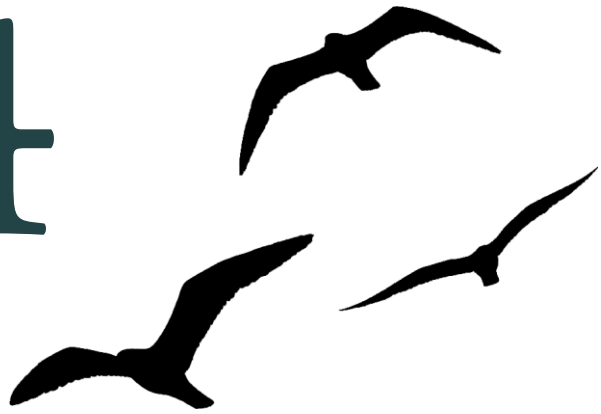
ICAPS 2014



Proceedings of the 6th Workshop on

Heuristics and Search for Domain Independent Planning

Edited By:

J. Benton, Patrik Haslum, Malte Helmert, Michael Katz and Jordan Thayer

Portsmouth, New Hampshire, USA - June 22, 2014

Organizing Committee

J. Benton
SIFT, USA
Patrik Haslum
NICTA/ANU, Australia
Malte Helmert
University of Basel, Switzerland
Michael Katz
IBM Research, Israel
Jordan Thayer
SIFT, USA

Program Committee

J. Benton, SIFT, USA
Patrik Haslum, NICTA/ANU, Australia
Malte Helmert, University of Basel, Switzerland
Michael Katz, IBM Research, Israel
Jordan Thayer, SIFT, USA

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 sixth 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, 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 for their hard work.

June 2014 J. Benton, Patrik Haslum, Malte Helmert, Michael Katz, and Jordan Thayer.

Table of Contents

Korf's Conjecture and the Future of Abstraction-based Heuristics

Robert C. Holte
Computing Science Department
University of Alberta
Edmonton, AB, Canada T6G 2E8
(holte@cs.ualberta.ca)

Abstract

In his 1997 paper on solving Rubik's Cube optimally using IDA* and pattern database heuristics (PDBs), Rich Korf conjectured that there was an inverse relationship between the size of a PDB and the amount of time required for IDA* to solve a problem instance on average. In the current paper, I examine the implications of this relationship, in particular how it limits the ability of abstraction-based heuristic methods, such as PDBs, to scale to larger problems. I discuss methods that might allow abstraction-based heuristics to scale better than Korf's Conjecture suggests and identify important auxiliary roles for abstraction-based heuristics in heuristic planning and search systems of the future that do not depend on their ability to scale well. Finally, I examine some key assumptions in the analysis underlying Korf's Conjecture, and identify two complications that arise in trying to apply it in practice.

Introduction

In 1997, Rich Korf published a paper (Korf 1997) in which random instances of Rubik's Cube were solved optimally for the first time using a general-purpose search algorithm (IDA*). This outstanding achievement was made possible by abstraction-based heuristics called pattern databases (PDBs), which had only recently been invented (Culberson and Schaeffer 1996). Korf's paper launched the golden age of abstraction-based methods for heuristic search, and a few years later Edelkamp (2001) introduced PDBs to the fledgling world of heuristic-search planning.

Korf's Rubik's Cube paper contained a second important but largely overlooked contribution that casts serious doubt on the long-term future of abstraction-based heuristics. Korf conjectured that there was an inverse relationship between m, the size of a PDB, and t, the number of node expansions that IDA*, using a PDB of size m, would perform to solve a problem instance on average, i.e. $m \cdot t = n$, where n is the size of the brute-force search tree for the original state space. Since node expansion is the primary operation in IDA*, t is indicative of IDA*'s execution time. Korf later gave a rigorous analysis, refining the conjecture to be $M \cdot t = n$, where $M = \frac{m}{1+log(m)}$ (Korf 2007).

In this paper I examine the implications of $M \cdot t = n$, in particular how it limits the ability of abstraction-based heuristic methods, such as PDBs, to scale to larger problems. Much of the discussion surrounds methods that might allow abstraction-based heuristics to scale better than Korf's Conjecture suggests. There are several promising possibilities, which I believe should be the focus of research in the next few years. I also identify important auxiliary roles for abstraction-based heuristics in heuristic planning and search systems of the future that do not depend on their ability to scale well. Finally, I examine some key assumptions in the analysis underlying Korf's Conjecture, and identify two complications that arise in trying to apply it in practice. This paper supercedes the version published last year (Holte 2013).

Background

Planning and heuristic search study the problem of finding a least-cost path (sequence of actions) from a given start state to a given goal state ($goal$).[1] The distance from state s to state t, denoted $d(s,t)$, is the cost of a least-cost path from s to t. An abstraction of a state space S is a mapping ϕ to another state space, S', such that $d(\phi(s),\phi(t)) \leq d(s,t)$ for all pairs of states $s,t \in S$. For any choice of S' and ϕ, $h(s) = d(\phi(s),\phi(goal))$ is an admissible, consistent heuristic for S. For example, consider the standard 3x3x3 Rubik's Cube, which consists of 8 corner cubies and 12 edge cubies. One way it can be abstracted is to consider all the corner cubies indistinguishable, e.g. to paint all the faces of every corner cubie black. A solution to this abstract problem is a sequence of actions that puts all the edge cubies in their goal positions without regard for what happens to the corner cubies. Distances in this space cannot be greater than the corresponding distances in the original Rubik's Cube space because the same operators exist in both spaces and solutions in the original space also must put the edge cubies in their goal positions. There are a variety of different families of abstraction functions that can be implemented as simple operations on standard state space representations, the most common of which are projection (Edelkamp 2001), and domain abstraction (Holte and Hernádvölgyi 1999).

I will use the term "pattern database" (PDB) to refer to any data structure that is indexed by individual abstract states and stores $d(s',\phi(goal))$ for each $s' \in S'$ from which the abstract goal can be reached. I will assume for the

[1] Korf's analysis assumes there is just one goal state.

present that the PDB is uncompressed, i.e. that the amount of memory needed for the PDB is linear in m, the number of abstract states in S' from which the abstract goal can be reached. Sometimes "PDB" is used in a narrower sense than this, but the reasoning underlying Korf's Conjecture applies to any data structure of this kind. A PDB is computed during a pre-processing phase by enumerating abstract states backwards from the abstract goal and recording the distance to each abstract state reached. During search, $h(s)$, the heuristic value for state $s \in S$, is computed by looking up the distance for $\phi(s)$ in the PDB.

Overview of Korf's Analysis

At the core of the analysis underlying Korf's Conjecture is the distribution of distances in the abstract space, which Korf calls the heuristic distribution since these distances are being used as heuristic values. Table 1 shows the distribution of distances in the abstraction of the 3x3x3 Rubik's Cube that ignores the corner cubies, as described above. In this space all moves cost 1, so the distance to the goal is the number of moves from the goal ("depth").

Given a heuristic distribution, Korf's Conjecture is derived in two steps. In the first step, the heuristic distribution is related to the size of the PDB. For this purpose, Korf assumes the number of abstract states at a given distance from the abstract goal grows exponentially as the distance increases, i.e. that there will be b^d abstract states at distance d for some branching factor b.[2] Korf recognizes that abstract spaces are usually graphs, not trees, so his analysis may underestimate the amount of pruning the heuristic will cause. The key to the analysis being a good approximation is that the exponential growth assumption be true of the majority of heuristic values, as it clearly is in Table 1 (see the "ratio" column).

The second step of the derivation relates the heuristic distribution to the number of nodes expanded by IDA*. In the formal analysis (Korf 2007) this step is done using the KRE formula (Korf, Reid, and Edelkamp 2001). The relationship $M \cdot t = n$ follows directly from some straightforward manipulation of the KRE formula with an exponential heuristic distribution.

Because this derivation is entirely formal, Korf's Conjecture is no longer a conjecture, except in the sense that the assumptions on which the analysis is based are being conjectured to hold in problems of interest. That is a topic I will return to later in the paper, but for the next few sections I will assume Korf's Conjecture has been proven and explore its implications.

[2]To simplify the analysis, Korf assumes that this b is the same as the branching factor in the search tree for the original space.

depth	#states	ratio
0	1	-
1	18	18.0
2	243	13.5
3	3,240	13.3
4	42,807	13.2
5	555,866	13.0
6	7,070,103	12.7
7	87,801,812	12.4
8	1,050,559,626	12.0
9	11,588,911,021	11.0
10	110,409,721,989	9.5
11	552,734,197,682	5.0
12	304,786,076,626	-
13	330,335,518	-
13	248	-

Table 1: Distance-to-goal ("depth") distribution for the 3x3x3 Rubik's Cube abstraction that ignores the corner cubies (Table 2 in (Korf 2008)). "ratio" is #states at depth d divided by #states at depth $d-1$ (not shown if less than 1).

The Problem of Scaling

The fact that the amount of memory needed to store a PDB is linear in m, the number of abstract states for which the PDB stores information, limits how useful abstraction-based heuristics can be in solving combinatorial problems. n grows exponentially as the size of a combinatorial problem increases (e.g. add 1 more block to the blocks world, 1 more disk to the Towers of Hanoi, 1 more row or column full of tiles to the sliding-tile puzzle). If Korf's Conjecture is true, then $M \cdot t$ must also grow exponentially. If we have a fixed amount of memory, t would have to grow exponentially, and, if we have an upper bound on how much time we are willing to wait for a solution, then m must grow exponentially. This represents a fundamental limitation on the ability of abstraction-based heuristics to guide search effectively.

If we allow m and t to both increase as n increases, it is somewhat encouraging to see that they can increase with the square root of n ($M{=}\sqrt{n} \Rightarrow t{=}\sqrt{n}$). If the time to construct the PDB is linear in M, then the total time to solve a single problem instance will also grow with the square root of n, instead of being linear in n as brute-force search would be. Nevertheless, if n grows exponentially as we increase the size of our combinatorial problems, t and m will both also grow exponentially.

Possible Solutions

In this section I review existing technologies that might provide a solution to the scaling problem.

Disk-based and Distributed PDBs. Storing PDBs on disk instead of in RAM (Sturtevant and Rutherford 2013; Zhou and Hansen 2005) or distributing them across a cluster of workstations that each have their own RAM (Edelkamp, Jabbar, and Kissmann 2008) allows m to be two or three orders of magnitude larger than it could be if the PDB had to fit in the RAM of one workstation. This is extremely useful, but, in the end, is just a big constant factor, not a solution to the scaling problem.

PDB Compression. Lossless compression of PDBs, such as symbolic PDBs (Edelkamp 2001), routinely reduces the memory needed to store a PDB by one to two orders of magnitude (Ball and Holte 2008). In certain special cases symbolic PDBs have been proven to be logarithmic in the uncompressed size of the PDB (Edelkamp and Kissmann 2011). The scaling problem is solved by symbolic PDBs

in these spaces, but not in general. Large memory reductions have also been obtained with lossy PDB compression methods (Felner et al. 2007; Samadi et al. 2008), which offer a little more hope for addressing the scaling issue because it seems that by allowing some loss of heuristic information, these methods sometimes produce a more favourable tradeoff between time and memory than is predicted by Korf's Conjecture. For example, Felner et al. (2007)'s Table 9 reports that a 9 times reduction in memory results in only 2.25 greater search time, and an even more favourable tradeoff is reported in their Table 4 for the Towers of Hanoi.

Hierarchical Heuristic Search. Instead of precomputing, and storing, the entire PDB, hierarchical heuristic search (Holte, Grajkowski, and Tanner 2005; Leighton, Ruml, and Holte 2011) computes, on demand, precisely those abstract distances that are required as heuristic values in solving a given problem instance. Experimentally, only about 1% of the PDB entries actually need to be computed to solve a problem instance (see Table 3 in (Holte, Grajkowski, and Tanner 2005)), which means about one order of magnitude less memory is required than for the full PDB since the data structure for caching distance-to-goal information in hierarchical heuristic search is not as compact as a good PDB data structure. What is not known is how the memory requirements of hierarchical heuristic search scale as the state space size increases. It is possible that hierarchical heuristic search scales better than PDBs and therefore provides at least a partial solution to the scaling problem.

Multiple PDBs. One direction that offers clear hope for mitigating the scaling problem is the use of multiple PDBs. Korf's Conjecture, as I have described it here, is about how search time is related to the size of a PDB when just one PDB is used to guide IDA*. But it is known that, for a fixed amount of memory, search using one PDB that uses all the memory is much slower than search that takes the maximum of two PDBs that each require half the memory (Holte et al. 2004; 2006). Could it be that as a search space scales up, the total size (m) of a set of PDBs does not have to grow exponentially in order to keep t constant?

In my opinion, the answer is almost certainly "no" when the maximum is taken over the set of PDBs. Korf (2007) analyzes this case when the PDBs are "independent". According to the formula he derives with this assumption, the optimal number of PDBs is two, which is not consistent with experimental data. Korf offers two reasons for this discrepancy. One is that the heuristic distribution is not necessarily exponential. I return to this point below. The other is that the derived formula overestimates the pruning power of a set of PDBs if they are not truly independent. I suspect that non-additive PDBs will rarely be independent, or even approximately independent, and so their total size will have to scale almost as quickly as a single PDB.

On the other hand, a set of PDBs based on additive abstractions (Yang et al. 2008) might well give us the scaling behaviour we want. Breyer and Korf (2010) proved that the speedup produced using a set of additive heuristics is the product of their individual speedups over brute-force search if the heuristics are independent in the same sense as above. This seems to me a much more plausible assumption for additive abstractions than non-additive ones. Breyer and Korf's analysis suggests that as a state space is scaled up, t could be kept constant by increasing the number of PDBs, which is only a linear increase in memory. A more powerful version of this idea, which I call "factored heuristics", is discussed below.

Multiple PDB Lookups. Another source of hope related to the use of multiple PDBs is the use of multiple lookups in a single PDB: to compute the heuristic value of state s one not only does the normal PDB lookup, but has a mechanism for extracting one or more additional values from the PDB that are also lower bounds on s's distance to goal. This obtains the benefits of having multiple PDBs while using the memory needed by only one PDB. Most studies that make multiple lookups in a PDB use symmetries in the search space to define the additional lookups (Zahavi et al. 2008).

A particularly powerful form of symmetry is seen in the Towers of Hanoi. The standard method of abstracting this space is to choose k disks and build a PDB containing the number of moves needed to get those k disks to their goal positions, entirely ignoring all the other disks (Felner et al. 2007). If the Towers of Hanoi problem one is trying to solve has $2k$ disks, two lookups can be made in this PDB and their values added: one lookup is based on any set of k disks, the other lookup is based on the other k disks. If the problem is made bigger by increasing the number of disks, the PDB remains the same, but more lookups are done and added together to compute a state's heuristic value. I call this a "factored heuristic" because the heuristic calculation is done by decomposing the state into parts (in this example, partitioning the set of disks into groups that each contain k or fewer disks), making separate lookups for each part in the same PDB, and then adding the results.

The same idea has been used for the Pancake puzzle (Torralba Arias de Reyna and Linares López 2011) and produced extremely good (sub-exponential) scaling behaviour. Doubling the number of pancakes from 20 to 40 increased the number of nodes generated by a factor of 120, a miniscule fraction of the increase in the size of the state space (from 20! to 40!).

Reliance on symmetries in the state space has limited the use of multiple PDB lookups. However, Pang and Holte (2012) report a technique that allows multiple PDB lookups to be based on multiple abstractions that all map to the same abstract space, thereby allowing multiple lookups to be done for state spaces that do not have symmetries.

Alternative Ways of Representing Abstraction-based Heuristics. It may be possible to avoid the scaling problem by representing abstraction-based heuristics in a form that is entirely different than a lookup table. Manhattan Distance, for example, could be implemented as an additive PDB, but is more commonly implemented as a procedure that is executed for a given state. In fact, this procedure strongly resembles hierarchical heuristic search, but with individual abstract spaces that are so small there is no need for a hierarchy of abstractions or for search results to be cached.

An alternative representation of abstraction-based heuristics that is especially useful if admissibility is not required is to use machine learning to create an extremely compact approximation of a PDB (e.g. a small neural network), as was done by Samadi et al. (2008). If admissibility is required, a lookup table can be used to store the PDB entries that the neural network overestimates. In the experiments by Samadi et al., only about 2% of the PDB entries needed to be explicitly stored.

A different way of compactly approximating an abstraction-based heuristic is to map the abstract states into a low-dimensional Euclidean space in such a way that Euclidean distances between states are admissible and consistent. The basic technology for this exists (Rayner, Bowling, and Sturtevant 2011), but at present it requires the same amount of memory as a PDB.

Alternative Roles for Abstraction-based Heuristics

In this section I consider what role there might be for abstraction-based heuristics if, indeed, they are unable to scale to provide effective guidance for solving much larger problems than we currently study. Assuming they do not scale, we need to consider roles in which weak heuristics can make important contributions to solving search problems.

