bars to the left. For more details, see the discussion of Figure 5.

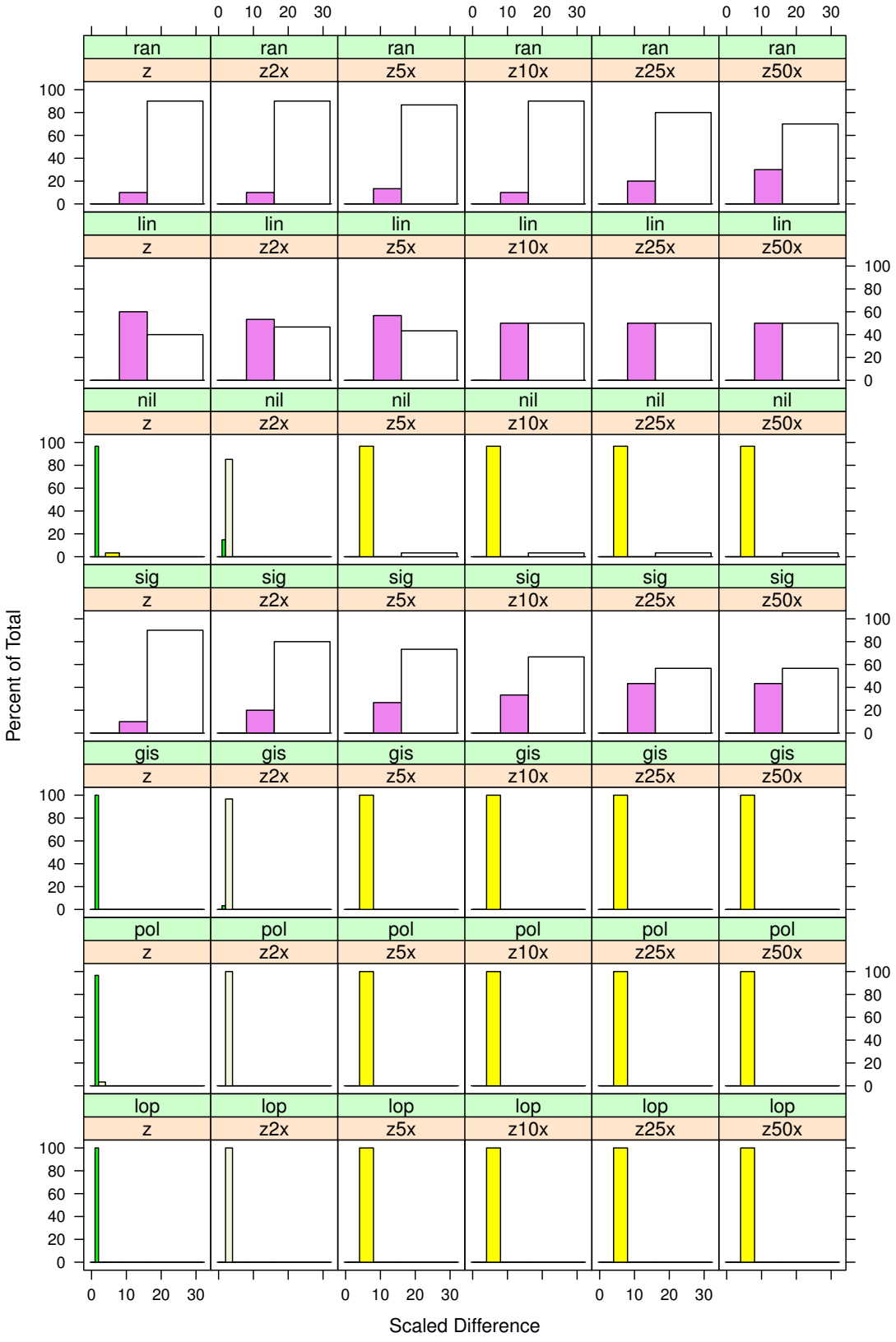


Figure 8: Log-Histograms of the scaled difference from the minimum for the z_{*x} problems. Bins for the bottom axis are set at $\{0, 1, 2, 4, 8, 16, \geq 32\}$ to provide a visual representation of how well an algorithm does. Better performance is indicated by thinner and taller bars to the left. For more details, see the discussion of Figure 5.