Node Ranking. Many search algorithms, including depth-first branch and bound, beam search, greedy best-first search (Doran and Michie 1966), and limited discrepancy search (Harvey and Ginsberg 1995), sort the nodes they generate according to what might be called a ranking function. Although there is no need for the ranking function to be an estimate of the cost to reach the goal (e.g. (Xu, Fern, and Yoon 2009)), a heuristic function is an obvious candidate to use for ranking. Even a relatively weak heuristic can be effective for ranking. For example, BULB (Furcy and Koenig 2005), which combines beam search and limited discrepancy backtracking, solves instances of the 9×9 sliding-tile puzzle in 120 seconds using Manhattan Distance for ranking, which is not an especially effective heuristic for that size of puzzle.[3]

Type Systems. Levi Lelis and his colleagues have developed methods for a variety of search-related tasks that require the nodes in a search tree to be partitioned into a set of "types" (Lelis et al. 2012; Lelis, Zilles, and Holte 2013a; 2013b; Lelis, Otten, and Dechter 2013; Xie et al. 2014). In the ideal partitioning, the search subtrees below nodes of the same type are identical in certain key regards, such as the size of the subtree, the cost of the cheapest solution in the subtree, etc. Type systems based on heuristic functions have proven very effective in all these studies, and, as with node ranking, there is no need for the heuristic to be especially accurate (many of Lelis's heuristics are small PDBs).

Features for Learning. There has been recent interest in using machine learning to create heuristic functions (Samadi, Felner, and Schaeffer 2008; Jabbari Arfaee, Zilles, and Holte 2011). In these studies small PDBs have proven to be excellent features for machine learning. Again, there is no obvious need for heuristics used for this purpose to be especially accurate.

Heuristic Search as a Component Technology. One approach to solving a very large problem is to decompose it into a sequence of subproblems that are then solved one by one, in order, to produce a solution to the original problem. Korf developed this technique for solving Rubik's Cube, with the subproblems being solved by brute-force search (Korf 1985). Later, Hernádvölgyi (2001) realized that abstraction-based heuristics could be used to guide the solving of the subproblems. This speeded up search so much that it allowed several subproblems to be merged to form a single subproblem that was still feasible to solve, resulting in much shorter solutions (50.3 moves, on average, compared to 82.7). This is just one example of how abstraction-based heuristic search can be used as a component in a different type of search system.

The Assumptions Underlying Korf's Analysis

The previous sections have outlined directions for the future of abstraction-based heuristics assuming that Korf's Conjecture holds. In this section, I examine the assumptions underlying Korf's analysis, and describe two complications that arise in trying to apply it in practice. The two points I will make in this section will both be illustrated using the 5-disk, 3-peg Towers of Hanoi puzzle, with states being represented the same way that Zilles and Holte (2010) represented states in the Blocks World with distinct table positions. In this representation there are 6 state variables for each peg. The first variable is an integer $(0 \ldots 5)$ saying how many disks are on the peg. The other 5 variables give the names of the disks ($d_1 \ldots d_5$ or $nodisk$) in the order they occur on the peg from bottom to top: the first of these variables names the disk at the bottom ($nodisk$ if the peg has no disks on it), the second names the disk second from the bottom ($nodisk$ if the peg has 0 or 1 disks on it), and so on. The abstraction of this space that I will discuss is shown in Figure 1. This abstraction keeps the variables that say how many disks are on each peg and deletes all the other variables, so that the identities of the disks on each peg are entirely lost. For example, the Towers of Hanoi goal state, which has all the disks on peg_1, would be mapped to abstract state **500** (at the top of the figure), and any Towers of Hanoi state in which there are 3 disks on peg_1, 2 disks on peg_2, and no disks on peg_3 would be mapped to abstract state **320**.

The first thing one notices about this space is that the distance distribution is not exponential, it is linear; there are exactly $d + 1$ abstract states at distance d from the abstract goal. It would seem therefore that Korf's Conjecture does not apply in this case. However, Korf's analysis could be

[3] In 2001 it was estimated that IDA* would need 50,000 years of CPU time to solve one instance of the 5×5 sliding-tile puzzle using Manhattan Distance (Korf, Reid, and Edelkamp 2001). Korf has re-calculated this based on the speed of today's computers and estimates that only 7,600 years would be required now (personal communication).

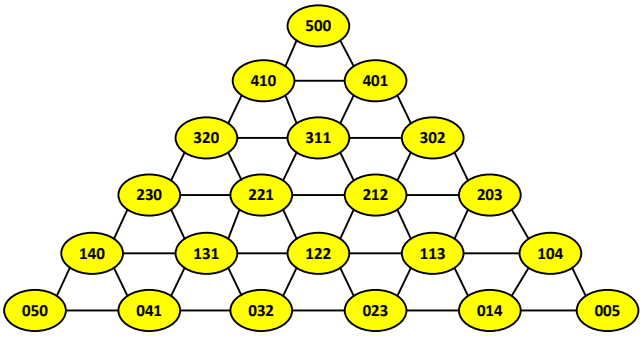


Figure 1: Abstraction of the 5-disk, 3-peg Towers of Hanoi state space that keeps track of how many disks are on each peg but not their identities. **320**, for example, is the abstract state in which there are 3 disks on peg_1, 2 disks on peg_2, and no disks on peg_3.

repeated with a linear model plugged into the KRE formula to derive the relation between t and m for this situation. I have not actually done that, but using the simpler, "back of the envelope" method of Korf's original, intuitive analysis, I estimate that, with a heuristic defined by an abstract space with a linear distance distribution, if n is doubled, m does not need to be doubled to keep t constant, it only needs to be increased additively by approximately $2\sqrt{m}$.[4]

In the preceding paragraph I said "it would seem" Korf's Conjecture does not apply because it is not correct, in general, to equate the distance distribution in the abstract space with the heuristic distribution. In the KRE formula the heuristic distribution is the fraction of states in the original state space that have a particular heuristic value,[5] not the fraction in the abstract space. The distance distribution in the abstract space and the heuristic distribution are only the same if the same number of states in the original space map to each abstract state. That is certainly not the case for the Towers of Hanoi abstraction in Figure 1. Only one Towers of Hanoi state maps to abstract state **500**, but 10 map to abstract state **320**, and 30 map to abstract state **221**. The number of Towers of Hanoi states that map to each abstract distance are 1 (distance 0), 10 (distance 1), 40, 80, 80, and 32 (distance 5), which is probably close enough to being an exponential distribution that Korf's Conjecture would hold in this case. This is an important consideration because it oftens happens naturally that different numbers of states that map to different abstract states, as in this example, and, in addition, there are several abstraction methods that are very likely to produce abstractions in which this occurs (e.g. CEGAR (Seipp and Helmert 2013), CFDP (Raphael 2001), and merge-and-shrink (Nissim, Hoffmann, and Helmert 2011)).

An important, but easily overlooked, feature of the numbers (1, 10, 40, etc.) just given is that they refer to the number of **reachable** states that map to each abstract state. If we consider unreachable states as well, then the same number of states map to each abstract state and we are back to a linear heuristic distribution. So, Korf's Conjecture rests on the heuristic distribution of reachable states being distributed exponentially. That is the first point about the conjecture I wish to make. This does not make the analysis any less valid, it just means it is more difficult to apply than doing a simple inspection of the abstract space, and that similar problems might have very different behaviours: removing the Towers of Hanoi restriction that a larger disk cannot be put onto a smaller one changes the heuristic distribution from being (approximately) exponential to being linear.

The other point I wish to make is that although the parameter m in the analysis seems perfectly well defined, it is, in fact, anything but. The most obvious source of doubt about how exactly m should be defined is spurious abstract states, which are abstract states that can be reached by backwards search from the abstract goal but whose pre-image[6] cannot be reached by forward search from the start state in the original space.[7] Because they are reached by backwards search from the abstract goal, there is an entry in the PDB for each spurious state, but because their pre-images cannot be reached during forward search from the start state in the original state space, they contribute nothing to the pruning power of the heuristic and therefore should not be included in the heuristic distribution or the value of m used in Korf's analysis (if m is increased by adding spurious states, t will obviously not change at all). For example, Zilles and Holte (2010) report an abstraction of the Blocks World in which there are $1,310,720$ abstract states from which the abstract goal can be reached, of which only $89,400$ are non-spurious. In Korf's analysis, $m = 89,400$ must be used, not $m = 1,310,720$. Unfortunately, there is no practical way, in general, of determining how many abstract states are spurious, or even deciding whether or not there are any spurious states (Zilles and Holte 2010).

The difficulties concerning m's definition raised by spurious states is a special case of a more general phenomenon where two abstract spaces of different sizes give rise to exactly the same heuristic values for all states in the original space. The space in Figure 1 has 21 states. Once **500** has been designated as the goal, the space can be compressed to just 6 states without losing any distance-to-goal information. This is done by mapping all the states on the same level in the figure to the leftmost state on the level. This is an ordinary abstraction of the space in Figure 1 defined by keeping the first state variable and deleting the other two. We now have a real dilemma in deciding what value of m to

[4] This calculation assumes that $t = 2^{D-\overline{h}}$, where D is the depth of the search in the original space, 2 is the branching factor in both the original space and the abstract space, and $\overline{h}$ is the average heuristic value, which, for a space with a linear distance distribution, is approximately $\sqrt{m}$.

[5] Even this is not a perfectly correct statement. The distribution described here is what Korf calls the overall distribution. The KRE analysis requires the "equilibrium" distribution, which in general is not the same as the overall distribution.

[6] If S is a state space and ϕ is an abstraction mapping S to S', the pre-image of $s' \in S'$ is $\{s \in S | \phi(s) = s'\}$.

[7] Spurious states can be caused either by the backwards (regression) search (Bonet and Geffner 2001) that computes the PDB or by the abstraction process itself (Zilles and Holte 2010).

use. There are no spurious states in Figure 1, so $m = 21$ is genuinely the size of the space, but the heuristic based on it is identical to a heuristic based on a linear space of size 6. If n is doubled and we want t to remain constant, doubling "m" means to make a space equivalent to a linear space of size 12, not a triangular space of size 42.[8] For the purpose of defining a heuristic, additional states at each level are as useless as spurious states: increasing m by adding more of them does not affect t at all.

Methods for creating symbolic PDBs (Edelkamp 2001) can be seen as doing exactly what I have just illustrated with the space in Figure 1: they take the distances-to-goal defined in one abstract space (the space in Figure 1, for example), and construct a smaller abstract space that returns exactly the same heuristic value for every state in the original space (the linear space with just 6 nodes, for example). This is also what merge-and-shrink aspires to do with its shrinking stategies that are h-preserving (Helmert, Haslum, and Hoffmann 2007) or based on bisimulation (Nissim, Hoffmann, and Helmert 2011). In fact, these methods can be seen as always producing a linear abstract space with one state at each distance, coupled with a memory-based indexing mechanism for mapping a given state to one of these abstract states.

The fact that abstract spaces of very different sizes can produce identical heuristics suggests that Korf's goal of relating the number of node expansions by IDA* to the number of states in an abstract space is simply impossible. And yet there is experimental data showing that a strong relation does indeed hold between PDB size and the number of node expansions, exactly as predicted by Korf's Conjecture. The data in Table 2 is based on the results reported in Korf (2007)'s Table 2, columns "Size" (m) and "IDA*" (t). One iteration of IDA* with a depth bound of 12 was run on the same 1000 Rubik's Cube instances using four PDBs of different sizes based on abstractions called "6 corners", "6 edges", "8 corners", and "7 edges". For a given PDB, t is the average number of nodes expanded by IDA* on these runs. The asymptotic branching factor of Rubik's Cube, 13.34847 (Korf 1997), was used as the base of the logarithm in computing M from m. If Korf's Conjecture was perfectly correct, the numbers in the $M \cdot t$ column would be identical. They are nearly so: the largest value is only about 13% larger than the smallest value. This shows that, in this particular experiment, Korf's Conjecture makes very accurate predictions about how the number of node expansions will change if the PDB size is changed. Additional support for Korf's Conjecture is given in an experiment that looked at a large number of PDBs of many different sizes for three state spaces (Holte and Hernádvölgyi 1999), although this study uses A*, not IDA*.

[8]The two are not the same; a triangular space would need 78 states to correspond to a linear space of length 12.

PDB	m	t	$M \cdot t$
6 corners	14,696,640	20,918,500	41,722 $\times 10^9$
6 edges	42,577,920	8,079,408	44,222 $\times 10^9$
8 corners	88,179,840	3,724,861	40,752 $\times 10^9$
7 edges	510,935,040	670,231	39,191 $\times 10^9$

Table 2: Evidence supporting Korf's Conjecture. $M \cdot t$ is almost constant for four PDBs of different sizes for Rubik's Cube. m and t are from Table 2 in (Korf 2007).

Conclusions

In this paper, I have examined how Korf's Conjecture ($M \cdot t = n$), if it is true, limits the ability of abstraction-based heuristic methods, such as PDBs, to scale to larger problems. It is certain that abstraction-based heuristics can and should play important auxiliary roles in the heuristic planning and search systems of the future, whether or not they scale well. If they do not scale well, there are several alternative types of heuristics in the planning literature—delete relaxations (McDermott 1996; Bonet, Loerincs, and Geffner 1997), heuristics based on linear programming (Bonet 2013), the causal graph heuristic (Helmert 2004), landmark-based heuristics (Bonet and Helmert 2010), and h^m (Haslum 2006)—whose scaling behaviour has only partly been studied (Helmert and Mattmüller 2008).

However, there are several reasons to expect that abstraction-based heuristics may continue to play a primary role in planning and search systems. In part, this hope lies in the use of merge-and-shrink, multiple additive abstractions, and multiple heuristic lookups in a single abstract space. These technologies exist, and might possibly greatly mitigate the scaling problem, but more research is needed on them. The second place where hope lies is in the scaling behaviour of compression methods and hierarchical heuristic search, which I assumed to be linear in m in this paper. If any of these were to scale logarithmically, say, instead of linearly, abstraction-based heuristic methods would continue to play a central role in solving large combinatorial problems.

This paper has also shown that it is not straightforward to apply Korf's Conjecture in practice. One reason is that the poor scaling behaviour rests on an assumption about the heuristic distribution of reachable states, not just on the distribution of distances in the abstract space. The other reason is that the number of entries in a PDB is definitely not the sole determining factor of the pruning power of the resulting heuristic: PDBs of very different sizes can give rise to exactly the same heuristic. More research is needed to better understand, and exploit, this phenomenon.

Acknowledgements

Thanks to Malte Helmert, Rich Korf, Carlos Linares López, Rong Zhou, Roni Stern, and Ariel Felner for their feedback, and to Canada's NSERC and Switzerland's SNF for their financial support.

References

Ball, M., and Holte, R. C. 2008. The compression power of symbolic pattern databases. In *ICAPS*, 2–11.

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

Bonet, B., and Helmert, M. 2010. Strengthening landmark heuristics via hitting sets. In *ECAI*, 329–334.

Bonet, B.; Loerincs, G.; and Geffner, H. 1997. A robust and fast action selection mechanism for planning. In *AAAI*, 714–719.

Bonet, B. 2013. An admissible heuristic for SAS+ planning obtained from the state equation. In *IJCAI*.

Breyer, T. M., and Korf, R. E. 2010. Independent additive heuristics reduce search multiplicatively. In *AAAI*, 33–38.

Culberson, J., and Schaeffer, J. 1996. Searching with pattern databases. In *Proc. 11th Biennial Conference of the Canadian Society for Computational Studies of Intelligence*, volume 1081 of *Lecture Notes in Computer Science*, 402–416. Springer.

Doran, J. E., and Michie, D. 1966. Experiments with the Graph Traverser program. In *Proceedings of the Royal Society*, 235–259.

Edelkamp, S., and Kissmann, P. 2011. On the complexity of BDDs for state space search: A case study in Connect Four. In *AAAI*, 18–23.

Edelkamp, S.; Jabbar, S.; and Kissmann, P. 2008. Scaling search with pattern databases. In *MoChArt*, 49–64.

Edelkamp, S. 2001. Planning with pattern databases. In *Proc. European Conference on Planning*, 13–24.

Felner, A.; Korf, R. E.; Meshulam, R.; and Holte, R. C. 2007. Compressed pattern databases. *Journal of Artificial Intelligence Research* 30:213–247.

Furcy, D., and Koenig, S. 2005. Limited discrepancy beam search. In *IJCAI*, 125–131.

Harvey, W. D., and Ginsberg, M. L. 1995. Limited discrepancy search. In *IJCAI*, 607–615.

Haslum, P. 2006. Improving heuristics through relaxed search - an analysis of TP4 and HSP*a in the 2004 planning competition. *Journal of Artificial Intelligence Research (JAIR)* 25:233–267.

Helmert, M., and Mattmüller, R. 2008. Accuracy of admissible heuristic functions in selected planning domains. In *AAAI*, 938–943.

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

Helmert, M. 2004. A planning heuristic based on causal graph analysis. In *ICAPS*, 161–170.

Hernádvölgyi, I., and Holte, R. C. 2000. Experiments with automatically created memory-based heuristics. In *Symposium on Abstraction, Reformulation and Approximation (SARA)*, volume 1864 of *Lecture Notes in Artificial Intelligence*, 281–290. Springer.

Hernádvölgyi, I. T. 2001. Searching for macro operators with automatically generated heuristics. In *Canadian Conference on AI*, 194–203.

Holte, R. C., and Hernádvölgyi, I. T. 1999. A space-time tradeoff for memory-based heuristics. In *AAAI*, 704–709.

Holte, R. C.; Newton, J.; Felner, A.; Meshulam, R.; and Furcy, D. 2004. Multiple pattern databases. In *ICAPS*, 122–131.

Holte, R. C.; 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.

Holte, R. C.; Grajkowski, J.; and Tanner, B. 2005. Hierarchical heuristic search revisited. In *Symposium on Abstraction, Reformulation and Approximation (SARA)*, 121–133.

Holte, R. C. 2013. Korf's conjecture and the future of abstraction-based heuristics. In *Symposium on Abstraction, Reformulation and Approximation (SARA)*.

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

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

Korf, R. E. 1985. Macro-operators: A weak method for learning. *Artificial Intelligence* 26(1):35–77.

Korf, R. 1997. Finding optimal solutions to Rubik's Cube using pattern databases. In *AAAI*, 700–705.

Korf, R. E. 2007. Analyzing the performance of pattern database heuristics. In *AAAI*, 1164–1170.

Korf, R. E. 2008. Minimizing disk i/o in two-bit breadth-first search. In *AAAI*, 317–324.

Leighton, M. J.; Ruml, W.; and Holte, R. C. 2011. Faster optimal and suboptimal hierarchical search. In *Symposium on Combinatorial Search (SoCS)*.

Lelis, L.; Stern, R.; Felner, A.; Zilles, S.; and Holte, R. C. 2012. Predicting optimal solution cost with bidirectional stratified sampling. In *ICAPS*.

Lelis, L. H. S.; Otten, L.; and Dechter, R. 2013. Predicting the size of depth-first branch and bound search trees. In *IJCAI*.

Lelis, L.; Zilles, S.; and Holte, R. C. 2013a. Stratified tree search: A novel suboptimal heuristic search algorithm. In *Twelfth International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 555–562.

Lelis, L. H. S.; Zilles, S.; and Holte, R. C. 2013b. Predicting the size of IDA*'s search tree. *Artificial Intelligence* 196:53–76.

McDermott, D. V. 1996. A heuristic estimator for means-ends analysis in planning. In *Proceedings of the Third International Conference on Artificial Intelligence Planning Systems (AIPS)*, 142–149.

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 *IJCAI*, 1983–1990.

Pang, B., and Holte, R. C. 2012. Multimapping abstractions and hierarchical heuristic search. In *Symposium on Combinatorial Search (SoCS)*.

Raphael, C. 2001. Coarse-to-fine dynamic programming. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 23(12):1379–1390.

Rayner, D. C.; Bowling, M. H.; and Sturtevant, N. R. 2011. Euclidean heuristic optimization. In *AAAI*, 81–86.

Samadi, M.; Siabani, M.; Felner, A.; and Holte, R. 2008. Compressing pattern databases with learning. In *ECAI*, 495–499.

Samadi, M.; Felner, A.; and Schaeffer, J. 2008. Learning from multiple heuristics. In *AAAI*, 357–362.

Seipp, J., and Helmert, M. 2013. Counterexample-guided Cartesian abstraction refinement. In *ICAPS*.

Sturtevant, N. R., and Rutherford, M. J. 2013. Minimizing writes in parallel external memory search. In *IJCAI*.

Torralba Arias de Reyna, Á., and Linares López, C. 2011. Size-independent additive pattern databases for the pancake problem. In *Symposium on Combinatorial Search (SoCS)*, 164–171.

Xie, F.; Müller, M.; Holte, R.; and Imai, T. 2014. Type-based exploration with multiple search queues for satisficing planning. In *AAAI*.

Xu, Y.; Fern, A.; and Yoon, S. W. 2009. Learning linear ranking functions for beam search with application to planning. *Journal of Machine Learning Research* 10:1571–1610.

Yang, F.; Culberson, J.; Holte, R.; Zahavi, U.; and Felner, A. 2008. A general theory of additive state space abstractions. *Journal of Artificial Intelligence Research* 32:631–662.

Zahavi, U.; Felner, A.; Holte, R. C.; and Schaeffer, J. 2008. Duality in permutation state spaces and the dual search algorithm. *Artificial Intelligence* 172(4-5):514–540.

Zhou, R., and Hansen, E. A. 2005. External-memory pattern databases using structured duplicate detection. In *AAAI*, 1398–1405.

Zilles, S., and Holte, R. C. 2010. The computational complexity of avoiding spurious states in state space abstraction. *Artificial Intelligence* 174:1072–1092.

"Distance"? Who Cares? Tailoring Merge-and-Shrink Heuristics to Detect Unsolvability

Jörg Hoffmann and **Peter Kissmann** and **Álvaro Torralba**
Saarland University, Saarbrücken, Germany
{hoffmann,kissmann,torralba}@cs.uni-saarland.de

Abstract

Research on heuristic functions is all about estimating the length (or cost) of solution paths. But what if there is no such path? Many known heuristics have the ability to detect (some) unsolvable states, but that ability has always been treated as a by-product. No attempt has been made to design heuristics specifically for that purpose, where there is no need to preserve distances. As a case study towards leveraging that advantage, we investigate merge-and-shrink abstractions in classical planning. We identify *safe abstraction* steps (no information loss regarding solvability) that would not be safe for traditional heuristics. We design practical algorithm configurations, and run extensive experiments showing that our heuristics outperform the state of the art for proving planning tasks unsolvable.

Introduction

Research on heuristic functions is all about estimating the length (or cost) of solution paths. There even is a perception that, on unsolvable problems, state ordering does not matter so computing a heuristic is a waste of time. That is false for heuristics with the ability to detect (some) dead-end states, like almost all known heuristics in planning. This is not in itself a new observation, but it has never been systematically explored. Unsolvability detection has always been treated as a by-product of estimating goal distance/cost. For example, all relaxed-plan based heuristics (e. g. (Hoffmann and Nebel 2001)), all landmark heuristics (e. g. (Richter and Westphal 2010)), and the recent red-black plan heuristics (Katz and Hoffmann 2013), are no better at unsolvability detection than the "Methuselah heuristic" $h^{\max}$. We introduce *unsolvability heuristics*, returning either ∞ or 0, as an alternative research focus aiming to address the questions: *How to design heuristics specifically for unsolvability detection? Can we leverage the lack of need to preserve distances? Is search with such heuristics competitive with other approaches for proving unsolvability?*

These are long-term research challenges, that are relevant due to (a) the practical importance of unsolvable problems (e. g., directed model checking (Edelkamp, Lluch-Lafuente, and Leue 2004) and over-subscription planning (Gerevini et al. 2009)), and (b) the practical importance of detecting dead-ends in solvable problems (e. g., when dealing with limited resources (Nakhost, Hoffmann, and Müller 2012; Coles et al. 2013)).

We investigate *merge-and-shrink abstractions* (Helmert, Haslum, and Hoffmann 2007) as a case study. M&S abstractions iteratively *merge* all state variables (build the cross-product of these variable's transition systems), and *shrink* the intermediate outcomes to keep abstraction size at bay. A key issue is how to shrink without losing too much information. We identify *safe abstraction* steps, that do not incur any information loss regarding solvability (but that do lose information regarding goal distance so would not be safe for traditional heuristics). Leveraging prior work on K*-catching bisimulation* (Katz, Hoffmann, and Helmert 2012), where the behavior of a subset of actions K is reflected exactly in the M&S abstraction, we identify sets K rendering this kind of abstraction safe. Approximating such K yields practical heuristics. We collect a suite of unsolvable benchmarks, and run comprehensive experiments. Competing approaches, including BDDs, are outperformed drastically; the advantage over previous M&S methods is less pronounced but still significant.

Our work is partly inspired by recent work (Bäckström, Jonsson, and Ståhlberg 2013) on unsolvable planning problems, testing whether *projections* onto a subset of variables (a special case of M&S) are unsolvable, where the tested variable subsets are systematically enumerated (starting with small ones). In contrast, we stick to the standard M&S process incorporating all variables, and investigate in-depth the abstraction steps (shrinking) during that process. Two prior works (Helmert 2006a; Haslum 2007) identify conditions under which a state variable can be projected away without affecting solvability. Helmert's condition (Helmert 2006a) is a special case of our techniques; Haslum's generalized condition (Haslum 2007) is not. We get back to this later.

Background

A **planning task** is a 4-tuple $\Pi = (V, A, I, G)$. V is a finite set of **variables** v, each associated with a finite domain D_v. A complete assignment to V is called a **state**; we identify (partial) assignments to V with sets of **facts**, i. e., variable-value pairs. I is the **initial state**, and the **goal** G is a partial assignment to V. A is a finite set of **actions**. Each action $a \in A$ is a pair (pre_a, eff_a) of partial assignments to V called **precondition** and **effect**, respectively. Each action is also

associated with a real-valued **cost**.

The semantics of planning tasks are defined via their **state spaces**, which are (labeled) **transition systems**. Such a system is a 5-tuple $\Theta = (S, L, T, I, S_G)$ where S is a finite set of states, L is a finite set of **labels**, $T \subseteq S \times L \times S$ is a set of **transitions**, $I \in S$ is the **initial state**, and $S_G \subseteq S$ is the set of **goal states**. We will usually write transitions $(s, l, s') \in T$ as $s \xrightarrow{l} s'$, or $s \to s'$ if the label does not matter. The state space of a planning task Π is the transition system Θ where: S is the set of all states; $L = A$; $s \in S_G$ if $G \subseteq s$; and $s \xrightarrow{a} s'$ if a is **applicable** to s and s' is the **resulting state**. Here, a is applicable to s if $\mathit{pre}_a \subseteq s$, and s' is the resulting state if $s'(v) = \mathit{eff}_a(v)$ where $\mathit{eff}_a(v)$ is defined, and $s'(v) = s(v)$ elsewhere. Π is **solvable** if Θ has a path from I to a state in S_G. We sometimes call Θ the "concrete" state space to distinguish it from abstract ones.

For a state s, **remaining cost** $h^*(s)$ is defined as the cost of a cheapest path from s to a state in S_G, or ∞ if there is no such path. A **heuristic** is a function $h : S \to \mathbb{R}_0^+ \cup \{\infty\}$. A heuristic is **perfect** if it coincides with h^*. Herein, we consider heuristics based on **abstractions**. An abstraction is a function α mapping S to a set of **abstract states** S^α. The **abstract state space** Θ^α is $(S^\alpha, L, T^\alpha, I^\alpha, S^\alpha_G)$, where $\alpha(s) \xrightarrow{l} \alpha(s')$ in T^α iff $s \xrightarrow{l} s'$ in T, $I^\alpha = \alpha(I)$, and $S^\alpha_G = \{\alpha(s) \mid s \in S_G\}$. The **abstraction heuristic** h^α maps each s to the remaining cost of $\alpha(s)$ in Θ^α. We will sometimes consider the **induced equivalence relation** $\sim^\alpha$, where $s \sim^\alpha t$ if $\alpha(s) = \alpha(t)$. If $s \sim^\alpha t$, we also say that s and t are **aggregated** by α.

Merge-and-shrink (Dräger, Finkbeiner, and Podelski 2006; Helmert, Haslum, and Hoffmann 2007; Dräger, Finkbeiner, and Podelski 2009; Helmert et al. 2014), short **M&S**, is a practical method to construct abstractions. The approach builds the abstraction in an incremental fashion, iterating between *merging* and *shrinking* steps. Namely, M&S abstractions are constructed using the following rules:

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

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

(iii) If α_1 and α_2 are M&S abstractions over disjoint sets W_1 and W_2, then $\alpha_1 \otimes \alpha_2$ is an M&S abstraction over $W_1 \cup W_2$.

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 $(\alpha_1 \otimes \alpha_2)(s) := (\alpha_1(s), \alpha_2(s))$.

Throughout the construction of α, for every intermediate abstraction β, M&S also maintains the corresponding abstract state space Θ^β. The details are not relevant to our work here.

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). Like all prior work on M&S in planning, we will use **linear and full merging strategies** only, where the variables V are ordered $v_1, \ldots, v_n$ (hence "linear") and we iteratively merge v_1 with v_2, merge their product with v_3, and so on until all variables have been merged (hence "full"). Prior to every merging step, a shrinking step is applied to both, the **current abstraction** over $\{v_1, \ldots, v_i\}$ and the atomic projection onto the variable v_{i+1} to be merged-in next.

Following recent work (Katz, Hoffmann, and Helmert 2012), each shrinking step is based on the notion of **K-catching bisimulation**. If $\Theta = (S, L, T, I, S_G)$ is a transition system and $K \subseteq L$ is a subset of its labels, then an equivalence relation $\sim$ on S is a K-catching bisimulation for Θ if $s \sim t$ implies that: (a) either $s, t \in S_G$ or $s, t \notin S_G$; (b) for every $l \in K$ we have that $\{[s'] \mid s \xrightarrow{l} s'\} = \{[t'] \mid t \xrightarrow{l} t'\}$, where $[s]$ for a state s denotes the equivalence class of s. An abstraction α is a K-catching bisimulation if the induced equivalence relation $\sim^\alpha$ is. Intuitively, a K-catching bisimulation (a) preserves goal states, and (b) preserves the behavior of transitions labeled with K. If $K = L$ then α is called a **bisimulation**, and preserves all transition behavior exactly. Note that a bisimulation does not actually have to make any aggregations: the identity function is a bisimulation. Whenever we say "K-catching bisimulation", we mean the **coarsest** one, aggregating maximally. Given a transition system Θ as input, coarsest K-catching bisimulations can be computed efficiently.

In difference to previous works, we will consider **composed shrinking strategies**, that (within every shrinking step) sequentially apply individual (component) shrinking steps. We will give each individual strategy a name "X"; "X+Y" is the sequential application of X and Y in that order. The strategy names will be postfixed with "-shrinking". The **K-shrinking** strategy chooses a subset $K \subseteq A$ of actions up front in a pre-process, and whenever rule (ii) is applied, defines γ as the coarsest K-catching bisimulation for Θ^β. When using full bisimulation ($K = A$), the strategy is called **A-shrinking**.

It is easy to see that K-catching bisimulation is invariant over M&S steps (i–iii). So, with K-shrinking, the outcome of M&S is a K-catching bisimulation of the concrete state space Θ, and particular choices of K allow to guarantee qualities of h^α. The simple limiting case is A-shrinking where h^α is perfect. More interesting choices of K were first explored by Katz et al. (Katz, Hoffmann, and Helmert 2012); we will adapt their observations to the unsolvability setup considered herein.

We run M&S with **label reduction** (Helmert, Haslum, and Hoffmann 2007): The transition labels $a = (\mathit{pre}_a, \mathit{eff}_a)$ in the current abstraction over the already merged-in variables $W = \{v_1, \ldots, v_i\}$ are projected onto $V \setminus W$. This yields the same heuristic, but it saves memory as previously distinct labels may collapse, and it can reduce bisimulation size exponentially.

For any $W \subseteq V$, we use Θ^W as a short-hand for the abstract state space Θ^{π_W} of the projection onto W. Any M&S abstraction α over W can be cast as an abstraction of Θ^W.

We will use s, t to denote concrete states, s^α, t^α to denote abstract states, and s^W, t^W to denote projected states. Any abstract state s^α is identified with a set of states, namely the equivalence class of states mapped to s^α. We will view abstract states as both, sets of concrete states s from Θ, and sets of projected states s^W from Θ^W. We sometimes denote assignments $\bigcup_{v \in U}\{v = d\}$ to a subset of variables U simply by d^U.

Unsolvability Heuristics

The definition of "unsolvability heuristic" is trivial. But as this is the basic concept distinguishing our setup from traditional heuristic search, and as that concept has (as best we know) not been introduced before, it seems appropriate to give it a name and make it explicit:[1]

Definition 1 *An **unsolvability heuristic** is a function $u : S \to \{0, \infty\}$ such that $u(s) = \infty$ only if $h^*(s) = \infty$.*

Our function u now merely indicates whether a state is recognized to be unsolvable ($u(s) = \infty$), or not ($u(s) = 0$).[2] Such truncated heuristics are useful for (a) search on unsolvable problems, and (b) dead-end detection in solvable problems. We can trivially obtain unsolvability heuristics from regular ones:

Definition 2 *Let h be a heuristic that returns $h(s) = \infty$ only if $h^*(s) = \infty$. Then the **induced unsolvability heuristic** $h|_u$ is defined by $h|_u(s) = \infty$ if $h(s) = \infty$, and $h|_u(s) = 0$ otherwise.*

*The **perfect unsolvability heuristic** u^* is defined by $u^* = h^*|_u$, and an unsolvability heuristic u is **perfect** if $u = u^*$.*

Note the close connection to "disregarding action costs": Denoting by $\Pi[0]$ the planning task with all action costs reduced to 0, $h|_u$ is perfect iff h is perfect in $\Pi[0]$. Moreover, for the abstraction heuristics we consider here, and more generally for any heuristic h whose $\mathbb{R}_0^+$ (i. e., non-∞) return values result from summing up action costs in an approximate solution, we have $h|_u = h(\Pi[0])$.

Unsolvability-Perfect M&S Abstractions

Abstractions induce unsolvability heuristics in the obvious manner. Focusing on M&S, in this and the next section we are concerned with conditions under which such use of abstractions is loss-free, i. e., where the resulting unsolvability heuristics are perfect:

Definition 3 *Let α be an abstraction. Then u^α is defined by $u^\alpha = h^\alpha|_u$. We say that α is **unsolvability perfect** if, for every pair s, t of states in Θ where $s \sim^\alpha t$, $u^*(s) = \infty$ iff $u^*(t) = \infty$.*

It is easy to see that u^α is perfect iff α is unsolvability perfect. We derive "safety" conditions on M&S, guaranteeing the latter property:

[1] Throughout the paper, we tacitly assume a planning task $\Pi = (V, A, I, G)$ with state space $\Theta = (S, A, T, I, S_G)$.

[2] We could in principle choose arbitrary return values to make this indication. The chosen ones $\{0, \infty\}$ are natural in that they correspond to disregarding action costs (see next).

Definition 4 *Let $W \subseteq V$ and let s^W, t^W be projected states in Θ^W. Then s^W and t^W are **safe to aggregate** if, for every assignment $d^{V \setminus W}$ to $V \setminus W$, $u^*(s^W \cup d^{V \setminus W}) = \infty$ iff $u^*(t^W \cup d^{V \setminus W}) = \infty$.*

*Let α be an abstraction of Θ^W. An abstract state s^α is **safe** if, for every pair of projected states $s^W, t^W \in s^\alpha$, s^W and t^W are safe to aggregate; α is safe if all its abstract states are.*

For $W = V$, being safe is equivalent to being unsolvability perfect. But not for $W \subsetneq V$: The aggregated states $s \sim^\alpha t$ in Θ are, then, all $s = s^W \cup d_s^{V \setminus W}$, $t = t^W \cup d_t^{V \setminus W}$ where $s^W \sim^\alpha t^W$ and $d_s^{V \setminus W}$, $d_t^{V \setminus W}$ are *arbitrary* extensions to the remaining variables. By contrast, safety only considers *identical* extensions $d_s^{V \setminus W} = d_t^{V \setminus W}$. This is appropriate provided that α will be merged with any safe abstraction of the remaining variables:

Lemma 1 *If α_1 and α_2 are safe abstractions of Θ^{W_1} and Θ^{W_2} respectively, then $\alpha_1 \otimes \alpha_2$ is a safe abstraction of $\Theta^{W_1 \cup W_2}$.*

Proof: Let $s^{W_1 \cup W_2}$ and $t^{W_1 \cup W_2}$ be any pair of projected states in $\Theta^{W_1 \cup W_2}$ so that $s^{W_1 \cup W_2} \sim^{\alpha_1 \otimes \alpha_2} t^{W_1 \cup W_2}$, and let $d^{V \setminus (W_1 \cup W_2)}$ be any extension to the remaining variables. Denote by s^{W_1}, t^{W_1}, s^{W_2}, and t^{W_2} the respective projections onto W_1 and W_2. By prerequisite, (1) $u^*(s^{W_1} \cup d'^{V \setminus W_1}) = \infty$ iff $u^*(t^{W_1} \cup d'^{V \setminus W_1}) = \infty$ for all extensions $d'^{V \setminus W_1}$ to $V \setminus W_1$, and (2) $u^*(s^{W_2} \cup d'^{V \setminus W_2}) = \infty$ iff $u^*(t^{W_2} \cup d'^{V \setminus W_2}) = \infty$ for all extensions $d'^{V \setminus W_2}$ to $V \setminus W_2$. Putting (1) and (2) together shows the claim: $u^*(s^{W_1 \cup W_2} \cup d^{V \setminus (W_1 \cup W_2)}) = \infty \Leftrightarrow u^*(s^{W_1} \cup s^{W_2} \cup d^{V \setminus (W_1 \cup W_2)}) = \infty \overset{(1)}{\Leftrightarrow} u^*(t^{W_1} \cup s^{W_2} \cup d^{V \setminus (W_1 \cup W_2)}) = \infty \overset{(2)}{\Leftrightarrow} u^*(t^{W_1} \cup t^{W_2} \cup d^{V \setminus (W_1 \cup W_2)}) = \infty \Leftrightarrow u^*(t^{W_1 \cup W_2} \cup d^{V \setminus (W_1 \cup W_2)}) = \infty$. □

In other words: *safety is invariant over merging steps*. Therefore, as atomic projections are trivially safe, if we start from a safe abstraction and merge in the remaining variables, then the final abstraction over all variables $W = V$ is safe and hence unsolvability perfect. Unless, of course, we apply any more shrinking steps in between.

As M&S without shrinking steps is void, our question now boils down to examining these steps. A **safe shrinking strategy** is one that, given a safe abstraction β as input, returns a safe abstraction $\gamma \circ \beta$ as its output. Obviously, if all components of a composed shrinking strategy are safe, then the composed strategy is also safe.

Corollary 1 *If the shrinking strategy is safe, then the final abstraction α of Θ is safe, and thus u^α is perfect.*

Safe Shrinking Strategies

We introduce safe shrinking strategies based on label simplifications, and safe selections of K for K-catching bisimulation.

Label Inheritance and Bisimulation

Consider any M&S abstraction over $W \subseteq V$. Consider transitions $s^W \xrightarrow{a} s'^W$ in Θ^W where every variable occurring in $a = (pre_a, eff_a)$ is contained in W. Clearly, such

transitions are **persistent** in the sense that, for every $d^{V\setminus W}$, $s^W \cup d^{V\setminus W} \to s'^W \cup d^{V\setminus W}$ is a transition in Θ. We refer to these transitions as **own-label transitions**, denoted $s^W \xrightarrow{own} s'^W$.[3] Our core observation is that we can exploit them to safely relax bisimulation:

Definition 5 *Given an M&S abstraction β of Θ^W, **ModLabel$A$-shrinking** computes an abstraction γ of Θ^β as follows:*

*(1) **Label inheritance.** Obtain transition system Θ_1 from Θ^β as follows: Set $\Theta_1 := \Theta^\beta$; whenever $s^\alpha \xrightarrow{own} t^\alpha$, s^α in Θ_1 inherits all outgoing transitions of t^α, and if t^α is an abstract goal state then s^α is made an abstract goal state in Θ_1 as well.*

*(2) **Goal-label pruning.** Obtain transition system Θ_2 from Θ_1 as follows: Set $\Theta_2 := \Theta_1$; denoting the variables on which the goal G is defined as V_G, if $V_G \subseteq W$ then remove all outgoing transitions from abstract goal states in Θ_2.*

(3) Obtain γ as a bisimulation of Θ_2, and interprete γ as an abstraction of Θ^β.

Explaining this definition bottom-up, step (3) works because all of Θ^β, Θ_1, and Θ_2 share the same set of abstract states.[4] Intuitively, step (2) is justified because β's abstract goal states will always remain goal states, so there is no point in distinguishing the ways by which we can leave them (note that this applies to any M&S abstraction, not just the ones we consider here). Intuitively, step (1) is justified because, the transition from s^α to t^α being persistent, the corresponding concrete states will have a transition in the state space, so if we only need to preserve solvability then we can just as well pretend that t^α's outgoing transitions/goal-state-flag are attached directly to s^α. Note that the latter does not work if we need to preserve path cost, as we are discounting the cost of getting from s^α to t^α.

Theorem 1 *ModLabel$A$-shrinking is safe.*

Proof Sketch: We need to prove that, for all abstract states s^β and t^β of Θ^β aggregated by bisimulation relative to Θ_2, $s^\beta \cup t^\beta$ is safe. Our proof is by assuming any s^β, t^β, and extension $d^{V\setminus W}$ where $s = s^W \cup d^{V\setminus W}$ is solvable, and proving by induction over the length n of that solution that $t = t^W \cup d^{V\setminus W}$ is solvable as well.

In the base case, $n = 0$, s is a goal state. Hence t^β must be an abstract goal state in Θ_2, which (as we're using label inheritance) implies that t^β has a path $\vec{p}$ in Θ^β of own-label transitions to an abstract state x^β that contains a goal state x_0. Because $d^{V\setminus W}$ must agree with the goal, we can assume WLOG that $x_0 = x_0^W \cup d^{V\setminus W}$. Considering the last abstract transition on $\vec{p}$, $y^\beta \to x^\beta$, we know that there exist $y_0^W \in y^\beta$ and $x_1^W \in x^\beta$ so that y_0^W has an own-label transition to x_1^W. Obtaining x_1 as $x_1 := x_1^W \cup d^{V\setminus W}$, as x^β is safe and x_0 is solvable, x_1 is solvable. Obtaining y_0 as $y_0 := y_0^W \cup d^{V\setminus W}$, as the transition $y_0^W \to x_1^W$ is persistent, there is a transition from y_0 to x_1, so y_0 is solvable. Iterating this argument backwards over $\vec{p}$, we obtain a solvable state $t_0 = t_0^W \cup d^{V\setminus W}$ in t^β. With safety of t^β, we get that $t^W \cup d^{V\setminus W}$ is solvable as well, as we needed to prove.

In the inductive case, say the length-n solution to s starts with action a, yielding resulting state s' whose solution length is $n-1$. By definition of abstractions, s^β has an outgoing transition labeled with a in Θ^β, say to abstract state s'^β.

We now need to distinguish case (1) where the transition $s^\beta \xrightarrow{a} s'^\beta$ was not removed by goal-label pruning so is still present in Θ_2; and the opposite case (2). In case (2), similarly as in the base case, we know that t^β is an abstract goal state in Θ_2; we know that $d^{V\setminus W}$ agrees with the goal simply because $V \setminus W$ cannot contain any goal variables; the rest of the proof is the same. In case (1), with Θ_2-bisimilarity of s^β and t^β, Θ_2 has a transition $t^\beta \xrightarrow{a'} t'^\beta$, where t'^β is Θ_2-bisimilar with s'^β, and a' is an action that (per label reduction, if it is applied to Θ^β) agrees with a on the variables $V \setminus W$. This implies that t^β has a path $\vec{p}$ in Θ^β of own-label transitions to an abstract state x^β that contains a state x_0 to which a' is applicable, yielding the resulting state t' where $t' \in t'^\beta$. Because a and a' agree on $V \setminus W$, we can assume WLOG that $x_0 = x_0^W \cup d^{V\setminus W}$. Applying the induction hypothesis to the states $s' = s'^W \cup d'^{V\setminus W}$ and $t' = t'^W \cup d'^{V\setminus W}$, we get that t' is solvable and hence x_0 is solvable. From there, the argument is the same as in the base case. □

Our fully detailed proof of Theorem 1 is available in a TR (Hoffmann, Kissmann, and Torralba 2014). As all aggregations made by ModLabel$A$-shrinking would be made by A-shrinking (i. e., using just bisimulation) as well, we have:

Corollary 2 *A-shrinking is safe.*

Recall that, with Corollary 1, this means that, if we use either of A-shrinking or ModLabel$A$-shrinking or any combination thereof, then the resulting u^α is perfect. Keep in mind that the same is true for all safe shrinking strategies we will identify in what follows.

Own-Label Shrinking

The problem with ModLabel$A$-shrinking, as quickly became apparent in our experiments, is that label inheritance consumes way too much runtime (and if one explicitly copies the labels, blows up memory as well). We hence defined the following sound approximation, which turns out to be very effective in practice:

Definition 6 *Given an M&S abstraction β of Θ^W, **OwnPath-shrinking** computes an abstraction γ of Θ^β as follows:*

[3] As configured here, either $W = \{v_1, \dots, v_i\}$ for the current abstraction, or $W = \{v_{i+1}\}$ for the atomic projection onto the variable v_{i+1} to be merged-in next. In the former (but not in the latter) case, own-label transitions are exactly those whose labels are empty after label reduction.

[4] We remark that the intermediate transition systems Θ_1 and Θ_2, as opposed to the final abstraction $\gamma \circ \beta$, are not abstractions of Θ in our sense, as they have additional transitions and goal states with respect to Θ.

(1) ***Own-label cycles.*** *Compute the strongly connected components C of Θ^β when considering only own-label transitions; aggregate each C into a single abstract state.*

(2) ***Own-label goal paths.*** *Denoting the variables on which the goal G is defined as V_G, if $V_G \not\subseteq W$ then do nothing. Otherwise, whenever t^α is an abstract goal state: if s^α is an abstract goal state as well then aggregate s^α and t^α into a single abstract state; else, if s^α has an own-label path to t^α, then aggregate s^α, t^α, and all states on the path into a single abstract state.*

Intuitively, (1) is sound as, with persistence of own-label paths, the strongly connected components will still be strongly connected at the end of the M&S process so are equivalent with respect to solvability. (2) is sound because, with $V_G \subseteq W$, abstract goal states remain goal states, so there is no need to distinguish them and no need to distinguish states that have a persistent path to them. For formal proof, our previous result on ModLabel$A$-shrinking is sufficient:

Lemma 2 *If a pair of abstract states is aggregated by OwnPath-shrinking, then it would be aggregated by ModLabel$A$-shrinking.*

Proof: For rule (1), as the aggregated states are strongly connected with own-label transitions, they would inherit each other's outgoing transitions; if any of them is a goal state, all would be marked as goal states. Hence they would become bisimilar, and be aggregated.

For rule (2), say s^α and t^α are aggregated. Then t^α is an abstract goal state, and as $V_G \subseteq W$, its outgoing transitions would be removed by goal-label pruning. If s^α is not already a goal, as there is an own-label path from s^α to t^α and t^α is a goal, label inheritance would mark s^α as a goal. So all outgoing transitions would be removed from s^α as well, making the two states bisimilar. □

Together with Theorem 1, this lemma immediately implies:

Theorem 2 *OwnPath-shrinking is safe.*

Once all variables are merged in (so all labels are own-labels), rule (2) will aggregate the entire solvable part of the state space into a single abstract state. Also, if a variable v has no incoming edges in the causal graph and a strongly connected DTG, then, when v is merged in, all its values are strongly connected by own-labels, so rule (1) will aggregate all values of v into a single abstract state. In our implementation, such variables v are ignored in the M&S construction.[5]

ModLabel$A$-shrinking can be exponentially stronger than OwnPath+A-shrinking, which can be exponentially stronger than using just bisimulation: (the proof is in the TR)

Theorem 3 *There exist families of planning tasks $\{\Pi_n\}$ and merging strategies so that M&S abstractions are exponentially smaller with ModLabel$A$-shrinking than with OwnPath+A-shrinking. The same is true for OwnPath+A-shrinking and A-shrinking.*

K-Catching Bisimulation

Let us finally consider $K \neq A$. This is important as catching less actions can substantially reduce bisimulation size, and as approximate methods choosing the actions to catch will be our primary method for generating approximate unsolvability heuristics.

Definition 7 *A subset K of actions is **safe**, or **path preserving**, if removing all transitions not labeled by an action from K does not render any solvable state in Θ unsolvable. K is **shortest-path preserving** if, for every solvable s in Θ, K contains an action a starting a shortest solution path from s.*

Being shortest-path preserving obviously is a sufficient condition for being path preserving, and is sometimes useful as an approximation because actions can be selected locally on a per-state basis.[6]

Theorem 4 *If K is safe, then K-shrinking is safe.*

Proof: Say β is any safe abstraction. Denote by Θ_K the concrete state space where all non-K transitions are removed. As solvability in Θ_K is the same as in Θ, β viewed as an abstraction on Θ_K is safe. By definition, any K-catching bisimulation γ of Θ^β is a bisimulation of Θ^β_K. Hence, by Corollary 2, γ is safe as an abstraction of Θ_K. Now, viewing γ as an abstraction on Θ, since solvability in Θ_K is the same as in Θ, γ is safe as we needed to prove. □

Practical M&S Strategies

Finding K guaranteed to be safe is not feasible (we would need to construct the concrete state space Θ first). Katz et al. (Katz, Hoffmann, and Helmert 2012) introduced two approximation strategies. We experimented with these as well as a variety of modified ones adapted to our context. The only one that turned out to be relevant empirically (i. e., for proving unsolvability effectively) is **Intermediate Abstraction (IntAbs):** Run A-shrinking until abstraction size has reached a parameter M. The labels are collected on that abstraction, and M&S continues with K-shrinking. M controls a trade-off as actions affecting only yet-to-be-merged variables form self-loops so will not be collected. This strategy was proposed by Katz et al. already. We proceed in the same way, but where Katz et al. collect all labels starting optimal paths, we instead collect a path preserving label set K. Trying to keep K small (finding minimum-size K is **NP**-hard in the size of the abstract state space), we start

[5] Such v are exactly those that satisfy Helmert's (Helmert 2006a) "safe abstraction" condition, so in that sense our techniques subsume that condition. The same is not true of Haslum's (Haslum 2007) generalized condition (his Theorem 1), which exploits values of v that are neither "externally required" nor "externally caused". It remains an open question whether Haslum's condition can be adapted to yield additional safe shrinking in M&S.

[6] Katz et al. define "globally relevant actions" K as the set of all actions starting a cheapest path for any solvable s. They prove that, with such K, K-shrinking yields perfect h^α. They overlook that, for that purpose, it would actually be enough to preserve at least one optimal solution path for each s.

from $K = \emptyset$ and iteratively include the action rendering the largest number of yet non-covered states solvable.

Like all previous works on M&S, we also use a parameter N which imposes an upper bound on abstraction size throughout M&S.

Merging strategies have so far been largely neglected in the planning literature: a grand total of 2 strategies has been tried (although it was observed that they can be important empirically). We conducted a comprehensive study in the context of proving unsolvability. There are two plausible main objectives for the merging strategy in that context: *(a) find an unsolvable variable subset quickly; and (b) make transition labels empty (and thus own-labels in the current abstraction) quickly, to yield smaller bisimulations and more OwnPath-shrinking.* We approximate these by lexicographic combinations of simple preference rules:

Goal: Prefer goal variables over non-goal variables. This addresses (a). It was used by Helmert et al. (Helmert, Haslum, and Hoffmann 2007) to obtain larger goal distances within the abstraction.

CG, **CGRoot**, and **CGLeaf:** Prefer variables with an outgoing causal graph arc to an already selected variable. For CGRoot and CGLeaf, if there are several such variables v, v', prefer v over v' if, in the strongly connected components (SCC) of the causal graph, that of v is ordered before that of v' (CGRoot), respectively behind that of v' (CGLeaf). This also addresses (a): unsolvability must involve connected variables, and might involve "more influential" variables close to the causal graph roots (CGRoot), respectively "more influenced" variables close to the causal graph leaves (CGLeaf). Helmert et al. used just CG, for the same reason as Goal.

Empty: Prefer variables merging which maximizes the number of empty-label transitions leading to abstract goal states. If there are several such variables v, prefer v maximizing the number of empty-label transitions, and if there are several such variables v, prefer v maximizing the number of transitions whose labels contain v. This addresses (b). It was not used in earlier works on M&S.

LevelRoot and **LevelLeaf:** Derived from FD's full linear order (Helmert 2006b). LevelRoot prefers variables "closest to be causal graph roots", and LevelLeaf prefers variables "closest to be causal graph leaves".

Variables are added one-by-one, always selecting a most preferred one next. Ties remaining after all criteria were applied are broken arbitrarily. For example, CGRoot-Goal-Empty, after selecting a goal variable, selects all its causal graph predecessors, preferring ones close to the root and yielding many empty labels. We use at most one of CG, CGRoot, and CGLeaf. We use at most one of LevelRoot and LevelLeaf, and they are included only at the end as they allow no more tie breaking. Finally, we do not use Goal at the start as that yields very bad performance (selecting only goal variables neither results in unsolvable sub-problems nor in abstraction size reductions, often breaking our memory limit before any other variable is selected). This leaves a total of 81 possible merging strategies.

Experiments

There is no standard set of unsolvable benchmarks. Bäckström et al. (Bäckström, Jonsson, and Ståhlberg 2013) have made a start, but their set consists of only 6 instances. We have vastly extended this, hoping to establish, or at least seed, a standard.[7] The benchmarks will be made available for download, and a full description will be in the TR. A brief summary follows. **Mystery** IPC'98: 9 unsolvable instances from the standard instance set (those not detected by FD's pre-processor). **UnsNoMystery**, **UnsRovers**, **UnsTPP**: As used by Nakhost et al. (Nakhost, Hoffmann, and Müller 2012) (their "large" suites for NoMystery and Rovers) with instances scaled systematically on "constrainedness" C, but using $C \in \{0.5, 0.6, 0.7, 0.8, 0.9\}$ where there are insufficient resources. **UnsTiles**: The sliding tiles puzzle with initial states from the unsolvable part of the state space; we used 10 8-Puzzle instances, and 10 (rectangular) "11-Puzzle" instances. **UnsPegsol**: As in the net-benefit track of IPC'08, but with the traditional goal state having only a single peg in the middle of the board (in this setting, all these instances are unsolvable); we skipped the 6 instances detected by FD's pre-processor. **3UNSAT** (extended from (Bäckström, Jonsson, and Ståhlberg 2013)): random unsolvable 3SAT formulas from the phase transition region, with $n \in \{5, 10, 15, 20, 25, 30\}$ variables and 5 random instances per n value. **Bottleneck** (extended from (Bäckström, Jonsson, and Ståhlberg 2013)): n agents travel to individual goal positions on an $n \times n$ grid. Once a cell has been visited, it becomes impassable. The agents all start on the left-hand side, and there is a wall in the middle with a hole of size $m < n$. We used $n \in \{4, 5, 6, 7, 8\}$, with all $m = 1, \ldots, n-1$ for each n.

All our techniques are implemented in Fast Downward. All experiments were run on a cluster of Intel E5-2660 machines running at 2.20 GHz, with runtime (memory) limits of 30 minutes (4 GB). Similarly as Katz et al. (Katz, Hoffmann, and Helmert 2012), as a hypothetical experiment we collected perfect label sets K, in instances small enough for that purpose. We cannot describe this for lack of space. The overall conclusion is that our label sets typically are smaller than Katz et al.'s, yielding mostly moderate, and sometimes strong, abstraction size reductions.

Consider Table 1. We compare against the main competing approaches for proving unsolvability, and we conduct comprehensive experiments with our practical M&S strategies. "Blind" is a heuristic that returns 0 on goal states and 1 elsewhere; note that this dominates, in terms of dead-end detection power vs. runtime overhead, any heuristic that does not have the ability to detect dead ends, such as certain kinds of landmark-based heuristics (e. g., (Karpas and Domshlak 2009)). Similarly, $h^{\max}$ is a canonical and cheap heuristic whose dead-end detection power is equivalent to h^{FF} as well as the state-of-the-art admissible heuristic LM-

[7] Bäckström et al. considered two domains, "Grid" and "Trucks", that we do *not* adopt: Unsolvability is trivially detected by h^2, and the domains appear non-natural in using a "key cycle" irrelevant to unsolvability (Grid) respectively consisting of two completely separate sub-problems (Trucks).

domain (# instances)	Blind	$h^{\max}$	BJS	H^2	BDD std	BDD H^2	BestOf KHH N100k M100k OldMrg	BestOf KHH N100k M100k Mrg1	A OldMrg	A Mrg1	Own+A std	Own+A H^2	MLA	Own+K N1m M500k std	Own+K N1m M500k $h^{\max}$	Own+K N1m M500k std KKHH	Own+K N1m M500k $h^{\max}$ KKHH	Own+K N100k M100k std	Own+K N100k M100k $h^{\max}$	Merging Strategies Own+A Mrg1	Merging Strategies Own+A Mrg2	Merging Strategies Own+A Mrg3
Bottleneck (25)	10	21	10	10	10	15	10	10	5	5	5	10	5	9	15	4	4	10	21	5	11	7
3UNSAT (30)	15	15	0	0	15	15	15	15	15	15	15	15	14	14	14	12	15	15	15	15	12	15
Mystery (9)	2	2	6	9	3	9	2	6	1	6	6	9	5	6	6	6	6	6	6	6	1	1
UnsNoMystery (25)	0	0	8	0	5	14	23	23	25	25	25	25	15	25	25	25	25	25	25	25	25	23
UnsPegsol (24)	24	24	0	0	24	24	24	24	24	24	24	24	0	24	24	24	24	24	24	24	0	0
UnsRovers (25)	0	1	3	3	6	10	0	9	0	17	17	17	7	11	11	11	11	9	9	17	17	0
UnsTiles (20)	10	10	10	0	10	10	10	10	0	0	10	10	0	10	10	10	10	10	10	10	10	10
UnsTPP (25)	5	5	2	1	0	1	14	11	17	9	9	9	3	11	8	10	8	11	9	9	17	19
Total (183)	66	78	39	23	73	98	98	108	87	101	111	119	49	110	113	102	103	110	119	111	93	75

Table 1: Coverage results on unsolvable benchmarks, i. e., number of instances proved unsolvable within the time/memory bounds. "Mrg1" stands for CGRoot-Goal-LevelLeaf, "Mrg2" for Empty-CGRoot-Goal-LevelLeaf, "Mrg3" for CGLeaf-Goal, and "OldMrg" for the shrinking strategy of (Helmert, Haslum, and Hoffmann 2007).

cut (Helmert and Domshlak 2009). "H^2" runs h^2 (Haslum and Geffner 2000) just once, on the initial state; we use the implementation of Torralba and Alcázar's recent work on constrained BDDs (Torralba and Alcázar 2013), where h^2 forms part of an extended FD pre-processor. "BDD H^2" are these constrained BDDs. "BDD std" is that implementation with all h^2 parts switched off (thus representing a standard BDD state space exhaustion). "(Bäckström, Jonsson, and Ståhlberg 2013)" is Bäckström et al.'s enumeration of projections (their implementation in C#). We did not run h^m heuristics (for $m > 2$) (Haslum and Geffner 2000) and pattern database heuristics (Haslum et al. 2007) because they are dominated by "(Bäckström, Jonsson, and Ståhlberg 2013)" in Bäckström et al.'s paper (plus, the h^m implementation in FD is extremely ineffective, and pattern databases are not geared towards proving unsolvability); we leave a detailed comparison to these heuristics for future work.

Regarding M&S strategies, "BestOf (Katz, Hoffmann, and Helmert 2012)" is, for each of the two underlying merging strategies, the best-performing M&S configuration (in terms of total coverage on our benchmarks) of the 12 ones shown in Table 2 of (Katz, Hoffmann, and Helmert 2012); the same configuration N=100k M=100k is best for both merging strategies.[8] "A" is for A-shrinking, "Own+A" for OwnPath+A-shrinking, "MLA" for ModLabel$A$-shrinking, and "Own+K" for OwnPath+K-shrinking. We run a strategy geared towards selecting an accurate label set and not doing much additional shrinking (N=1million M=500k), and a strategy geared towards greedy label selection and shrinking (N=100k M=100k, like in BestOf (Katz, Hoffmann, and Helmert 2012)). In the "$h^{\max}$" variants of Own+K, the heuristic we use is $\max(h^{\max}, u^{\alpha})$. In the "$K$(Katz, Hoffmann, and Helmert 2012)" variants, we use Katz et al.'s "globally relevant labels" (the best label selection method in (Katz, Hoffmann, and Helmert 2012)) instead of our path preserving label set. All heuristic functions (except h^2) are run in greedy best-first search.

Let us first discuss merging strategies (rightmost part of Table 1). For this part of the evaluation, we fixed Own+A as a canonical well-performing shrinking strategy. It turns out that, of the 81 possible merging strategies, 3 are enough to represent the highest coverage achieved in every domain. CGRoot-Goal-LevelLeaf (Mrg1) has maximal total coverage, as well as maximal coverage in all domains except Bottleneck and UnsTPP. Empty-CGRoot-Goal-LevelLeaf (Mrg2) has maximal coverage among a total of 13 merging strategies that achieve coverage 11 and 17 in Bottleneck and UnsTPP, respectively. CGLeaf-Goal (Mrg3) is the only strategy with coverage > 17 in UnsTPP. The reasons for this behavior are fairly idiosyncratic per domain. CGRoot-Goal-LevelLeaf seems to make a good compromise between "influential" and "influenced" variables (note here how these two conflicting directions are traded against each other via a preference for "more influential" variables in CGRoot and a preference for "more influenced" variables in LevelLeaf).

For the evaluation of shrinking strategies (middle part of Table 1), we fixed the best merging strategy (Mrg1). The only exceptions are BestOf KHH and A, where we also ran the best previous merging strategy ("OldMrg"), for comparison.

The competing approaches (leftmost part of Table 1) are clearly outperformed by M&S. Coverage in most cases is dominated either by Own+A or by Own+K with N=100k M=100k. The most notable exception is $h^{\max}$, which is best in Bottleneck. The "H^2" column for Own+A employs Torralba and Alcázar's (Torralba and Alcázar 2013) extended FD pre-processor. This shows that Own+A benefits as well, though not as drastically as BDD H^2, because in difference to that approach which uses h^2 mutexes to prune the BDDs, we do not use these mutexes within the M&S abstraction; doing so is a topic for future work.

The closest competitors are the previous M&S configurations, i. e., BestOf KHH and A. From the OldMrg vs. Mrg1 columns, the importance of our new merging strategies is immediately apparent.

For OwnPath-shrinking, compare "A Mrg1" vs. "Own+A std" (which differ only in not using vs. using OwnPath-shrinking). Own+A has a coverage advantage, but only due to the sliding tiles puzzle. Apart from that domain,

[8] In (Katz, Hoffmann, and Helmert 2012), that configuration is listed as "N=∞ M=100k", but there was a bug in the implementation causing it to behave exactly like N=100k M=100k.

domain	commonly solved instances	$h^{\max}$	OwnPath+K N1m M500k std	N1m M500k $h^{\max}$	N100k M100k std	N100k M100k $h^{\max}$
Bottleneck	9	1844.61	1.45	21560.89	2.74	28022.86
3UNSAT	14	3.18	∞	∞	∞	∞
Mystery	2	5.26	∞	∞	∞	∞
UnsPegsol	24	1.84	1.01	1.86	1.01	1.86
UnsTiles	10	1.00	1.00	1.00	1.00	1.00
UnsTPP	4	49.99	∞	∞	4450.88	4572.16

Table 2: Number of expansions relative to blind search: Median, over instances commonly solved by all shown approaches, of the ratio blind/X, taken to be ∞ where X has 0 expansions.

OwnPath-shrinking yields significant advantages in NoMystery, and moderate advantages in Bottleneck. This does not result in increased coverage here, but results in increased coverage, e. g., in Nakhost et al.'s (Nakhost, Hoffmann, and Müller 2012) "small" NoMystery test suite (which contains less packages etc. but 2 trucks instead of 1): Coverage goes up from 84% to 100% when C is close to 1, i. e., when there is just not enough fuel. In our other domains, OwnPath-shrinking has no effect at all. The picture is similar for approximate strategies, i. e., for (OwnPath+)K-shrinking. ModLabel$A$-shrinking (MLA), on the other hand, yields some reduction in all domains except UnsPegSol, but never pays off due to the overhead it incurs.

For the effect of our new label catching strategy, consider the Own+K part of the table. When using Katz et al.'s "globally relevant labels" (K(Katz, Hoffmann, and Helmert 2012)), leaving everything else the same (in particular still using OwnPath-shrinking), coverage remains the same for N=100k M=100k and hence no separate data is shown. But performance does become considerably weaker for N=1m M=500k. Katz et al.'s method, while selecting more labels resulting in more expensive abstractions, does not provide more accurate estimates. This is drastic in Bottleneck, reducing coverage, and yields larger total runtimes in all other domains (except 3UNSAT with $h^{\max}$) as well, most significantly in UnsPegSol with a mean of 200 vs. 76 seconds.

Table 2 sheds some light on the number of expansions required by approximate approaches (imperfect unsolvability heuristics). In difference to $h^{\max}$, our M&S strategies yield excellent dead-end detectors in half of these domains. In Bottleneck, where $h^{\max}$ is drastically better, combining both heuristics yields an advantage (which does not pay off in total runtime, due to the abstraction overhead). The intended advantage of N1m M500k over N100k M100k, yielding a more accurate heuristic, manifests itself in UnsTPP, as well as in 3UNSAT and UnsPegsol (not visible in the median) and UnsRovers (not contained in this table for lack of commonly solved instances).

Conclusion

A crystal clear message from our experiments is that *heuristic search, in particular with M&S heuristics, is a viable method to prove unsolvability in planning*. It clearly beats BDDs, a method traditionally used for state space exhaustion. The empirical impact of our merging strategies is good. Our theory results (i. e., OwnPath-shrinking) yield significant advantages in 2 of 8 domains. It remains an open question whether that can be improved, e. g., by approximating ModLabel$A$-shrinking more tightly or by exploiting Haslum's (Haslum 2007) notions.

The big open lines of course are the use of unsolvability heuristics for dead-end detection on solvable tasks (we had limited success with this so far), and tailoring other heuristics to unsolvability detection. An example that immediately springs to mind are semi-relaxed plan heuristics obtained from explicit compilation of a fact conjunction set C (Keyder, Hoffmann, and Haslum 2012), where (a) unsolvability heuristics correspond to $h^{\max}$ so are easier to extract, and (b) one may tailor the selection of C.

References

Bäckström, C.; Jonsson, P.; and Ståhlberg, S. 2013. Fast detection of unsolvable planning instances using local consistency. In Helmert, M., and Röger, G., eds., *Proceedings of the 6th Annual Symposium on Combinatorial Search (SOCS'13)*, 29–37. AAAI Press.

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

Dräger, K.; Finkbeiner, B.; and Podelski, A. 2006. Directed model checking with distance-preserving abstractions. In Valmari, A., ed., *Proceedings of the 13th International SPIN Workshop (SPIN 2006)*, volume 3925 of *Lecture Notes in Computer Science*, 19–34. Springer-Verlag.

Dräger, K.; Finkbeiner, B.; and Podelski, A. 2009. Directed model checking with distance-preserving abstractions. *STTT* 11(1):27–37.

Edelkamp, S.; Lluch-Lafuente, A.; and Leue, S. 2004. Directed explicit-state model checking in the validation of communication protocols. *International Journal on Software Tools for Technology*.

Gerevini, A.; Haslum, P.; Long, D.; Saetti, A.; and Dimopoulos, Y. 2009. Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. *Artificial Intelligence* 173(5-6):619–668.

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.

Haslum, P. 2007. Reducing accidental complexity in planning problems. In Veloso, M., ed., *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, 1898–1903. Hyderabad, India: Morgan Kaufmann.

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

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

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. 2006a. Fast (diagonally) downward. In *IPC 2006 planner abstracts*.

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

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

Hoffmann, J.; Kissmann, P.; and Torralba, A. 2014. "Distance"? Who Cares? Tailoring merge-and-shrink heuristics to detect unsolvability. Technical report, Saarland University. Available at `http://fai.cs.uni-saarland.de/hoffmann/papers/tr14.pdf`.

Karpas, E., and Domshlak, C. 2009. Cost-optimal planning with landmarks. In Boutilier, C., ed., *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, 1728–1733. Pasadena, California, USA: Morgan Kaufmann.

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

Katz, M.; 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'12)*, 101–109. AAAI Press.

Keyder, E.; Hoffmann, J.; and Haslum, P. 2012. Semi-relaxed plan heuristics. In Bonet, B.; McCluskey, L.; Silva, J. R.; and Williams, B., eds., *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*, 128–136. AAAI Press.

Nakhost, H.; Hoffmann, J.; and Müller, M. 2012. Resource-constrained planning: A monte carlo random walk approach. In Bonet, B.; McCluskey, L.; Silva, J. R.; and Williams, B., eds., *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*, 181–189. AAAI Press.

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

Torralba, A., and Alcázar, V. 2013. Constrained symbolic search: On mutexes, BDD minimization and more. In Helmert, M., and Röger, G., eds., *Proceedings of the 6th Annual Symposium on Combinatorial Search (SOCS'13)*, 175–183. AAAI Press.

Learning Pruning Rules for Heuristic Search Planning

Michal Krajňanský and **Jörg Hoffmann**
Saarland University
Saarbrücken, Germany
{krajnansky,hoffmann}@cs. uni-saarland.de

Olivier Buffet
INRIA / Université de Lorraine
Nancy, France
olivier.buffet@loria.fr

Alan Fern
Oregon State University
Corvallis, USA
afern@eecs.oregonstate.edu

Abstract

When it comes to learning control knowledge for planning, most works focus on "how to do it" knowledge which is then used to make decisions regarding which actions should be applied in which state. We pursue the opposite approach of learning "how to *not* do it" knowledge, used to make decisions regarding which actions should *not* be applied in which state. Our intuition is that "bad actions" are often easier to characterize than "good" ones. An obvious application, which has not been considered by the few prior works on learning bad actions, is to use such learned knowledge as action pruning rules in heuristic search planning. Fixing a canonical rule language and an off-the-shelf learning tool, we explore a novel method for generating training data, and implement rule evaluators in state-of-the-art planners. The experiments show that the learned rules can yield dramatic savings, even when the native pruning rules of these planners, i.e., preferred operators, are already switched on.

Introduction

Learning can be applied to planning in manifold ways (see (Celorrio et al. 2012) for a recent overview). To name a few, existing approaches include learning of action models (e.g., (Walsh and Littman 2008)), learning to predict planner performance (e.g., (Roberts and Howe 2009; Cenamor, de la Rosa, and Fernández 2013)), learning macro actions (e.g., (Botea et al. 2005; Newton et al. 2007; Coles and Smith 2007)), learning to improve a heuristic (e.g., (Yoon, Fern, and Givan 2008; Xu, Fern, and Yoon 2009; Virseda, Borrajo, and Alcázar 2013)), learning which heuristic to use when (Domshlak, Karpas, and Markovitch 2012), and learning portfolio configurations (e.g., (Núñez, Borrajo, and López 2012; Seipp et al. 2012)).

The approach we pursue here is the venerable (i.e., old) idea of learning *control knowledge*, in the sense of "domain-dependent information about the structure of plans". That approach has a long tradition, focusing almost entirely on "how to do it" knowledge, mostly learning representations of closed-loop action-selection policies or open-loop macro actions. Learned policies are often used for search-free plan generation (e.g., (Khardon 1999; Martin and Geffner 2000; Yoon, Fern, and Givan 2002; Fern, Yoon, and Givan 2006; Gretton 2007; Xu, Fern, and Yoon 2010; de la Rosa et al. 2011; Srivastava, Immerman, and Zilberstein 2012)), while learned macros are typically integrated into complete heuristic search algorithms (e.g., (Botea et al. 2005; Newton et al. 2007; Coles and Smith 2007)). However, recentwork has also used learned policies for macro generation during search (e.g., (Yoon, Fern, and Givan 2008; de la Rosa et al. 2011)).

In this work, we pursue an alternative approach of learning "how to *not* do it" knowledge. Consider, e.g., Sokoban. Finding the "good" actions in many critical states is very hard to do, as it effectively entails search or already knowing what the solution is. In contrast, with a bit of practice it is often easy to avoid clearly "bad" actions (like, blocking an exit) based on simple features of the state. A plausible hypothesis therefore is that it may be easier to learn a representation that is able to reliably identify *some* of the bad actions in a state, compared to learning to reliably select a good action.[1]

Indeed, in the literature on search, *pruning rules* – conditions under which the search discards an applicable action – play a prominent role. Temporal logic pruning rules are highly successful in hand-tailored planning with TLPlan (Bacchus and Kabanza 2000) and TALPlanner (Kvarnström and Magnusson 2003). Pruning rules derived as a side effect of computing a heuristic function, commonly referred to as *helpful actions* or *preferred operators*, are of paramount importance to the performance of domain-independent heuristic search planners like FF (Hoffmann and Nebel 2001), Fast Downward (Helmert 2006), and LAMA (Richter and Westphal 2010). In fact, it has been found that such pruning typically is more important to performance than the differences between many of the heuristic functions that have been developed (Richter and Helmert 2009).

Despite the prominence of pruning from a search perspective, hardly any research has been done on learning to characterize bad actions (presumably due to the traditional focus on learning stand-alone knowledge as opposed to helping

[1] Note the "some" here: learning to reliably identify *all* bad actions is equivalent to learning to identify all good actions. Our focus is on learning a *subset* of the bad actions. From a machine learning perspective, this corresponds to the precision-recall trade-off. We are willing to sacrifice recall (the percentage of bad actions that are pruned), in favor of precision (the percentage of pruned actions that are bad). This makes sense as it avoids removing solutions from the search space.

a search algorithm). To the best of our knowledge, there are exactly two such prior works. Considering SAT-based planning, Huang et al. (Huang, Selman, and Kautz 2000) learn simple datalog-style conjunctive pruning rules, conveniently expressed in the form of additional clauses. They find this method to be very effective empirically, with speed-ups of up to two orders of magnitude on a collection of mostly transport-type domains (although, from today's perspective, it should be mentioned that the original planner, but not the one using the pruning rules, is time-step optimal). More recently, de la Rosa and McIlraith (de la Rosa and McIlraith 2011) tackled the long-standing question of how to automatically derive the control knowledge for TLPlan and TALPlanner. Accordingly, their pruning rules are formulated in linear temporal logic (LTL); they introduce techniques to automatically generate derived predicates to expand the feature space for these rules. Experiments in three domains show that these rules provide for performance competitive with that of hand-written ones.

Against this background, our work is easy to describe: Like de la Rosa and McIlraith, we hook onto the search literature in attempting to learn a prominent form of pruning; while de la Rosa and McIlraith considered TLPlan, we consider action pruning (à la preferred operators) in heuristic search planning. The idea is to let that powerful search framework do the job of finding the "good" actions, reducing our job to helping out with quickly discarding the bad ones. Like Huang et al., we concentrate on simple datalog-style conjunctive pruning rules, the motivation being to determine first how far such a simple framework carries. (More complex frameworks, and in particular the application of de la Rosa and McIlraith's rules in heuristic search planning, are left open as future topics.) We also diverge from prior work in the generation of training data, which we derive comprehensively from all optimal states as opposed to just the states visited by one (or a subset of) solutions.

As it turns out, our simple approach is quite promising. Experimenting with the IPC'11 learning track benchmarks, we obtain dramatic speed-ups over standard search configurations in Fast Downward, on several domains. The speed-ups are counter-balanced by equally dramatic losses on other domains, but a straightforward portfolio approach suffices to combine the complementary strengths of the different configurations involved.

We next introduce our notations. We then detail our features for learning, the generation of training data, our formulation of pruning rules and how they are being learned, as well as their usage during the search. We present our experiments and conclude.

Preliminaries

Our approach requires that states be represented as sets of instantiated first-order atoms (so we can learn first-order conjunctive pruning conditions), that actions are instantiated action schemas (so the pruning conditions can be interpreted as rules disallowing particular schema instantiations in a given state), and that the first-order predicates and the action schemas are shared across the entire planning domain (so the rules can be transferred across instances of the domain). Apart from this, we don't need to make any assumptions, in particular as to how exactly action schemas are represented and how their semantics is defined.

Our assumptions are obviously satisfied by sequential planning in all variants of deterministic non-metric non-temporal PDDL. Our pruning rules are designed for use during a forward search. In our concrete implementation, we build on FF (Hoffmann and Nebel 2001) and Fast Downward (FD) (Helmert 2006). In what follows, we introduce minimal notation as will be needed to describe our techniques and their use in forward search.

We presume a fixed *planning domain* D, associated with a set P of first-order *predicates*, each $p \in P$ with *arity* $arity_p$; we identify p with a string (its "name"). D is furthermore associated with a set A of *action schemas*, each of which has the form $a[X]$ where a is the schema's name and X is a tuple of *variables*; we will sometimes identify X with the set of variables it contains.

A *first-order atom* has the form $p[X]$ where $p \in P$ and X is an $arity_p$-tuple of variables; like for action schemas, we will sometimes identify X with the set of variables it contains. A *first-order literal* $l[X]$ is either a first-order atom $p[X]$ (a *positive literal*), or a negated first-order atom $\neg p[X]$ (a *negative literal*).

An *instance* Π of the domain D comes with a set O of *objects*. A *ground atom* then has the form $p[o_1, \ldots, o_k]$ where $p \in P$, $o_i \in O$, and $k = arity_p$. *Ground literals* are defined in the obvious manner. A *ground action* has the form $a[o_1, \ldots, o_k]$ where $a[X] \in A$, $o_i \in O$, and $k = |X|$; we will often denote ground actions simply with "a". A *state* s is a set of ground atoms.

Each domain instance Π is furthermore associated with a state I called the *initial state*, and with a set G of ground atoms called the *goal*. A state s is a *goal state* if $G \subseteq s$.

If s is a state and a is a ground action, then we assume that there is some criterion stating whether a is *applicable* to s, and what the *resulting state* of applying a to s is. A *solution* (or *plan*) for a domain instance is a sequence of ground actions that is iteratively applicable to I, and whose iterated application results in a goal state. The solution is *optimal* if its length is minimal among all solutions. (For simplicity, we do not consider more general action costs, although our approach is applicable to these in principle.)

Features

A basic decision is which features to use as input for the learning algorithm. Many previous works on learning control knowledge for states (e.g., (Yoon, Fern, and Givan 2008; Xu, Fern, and Yoon 2010; de la Rosa et al. 2011; de la Rosa and McIlraith 2011; Virseda, Borrajo, and Alcázar 2013)) used features different from the state itself, or in addition to the state itself. We did not do that for now, as the simpler approach already led to good results. However, of course, *whether an action is "good" or "bad" often depends on the goal*. As the goal is not reflected in the states during a forward search, we need to *augment* the states with that information.

Given a domain instance Π and a predicate p, denote by $Goal(p)$ some new predicate unique to p (in our implemen-

tation, $Goal(p)$ prefixes p's name with the string "$Goal$-"), and with the same arity as p. The **augmented predicates** are obtained as $P \cup \{ Goal(p) \mid p \in P \}$. Given a state s in Π, the **augmented state** is obtained as $s \cup \{ Goal(p)[o_1, \ldots, o_k] \mid p[o_1, \ldots, o_k] \in G \}$ where G is the instance's goal. In words, we make goal-indicator copies of the predicates, and introduce the respective ground atoms into the states. We assume from now on that this operation has been performed, without explicitly using the word "augmented". The input to the learning algorithm are (augmented) states, the learned rules employ (augmented) predicates, and the rule usage is based on evaluating these (augmented) predicates against (augmented) states during the search.

For example, in a transportation domain with predicate $at[x, y]$, we introduce the augmented predicate $Goal\text{-}at[x, y]$. If $at[o_1, c_2] \in G$ is a goal atom, we augment all states with $Goal\text{-}at[o_1, c_2]$. In our experiments, the majority of the learned rules ($\geq 70\%$ in 5 of 9 domains) contain at least one augmented predicate in the rule condition.

Generating the Training Data

The pruning rules we wish to learn are supposed to represent, given a state s, what are the "bad action choices", i.e., which applicable ground actions should not be expanded by the search. *But when is an action "bad" in a state? How should we design the training data?*

Almost all prior approaches to learning control knowledge (e.g., (Khardon 1999; Martin and Geffner 2000; Yoon, Fern, and Givan 2002; Fern, Yoon, and Givan 2006; Yoon, Fern, and Givan 2008; Xu, Fern, and Yoon 2010)) answer that question by choosing a set of *training problem instances*, generating a single plan for each, and extracting the *training data* from that plan. In case of learning which actions should be applied in which kinds of states, in particular, it is basically assumed that the choices made by the plan – the action a applied in any state the plan s visits – are "good", and every other action a' applicable to these states s is "bad". Intuitively, the "good" part is justified as the training plan works for its instance, but the "bad" part ignores the fact that other plans might have worked just as well, resulting in noisy training data. Some prior approaches partly counter-act this by removing unnecessary ordering constraints from the plan, thus effectively considering a subset of equally good plans. However, those approaches are incomplete and can still mislabel "good" actions as "bad". Herein, we employ a more radical approach based on generating *all* optimal plans.

We assume any planning tool that parses domain D and an instance Π, that provides the machinery to run forward state space search, and that provides an admissible heuristic function h. To generate the training data, we use A* with small modifications. Precisely, our base algorithm is the standard one for admissible (but potentially inconsistent) heuristics: best-first search on $g + h$ where g is path length; maintaining a pointer to the parent node in each search node; duplicate pruning against all generated states, updating the parent pointer (and re-opening the node if it was closed already) if the new path is cheaper. We modify two aspects of this algorithm, namely (a) the termination condition and (b) the maintenance of parent pointers.

For (a), instead of terminating when the first solution is found, we stop the search only when the best node in the open list has $g(s) + h(s) > g^*$ where g^* is the length of the optimal solution (which we found beforehand). For (b), instead of maintaining just one pointer to the best parent found so far, we maintain a list of pointers to all such parents. Thanks to (a), as $g(s) + h(s)$ is a lower bound on the cost of any solution through s, and as all other open nodes have at least value $g + h$, upon termination we must have generated all optimal solutions. Thanks to (b), at that time we can find the set S^* of all states on optimal plans very easily: Simply start at the goal states and backchain over all parent pointers, collecting all states along the way until reaching the initial state. The training data then is:

- **Good examples E^+:** Every pair (s, a) of state $s \in S^*$ and ground action a applicable to s where the outcome state s' of applying a to s is a member of S^*.
- **Bad examples E^-:** Every pair (s, a) of state $s \in S^*$ and ground action a applicable to s where the outcome state s' of applying a to s is *not* a member of S^*.

Given several training instances, E^+, respectively E^-, are obtained simply as the union of E^+, respectively E^-, over all those instances.

To our knowledge, the only prior work taking a similar direction is that of de la Rosa et al. (de la Rosa et al. 2011). They generate all optimal plans using a depth-first branch and bound search with no duplicate pruning. A subset of these plans is then selected according to a ranking criterion, and the training data is generated from that subset. The latter step, i.e. the training data read off the solutions, is similar to ours, corresponding basically to a subset of S^* (we did not investigate yet whether such subset selection could be beneficial for our approach as well). The search step employed by de la Rosa et al. is unnecessarily ineffective as the same training data could be generated using our A*-based method, which does include duplicate pruning (a crucial advantage for search performance in many planning domains).

We will refer to the above as the

- **conservative** training data (i.e.based on all optimal plans), contrasted with what we call
- **greedy** training data.

The latter is oriented closely at the bulk of previous approaches: For the greedy data we take S^* to be the states along a single optimal plan only, otherwise applying the same definition of E^+ and E^-. In other words, in the greedy training data, (s, a) is "good" if the optimal plan used applies a to s, and is "bad" if the optimal plan passed through s but applied an action $a' \neq a$.

Note that above all actions in each state of S^* are included in either E^+ or E^-. We refer to this as the

- **all-operators** training data, contrasted with what we call
- **preferred-operators** training data.

In the latter, E^+ and E^- are defined as above, but are restricted to the subset of state/action pairs (s, a) where

$s \in S^*$, and ground action a is applicable to s *and* is a helpful action for s (according to the relaxed plan heuristic h^{FF} (Hoffmann and Nebel 2001)). Knowledge learned using this modified training data will be used only within searches that already employ this kind of action pruning: The idea is to focus the rule learning on those aspects missed by this native pruning rule.

Similarly to de la Rosa et al. (de la Rosa et al. 2011), in our implementation the training data generation is approximate in the sense that we use the relaxed plan heuristic h^{FF} as our heuristic h. h^{FF} is not in general admissible, but in practice it typically does not over-estimate (h^{FF} is usually close to h^+ (Hoffmann 2005)). Hence this configuration is viable in terms of runtime and scalability(strong admissible heuristics like LM-cut (Helmert and Domshlak 2009) are much slower than h^{FF}), and in terms of the typical quality of the training data generated.

There is an apparent mismatch between the distribution of states used to create the training data (only states on optimal plans) and the distribution of states that will be encountered during search (both optimal and sub-optimal states). Why then should we expect the rules to generalize properly when used in the context of search?

In general, there is no reason for that expectation, beyond the intuition that bad actions on optimal states will typically be bad also on sub-optimal ones sharing the relevant state features. It would certainly be worthwhile to try training on intelligently selected suboptimal states, similar in spirit to recent work on learning from imitation (Ross and Bagnell 2010). Note though that, as long as the pruning on the optimal states retains the optimal plans (which is what we are trying to achieve when learning from conservative data), even arbitrary pruning decisions at suboptimal states do not impact the availability of optimal plans in the search space.

Learning the Pruning Rules

Our objective is to learn some representation R, in a form that generalizes across instances of the same domain D, so that R covers a large fraction of bad examples in E^- without covering any of the good examples E^+. We want to use R for pruning during search, where on any search state s, an applicable ground action a will not be expanded in case (s, a) is covered by R. It remains to define *what kind of representation will underlie R, what it means to "cover" a state/action pair* (s, a)*, and how R will be learned.* We consider these in turn.

As previously advertized, we choose to represent R in the form of a set of *pruning rules*. Each rule $r[Y] \in R$ takes the form $r[Y] =$

$$\neg a[X] \Leftarrow l_1[X_1] \wedge \cdots \wedge l_n[X_n]$$

where $a[X]$ is an action schema from the domain D, $l_i[X_i]$ are first-order literals, and $Y = X \cup \bigcup_i X_i$ is the set of all variables occuring in the rule. In other words, we associate each action schema with conjunctive conditions identifying circumstances under which the schema is to be considered "bad" and should be pruned. As usual, we will sometimes refer to $\neg a[X]$ as the rule's *head* and to the condition $l_1[X_1] \wedge \cdots \wedge l_n[X_n]$ as its *body*.

We choose this simple representation for precisely that virtue: simplicity. Our approach is (relatively) simple to implement and use, and as we shall see can yield excellent results.

Given a domain instance with object set O, and a pruning rule $r[Y] \in R$, a *grounding* of $r[Y]$ takes the form $r =$

$$\neg a[o^1, \ldots, o^k] \Leftarrow l_1[o^1_1, \ldots, o^{k_1}_1] \wedge \cdots \wedge l_n[o^1_n, \ldots, o^{k_n}_n]$$

where $o^j = o^{j'}_{i'}$ whenever X and $X_{i'}$ share the same variable at position j respectively j', and $o^j_i = o^{j'}_{i'}$ whenever X_i and $X_{i'}$ share the same variable at position j respectively j'. We refer to such r as a *ground pruning rule*. In other words, ground pruning rules are obtained by substituting the variables of pruning rules with the objects of the domain instance under consideration.

Assume now a state s and a ground action a applicable to s. A ground pruning rule $r = [\neg a' \Leftarrow l_1 \wedge \cdots \wedge l_n]$ *covers* (s, a) if $a' = a$ and $s \models l_1 \wedge \cdots \wedge l_n$. A pruning rule $r[Y]$ covers (s, a) if there exists a grounding of $r[Y]$ that covers (s, a). A set R of pruning rules covers (s, a) if one of its member rules does.

With these definitions in hand, our learning task – learn a set of pruning rules R which covers as many bad examples in E^- as possible without covering any of the good examples E^+ – is a typical *inductive logic programming* (ILP) (Muggleton 1991)problem: We need to learn a set of logic programming rules that explains the observations as given by our training data examples. It is thus viable to use off-the-shelf tool support. We chose the well-known Aleph toolbox (Srinivasan 1999). (Exploring application-specific ILP algorithms for our setting is an open topic.)

In a nutshell, in our context, Aleph proceeds as follows:

1. If $E^- = \emptyset$, stop. Else, select an example $(s, a) \in E^-$.
2. Construct the "bottom clause", i.e., the most specific conjunction of literals that covers (s, a) and is within the language restrictions imposed. (See below for the restrictions we applied.)
3. Search for a subset of the bottom clause yielding a rule $r[Y]$ which covers (s, a), does not cover any example from E^+, and has maximal *score* (covers many examples from E^-).
4. Add $r[Y]$ to the rule set, and remove all examples from E^- covered by it. Goto 1.

Note that our form of ILP is simple in that there is no recursion. The rule heads (the action schemas) are from a fixed and known set separate from the predicates to be used in the rule bodies. Aleph offers support for this simply by separate lists of potential rule heads respectively potential body literals. These lists also allow experimentation with different language variants for the rule bodies:

- **Positive vs. mixed conditions:** We optionally restrict the rule conditions to contain only positive literals, referring to the respective variant as "positive" respectively "mixed". The intuition is that negative condition literals sometimes allow more concise representations of situations, but their presence also has the potential to unnecessarily blow up the search space for learning.

- **With vs. without inequality constraints:** As specified above, equal variables in a rule will always be instantiated with the same object. But, per default, different variables also may be instantiated with the same object. Aleph allows "$x \neq y$" body literals to prevent this from happening. Similarly to the above, such inequality constraints may sometimes help, but may also increase the difficulty of Aleph's search for good rules.

As the two options can be independently switched on or off, we have a total of four condition language variants. We will refer to these by **P**, **M**, $\mathbf{P}^{\neq}$, and $\mathbf{M}^{\neq}$ in the obvious manner.

We restrict negative condition literals, including literals of the form $x \neq y$, to use **bound variables** only: In any rule $r[Y]$ learned, whenever variable x occurs in a negative condition literal, then x must also occur in either a positive condition literal or in the rule's head.[2] Intuitively, this prevents negative literals from having excessive coverage by instantiating an unbound variable with all values that do *not* occur in a state (e.g., "$\neg at[x, y]$" collects all but one city y for every object x). Note that, in our context, head variables are considered to be bound as their instantiation will come from the ground action a whose "bad" or "good" nature we will be checking.

Aleph furthermore allows various forms of fine-grained control over its search algorithm. We used the default setting for all except two parameters. First, the rule length bound restricts the search space to conditions with at most L literals. We empirically found this parameter to be of paramount importance for the runtime performance of learning. Furthermore, we found that $L = 6$ was an almost universally good "magic" setting of this parameter in our context: $L > 6$ rarely ever lead to better-performing rules, i.e., to rules with more pruning power than those learned for $L = 6$; and $L < 6$ very frequently lead to much worse-performing rules. We thus fixed $L = 6$, and use this setting throughout the experiments reported. Second, minimum coverage restricts the search space to rules that cover at least C examples from E^-. We did not run extensive experiments examining this parameter, and fixed it to $C = 2$ to allow for a maximally fine-grained representation of the training examples (refraining only from inserting a rule for the sake of a single state/action pair).

Using the Pruning Rules

Given a domain instance Π, a state s during forward search on Π, and an action a applicable to s, we need to test whether R covers (s, a). If the answer is "no", proceed as usual; if the answer is "yes", prune a, i.e., do not generate the resulting state.

The issue here is computational efficiency: We have to pose the question "does R cover (s, a)?" not only for every state s during a combinatorial search, but even for every action a applicable to s. So it is of paramount importance for that test to be fast. Indeed, we must avoid the infamous utility problem (Minton 1990), identified in early work on learning for planning, where the overhead of evaluating the learned knowledge would often dominate the potential gains.

Unfortunately, the problem underlying the test is **NP**-complete: For rule heads with no variables, and rule bodies with only positive literals, we are facing the well-known problem of evaluating a Boolean conjunctive query (the rule body) against a database (the state) (Chandra and Merlin 1977). More precisely, the problem is **NP**-complete when considering arbitrary-size rule bodies ("combined complexity" in database theory). When fixing the rule body size, as we do in our work (remember that $L = 6$), the problem becomes polynomial-time solvable ("data complexity"), i.e., exponential in the fixed bound. For our bound 6, this is of course still way too costly with a naïve solution enumerating all rule groundings. We employ backtracking in the space of partial groundings, using unification to generate only partial groundings that match the state and ground action in question. In particular, a key advantage in practice is that, typically, many of the rule variables occur in the head and will thus be fixed by the ground action a already, substantially narrowing down the search space.

For the sake of clarity, let us fill in a few details. Say that s is a state, $a[o_1, \ldots, o_k]$ is a ground action, and $\neg a[x_1, \ldots, x_k] \Leftarrow l_1[X_1] \wedge \cdots \wedge l_n[X_n]$ is a pruning rule for the respective action schema. We view the positive respectively negative body literals as sets of atoms, denoted L_P respectively L_N. With $\alpha := \{(x_1, o_1), \ldots, (x_k, o_k)\}$, we set $L_P := \alpha(L_P)$ and $L_N := \alpha(L_N)$, i.e., we apply the partial assignment dictated by the ground action to every atom. We then call the following recursive procedure:

if $L_P \neq \emptyset$ **then**
  select $l \in L_P$
  for all $q \in s$ unifiable with l via partial assignment β **do**
    if recursive call on $\beta(L_P \setminus \{l\})$ and $\beta(L_N)$ succeeds **then**
      succeed
    endif
  endfor
  fail
else /* $L_P = \emptyset$ */
  if $L_N \cap s = \emptyset$ **then** succeed **else** fail **endif**
endif

The algorithm iteratively processes the atoms in L_P. When we reach L_N, i.e., when all positive body literals have already been processed, all variables must have been instantiated because negative literals use bound variables only (cf. previous section). So the negative part of the condition is now a set of ground atoms and can be tested simply in terms of its intersection with the state s.

We use two simple heuristics to improve runtime. Within each rule condition, we order predicates with higher arity up front so that many variables will be instantiated quickly. Across rules, we dynamically adapt the order of evaluation. For each rule r we maintain its "success count", i.e., the number of times r fired (pruned out an action). Whenever r fires, we compare its success count with that of the preceding rule r'; if the count for r is higher, r and r' get switched.

[2] We implemented this restriction via the "input/output" tags Aleph allows in the lists of potential rule heads and body literals. We did not use these tags for any other purpose than the one described, so we omit a description of their more general syntax and semantics.

This simple operation takes constant time but can be quite effective.

Experiments

We use the benchmark domains from the learning track of IPC'11. All experiments were run on a cluster of Intel E5-2660 machines running at 2.20 GHz. We limited runtime for training data generation to 15 minutes (per task), and for rule learning to 30 minutes (per domain, configuration, and action schema). To obtain the training data, we manually played with the generator parameters to find maximally large instances for which the learning process was feasible within these limits. We produced 8–20 training instances per domain and training data variant (i.e., **conservative** vs. **greedy**). Handling sufficiently large training instances turned out to be a challenge in Gripper, Rovers, Satellite and TPP. For example, in Gripper the biggest training instances contain 3 grippers, 3 rooms and 3 objects; for Rovers, our training instances either have a single rover, or only few waypoints/objectives. We ran all four condition language variants – **P**, **M**, $\mathbf{P}^{\neq}$, and $\mathbf{M}^{\neq}$ – on the same training data. We show data only for the language variants with inequality constraints, i.e., for $\mathbf{P}^{\neq}$ and $\mathbf{M}^{\neq}$), as these generally performed better.

	all-operators								preferred-operators							
	Conservative				Greedy				Conservative				Greedy			
	$P^{\neq}$		$M^{\neq}$		$P^{\neq}$		$M^{\neq}$		$P^{\neq}$		$M^{\neq}$		$P^{\neq}$		$M^{\neq}$	
	#	L	#	L	#	L	#	L	#	L	#	L	#	L	#	L
Barman	14	2.7	5	2.4	17	2.1	17	1.8	7	2.9	5	2.4	8	2.1	8	1.5
Blocksworld	29	4.4	0	—	61	3.8	23	2.7	28	4.3	0	—	46	3.7	21	2.7
Depots	2	4.5	1	4	16	3.3	10	2.8	4	4.8	2	4	12	3.4	9	3.1
Gripper	27	4.9	1	4	26	4.1	23	3.2	20	4.8	9	4	17	4.2	11	3.4
Parking	92	3.4	51	2.8	39	2.6	31	2.2	71	3.3	48	2.8	20	2.6	18	2.1
Rover	30	2.2	18	1.8	45	1.8	36	1.6	3	2	3	2	14	1.7	16	1.7
Satellite	27	3.2	26	3	25	2.6	22	2.2	12	3.4	12	3	9	3	9	2.6
Spanner	1	3	1	3	1	3	1	3	1	3	1	3	1	3	1	3
TPP	13	2.5	10	2.4	18	2.6	21	2.6	6	2.8	5	2.8	11	2.7	12	2.8

Table 1: Statistics regarding the rule sets learned. "#": number of rules; "L": average rule length (number of rule body literals).

Table 1 shows statistics about the learned rule sets. One clear observation is that fewer rules tend to be learned when using preferred-operators training data. This makes sense simply as that training data is smaller. A look at rule length shows that rules tend to be short except in a few cases. A notable extreme behavior occurs in Spanner, where we learn a single three-literal pruning rule, essentially instructing the planner to not leave the room without taking along all the spanners. As it turns out, this is enough to render the benchmark trivial for heuristic search planners. We get back to this below.

We implemented parsers for our pruning rules, and usage during search, in FF (Hoffmann and Nebel 2001) and Fast Downward (FD) (Helmert 2006). We report data only for FD; that for FF is qualitatively similar. To evaluate the effect of our rules when using/not using the native pruning, as "base planners" we run FD with h^{FF} in single-queue lazy greedy best-first search (**FD1**), respectively in the same configuration but with a second open list for states resulting from preferred operators (**FD2**). To evaluate the effect of our rules on a representation of the state of the art in runtime, we run (the FD implementation of) the first search iteration of **LAMA** (Richter and Westphal 2010), which also is a dual-queue configuration where one open list does, and one does not, use the native pruning. As we noticed that, sometimes, FD's *boosting* (giving a higher preference to the preferred-operators queue), is detrimental to performance, we also experimented with configurations not using such boosting.

In both dual-queue configurations, we apply our learned pruning rules only to the preferred-operators queue, keeping the other "complete" queue intact. The **preferred-operators** training data is used in these cases. For FD1, where we apply the rules to a single queue not using preferred operators, we use the **all-operators** training data.

For the experiments on test instances, we used runtime (memory) limits of 30 minutes (4 GB). We used the original test instances from IPC'11 for all domains except Gripper and Depots, where LAMA was unable to solve more than a single instance (with or without our rules). We generated smaller test instances using the generators provided, using about half as many crates than the IPC'11 test instances in Depots, and cutting all size parameters by about half in Gripper.

Table 2 gives a summary of the results. Considering the top parts of the tables (FD-default with boosting where applicable), for 4 out of 9 domains with FD1, for 4 domains with FD2, and for 4 domains with LAMA, the best coverage is obtained by one of our rule-pruning configurations. Many of these improvements are dramatic: 2 domains (FD1: Barman and Spanner), 3 domains (FD2: Barman, Blocksworld, and Parking), respectively 1 domain (LAMA: Barman). When switching the boosting off in FD2 and LAMA, a further dramatic improvement occurs in Satellite (note also that, overall, the baselines suffer a lot more from the lack of boosting than those configurations using our pruning rules). Altogether, our pruning rules help in different ways for different base planners, and can yield dramatic improvements in 5 out of the 9 IPC'11 domains.

The Achilles heel lies in the word "can" here: While there are many great results, they are spread out across the different configurations. We did not find a single configuration that combines these advantages. Furthermore, on the two domains where our pruning techniques are detrimental – Rovers and TPP – we lose dramatically, so that, for the default (boosted) configurations of FD2 and LAMA, in overall coverage we end up doing substantially worse.

In other words, our pruning techniques (a) have high variance and are sensitive to small configuration details, and (b) often are highly complementary to standard heuristic search planning techniques. Canonical remedies for this are *auto-tuning*, learning a configuration per-domain, and/or *portfolios*, employing combinations of configurations. Indeed, from that perspective, both (a) and (b) could be good news, especially as other satisficing heuristic search planning techniques have a tendency to be strong in similar domains.

A comprehensive investigation of auto-tuning and port-

	FD1 (h^{FF})									FD2 (dual queue h^{FF} + preferred operators)																			
	base pl.	Cons $P^{\neq}$		Cons $M^{\neq}$		Greedy $P^{\neq}$		Greedy $M^{\neq}$		base planner			Cons $P^{\neq}$				Cons $M^{\neq}$				Greedy $P^{\neq}$				Greedy $M^{\neq}$			
	C	C	¬S	C	¬S	C	¬S	C	¬S	C	T	E	C	T	E	RT	C	T	E	RT	C	T	E	RT	C	T	E	RT
Barman (30)	0	**27**	0	0	0	0	0	0	0	14	609.6	271972	13	12.9	28.9	63%	23	17.1	39.2	57%	**27**	1.0	1.4	47%	21	1.6	2.3	45%
Blocksworld (30)	0	0	0	0	0	0	18	**1**	0	19	37.4	19916	18	0.6	1.0	54%	19	1.2	1.0	0%	1	0.0	0.0	85%	**27**	3.6	3.4	17%
Depots (30)	**13**	**13**	0	**13**	0	**13**	12	**13**	11	18	48.2	111266	18	0.7	1.1	33%	18	0.8	1.0	20%	**23**	1.6	2.1	18%	21	3.2	3.5	18%
Gripper (30)	13	0	0	**15**	0	0	23	0	20	**29**	3.9	2956	19	0.0	0.1	95%	26	0.0	0.1	90%	19	0.0	0.3	96%	17	0.0	0.2	84%
Parking (30)	1	3	0	**4**	0	0	30	0	30	7	642.5	16961	8	0.5	0.5	6%	6	0.8	0.8	5%	**25**	35.5	15.2	2%	14	15.3	11.8	1%
Rover (30)	0	0	29	0	3	0	1	0	0	**30**	41.9	22682	11	0.0	0.1	91%	12	0.0	0.1	91%	3	0.0	0.1	94%	13	0.0	0.1	83%
Satellite (30)	0	0	0	0	0	0	0	0	1	**3**	752.3	51741	0	—	—	—	0	—	—	—	2	0.5	0.7	54%	0	—	—	—
Spanner (30)	0	**30**	0	**30**	0	**30**	0	**30**	0	0	—	—	0	—	—	—	0	—	—	—	0	—	—	—	0	—	—	—
TPP (30)	0	0	0	0	0	0	0	0	0	**29**	232.5	13057	0	—	—	—	0	—	—	—	0	—	—	—	0	—	—	—
$\sum$ (270)	27	**73**	29	62	3	43	84	44	62	**149**			87				104				100				113			
										no FD preferred operators boosting																		
Satellite (30)										2	1009,0	68253	0	—	—	—	12	1,1	11,1	92%	0	—	—	—	**16**	3,4	23,2	84%
$\sum$ (270)										53			50				65				**80**				72			

	LAMA (first iteration)																				AutoTune	Portfolios	
	base planner			Cons $P^{\neq}$				Cons $M^{\neq}$				Greedy $P^{\neq}$				Greedy $M^{\neq}$						Seq-Uniform	Seq-Hand
	C	T	E	C	T	E	RT	C	T	E	RT	C	T	E	RT	C	T	E	RT		C	C	C
Barman (30)	7	648.1	151749	**30**	23.8	51.1	53%	**30**	5.0	9.7	44%	22	0.8	1.3	38%	21	0.8	1.3	36%		23	30	30
Blocksworld (30)	27	63.5	13093	24	0.7	1.0	45%	27	1.3	1.0	0%	6	0.3	0.6	55%	**30**	14.2	13.5	19%		27	27	28
Depots (30)	23	43.2	37299	22	0.9	1.2	35%	25	0.9	1.0	25%	**26**	7.0	9.9	22%	25	15.3	17.3	22%		23	24	25
Gripper (30)	**29**	6.4	3122	9	0.0	0.0	85%	16	0.0	0.0	87%	21	0.0	0.4	93%	24	0.0	0.2	76%		29	28	29
Parking (30)	26	699.3	3669	10	0.4	0.2	7%	16	1.4	1.2	7%	**29**	10.2	5.5	2%	28	11.3	6.1	2%		28	30	30
Rover (30)	**29**	211.2	28899	9	0.1	0.2	78%	10	0.1	0.2	80%	0	—	—	—	7	0.1	0.1	65%		30	29	29
Satellite (30)	**4**	986.7	34739	0	—	—	—	0	—	—	—	0	—	—	—	0	—	—	—		3	13	16
Spanner (30)	0	—	—	0	—	—	—	0	—	—	—	0	—	—	—	0	—	—	—		30	30	30
TPP (30)	**20**	360.5	13262	0	—	—	—	0	—	—	—	0	—	—	—	0	—	—	—		29	18	18
$\sum$ (270)	**165**			104				124				104				135					222	229	235
	no FD preferred operators boosting																						
Satellite (30)	3	819,7	32301	0	—	—	—	22	4,1	26,4	85%	1	0,4	0,8	73%	**23**	4,2	14,0	78%				
$\sum$ (270)	84			80				106				95				**125**							

Table 2: Performance overview. "C": coverage; "¬S": all solutions pruned out (search space exhausted); "T" search time and "E" number of expanded states (median for base planner, median ratio "base-planner/rules-planner" for planners using our rules); "RT": median percentage of total time spent evaluating rules. For each base planner, best coverage results are highlighted in boldface. By default, FD's preferred operators queue in FD2 and LAMA is boosted; we show partial results switching that boosting off. For explanation of the "AutoTune" and "Portfolios" data, see text.

folios is beyond the scope of this paper, but to give a first impression we report preliminary data in Table 2 (bottom right), based on the configuration space $\{$FD1, FD2, LAMA$\} \times \{\mathbf{P}, \mathbf{M}, \mathbf{P}^{\neq}, \mathbf{M}^{\neq}\} \times \{$boost, no-boost$\}$. For "AutoTune", we created medium-size training data (in between training data and test data size) for each domain, and selected the configuration minimizing summed-up search time on that data. For "Portfolios", we created sequential portfolios of four configurations, namely FD1 Cons $P^{\neq}$, FD2 base planner (boosted), LAMA Cons $P^{\neq}$ (boosted), and LAMA Greedy $M^{\neq}$ not boosted. For "Seq-Uniform" each of these gets 1/4 of the runtime (i.e., 450 seconds); for "Seq-Hand", we played with the runtime assignments a bit, ending up with 30, 490, 590, and 690 seconds respectively. Despite the comparatively little effort invested, these auto-tuned and portfolio planners perform vastly better than any of the components, including LAMA.

Regarding rule content and its effect on search, the most striking, and easiest to analyze, example is Spanner. Failing to take a sufficient number of spanners to tighten all nuts is the major source of search with delete relaxation heuristics. Our single learned rule contains sufficient knowledge to get rid of that, enabling FD1 to solve every instance in a few seconds. This does not work for FD2 and LAMA because their preferred operators prune actions taking spanners (the relaxed plan makes do with a single one), so that the combined pruning (preferred operators *and* our rule) removes the plan. We made an attempt to remedy this by pruning with our rules on one queue and with preferred operators on the other, but this did not work either (presumably because, making initial progress on the heuristic value, the preferred operators queue gets boosted). The simpler and more successful option is to use a portfolio, cf. above.

Regarding conservative vs. greedy training data, consider FD1. As that search does not employ a complete "back-up" search queue, if our pruning is too strict then no solution can be found. The "¬S" columns vividly illustrate the risk incurred. Note that, in Parking, while the greedy rules prune out all solutions on FD1 (the same happens when training them on the preferred-operators training data), they yield dramatic improvements for FD2, and significant improvements for LAMA. It is not clear to us what causes this.

Regarding the overhead for rule evaluation, the "RT" columns for LAMA show that this can be critical in Grip-

per, Rovers, and Satellite. Comparing this to Table 1 (right half), we do see that Gripper tends to have long rules, which complies with our observation. On the other hand, for example, Parking has more and longer rules than Rovers, but its evaluation overhead is much smaller. Further research is needed to better understand these phenomena.

For TPP, where none of the configurations using our rules can solve anything and so Table 1 does not provide any indication what the problem is, observations on smaller examples suggest that solutions otherwise found quickly are pruned: the FD1 search space became larger when switching on the rule usage.

Conclusion

We realized a straightforward idea – using off-the-shelf ILP for learning conjunctive pruning rules acting like preferred operators in heuristic search planning – that hadn't been tried yet. The results are quite good, with substantial to dramatic improvements across several domains, yielding high potential for use in portfolios. Together with the simplicity of the approach, this strongly suggests that further research on the matter may be worthwhile. The most immediate open lines in our view are to (a) systematically explore the design of complementary configurations and portfolios thereof, as well as (b) understanding the behavior of the technique in more detail.

References

Bacchus, F., and Kabanza, F. 2000. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence* 116(1-2):123–191.

Botea, A.; Enzenberger, M.; Müller, M.; and Schaeffer, J. 2005. Macro-FF: Improving AI planning with automatically learned macro-operators. *Journal of Artificial Intelligence Research* 24:581–621.

Celorrio, S. J.; de la Rosa, T.; Fernández, S.; Fernández, F.; and Borrajo, D. 2012. A review of machine learning for automated planning. *Knowledge Engineering Review* 27(4):433–467.

Cenamor, I.; de la Rosa, T.; and Fernández, F. 2013. Learning predictive models to configure planning portfolios. In *Proceedings of the ICAPS Workshop on Planning and Learning (PAL'13)*.

Chandra, A. K., and Merlin, P. M. 1977. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of the 9th ACM Symposium on the Theory of Computation*, 77–90.

Coles, A., and Smith, A. 2007. Marvin: A heuristic search planner with online macro-action learning. *Journal of Artificial Intelligence Research* 28:119–156.

de la Rosa, T., and McIlraith, S. 2011. Learning domain control knowledge for TLPlan and beyond. In *Proceedings of the ICAPS Workshop on Planning and Learning (PAL'11)*.

de la Rosa, T.; Celorrio, S. J.; Fuentetaja, R.; and Borrajo, D. 2011. Scaling up heuristic planning with relational decision trees. *Journal of Artificial Intelligence Research* 40:767–813.

Domshlak, C.; Karpas, E.; and Markovitch, S. 2012. Online speedup learning for optimal planning. *Journal of Artificial Intelligence Research* 44:709–755.

Fern, A.; Yoon, S. W.; and Givan, R. 2006. Approximate policy iteration with a policy language bias: Solving relational Markov decision processes. *Journal of Artificial Intelligence Research* 25:75–118.

Gretton, C. 2007. Gradient-based relational reinforcement-learning of temporally extended policies. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling (ICAPS'07)*.

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

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

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

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

Huang, Y.-C.; Selman, B.; and Kautz, H. A. 2000. Learning declarative control rules for constraint-based planning. In Langley, P., ed., *Proceedings of the 17th International Conference on Machine Learning (ICML-00)*, 415–422. Stanford University, Stanford, USA: Morgan Kaufmann.

Khardon, R. 1999. Learning action strategies for planning domains. *Artificial Intelligence* 113(1-2):125–148.

Kvarnström, J., and Magnusson, M. 2003. TALplanner in the third international planning competition: Extensions and control rules. *Journal of Artificial Intelligence Research* 20:343–377.

Martin, M., and Geffner, H. 2000. Learning generalized policies in planning using concept languages. In Cohn, A.; Giunchiglia, F.; and Selman, B., eds., *Principles of Knowledge Representation and Reasoning: Proceedings of the 7th International Conference (KR-00)*, 667–677. Breckenridge, CO, USA: Morgan Kaufmann.

Minton, S. 1990. Quantitative results concerning the utility of explanation-based learning. *Artificial Intelligence* 42(2):363–391.

Muggleton, S. 1991. Inductive logic programming. *New Generation Computing* 8(4):295–318.

Newton, M. A. H.; Levine, J.; Fox, M.; and Long, D. 2007. Learning macro-actions for arbitrary planners and domains. In Boddy, M.; Fox, M.; and Thiebaux, S., eds., *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS'07)*, 256–263. Providence, Rhode Island, USA: Morgan Kaufmann.

Núñez, S.; Borrajo, D.; and López, C. L. 2012. Performance analysis of planning portfolios. In Borrajo, D.; Felner, A.; Korf, R.; Likhachev, M.; Linares López, C.; Ruml, W.; and

Sturtevant, N., eds., *Proceedings of the 5th Annual Symposium on Combinatorial Search (SOCS'12)*. AAAI Press.

Richter, S., and Helmert, M. 2009. Preferred operators and deferred evaluation in satisficing planning. In Gerevini, A.; Howe, A.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*, 273–280. AAAI Press.

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

Roberts, M., and Howe, A. 2009. Learning from planner performance. *Artificial Intelligence* 173(5-6):536–561.

Ross, S., and Bagnell, D. 2010. Efficient reductions for imitation learning. In *International Conference on Artificial Intelligence and Statistics*, 661–668.

Seipp, J.; Braun, M.; Garimort, J.; and Helmert, M. 2012. Learning portfolios of automatically tuned planners. In Bonet, B.; McCluskey, L.; Silva, J. R.; and Williams, B., eds., *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*. AAAI Press.

Srinivasan, A. 1999. The Aleph manual. Available from http://www.cs.ox.ac.uk/activities/machlearn/Aleph/.

Srivastava, S.; Immerman, N.; and Zilberstein, S. 2012. Applicability conditions for plans with loops: Computability results and algorithms. *Artificial Intelligence* 191-192:1–19.

Virseda, J.; Borrajo, D.; and Alcázar, V. 2013. Learning heuristic functions for cost-based planning. In *Proceedings of the ICAPS Workshop on Planning and Learning (PAL'13)*.

Walsh, T. J., and Littman, M. L. 2008. Efficient learning of action schemas and web-service descriptions. In Fox, D., and Gomes, C., eds., *Proceedings of the 23rd National Conference of the American Association for Artificial Intelligence (AAAI-08)*, 714–719. Chicago, Illinois, USA: AAAI Press.

Xu, Y.; Fern, A.; and Yoon, S. 2009. Learning linear ranking functions for beam search with application to planning. *The Journal of Machine Learning Research* 10:1571–1610.

Xu, Y.; Fern, A.; and Yoon, S. W. 2010. Iterative learning of weighted rule sets for greedy search. In Brafman, R. I.; Geffner, H.; Hoffmann, J.; and Kautz, H. A., eds., *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS'10)*, 201–208. AAAI Press.

Yoon, S. W.; Fern, A.; and Givan, R. 2002. Inductive policy selection for first-order MDPs. In Darwiche, A., and Friedman, N., eds., *Proceedings of the 18th International Conference on Uncertainty in AI (UAI-02)*, 568–576. Edmonton, Alberta, Canada: Morgan Kaufmann.

Yoon, S. W.; Fern, A.; and Givan, R. 2008. Learning control knowledge for forward search planning. *Journal of Machine Learning Research* 9:683–718.

What Does it Take to Render $h^+(\Pi^C)$ Perfect?

Jörg Hoffmann and **Marcel Steinmetz**
Saarland University
Saarbrücken, Germany
hoffmann@cs.uni-saarland.de, s9mrstei@stud.uni-saarland.de

Patrik Haslum
The Australian National University & NICTA
Canberra, Australia
patrik.haslum@anu.edu.au

Abstract

It is well-known that $h^+(\Pi^C)$ is perfect in the limit, i. e., we can always choose C so that $h^+(\Pi^C) = h^*$. But the proof is trivial (select C as the set of all conjunctions), and completely ignores the actual power of $h^+(\Pi^C)$, basically pretending that h^+ is the same as h^1. It is thus interesting to ask: *Can we characterize the power of $h^+(\Pi^C)$ more accurately? How large does C have to be, under which circumstances?*

We present first results towards answering these questions. We introduce a "direct" characterization of $h^+(\Pi^C)$, in terms of equations, not employing a compilation step. We identify a first tractable fragment (similar to fork causal graphs) where size-2 conjunctions suffice to render $h^+(\Pi^C)$ perfect. We present results comparing $h^+(\Pi^C)$ to alternative partial delete relaxation methods (red-black planning and fluent merging). We finally present a number of wild speculations as to what might be interesting to investigate in the future.

Disclaimer: We are enthusiastic about the research direction, but our work as yet raises far more questions than answers. We think that HSDIP is a great forum to discuss this big riddle, and we hope that other researchers may feel compelled to look at it.

Introduction

Haslum's (2009) work on compiling fact conjunctions into the planning task, allowing to simulate h^m via h^1, led a few years later to a partial delete relaxation method able to interpolate all the way between h^+ and h^*: The Π^C compilation (Haslum 2012) allows to select any subset C of fact conjunctions, and outputs a compiled task Π^C so that $h^+(\Pi^C)$ is admissible and perfect in the limit, i. e., we can always choose C so that $h^+(\Pi^C) = h^*$.

The size of Π^C is worst-case exponential in $|C|$, which has been solved via a slightly weaker compilation Π^C_{ce} (Keyder, Hoffmann, and Haslum 2012) exploiting conditional effects, but for the sake of simplicity we abstract from that issue here and consider only $h^+(\Pi^C)$. Our primary objective is to *scratch the itch that results from reading the proof of $h^+(\Pi^C)$ convergence*: The proof is derived from the inequalities (a) $h^m \leq h^1(\Pi^C)$ when C contains all m-tuples, and (b) $h^1(\Pi^C) \leq h^+(\Pi^C)$. In other words, $h^+(\Pi^C)$ convergence is inherited from that of h^m which is completely impractical (set m to the total number of facts). The proof completely ignores the actual added power of $h^+(\Pi^C)$, namely (a) being able to choose C freely, as well as (b) *the advantage of h^+ over h^1!*

Another way to say this is that our theory so far is completely disconnected from practice, where of course $h^+(\Pi^C)$ with C being all fact pairs will in most cases be a *much* better heuristic than h^2. Can we reconcile the theory with practice?[1] Can we characterize more accurately the circumstances under which $h^+(\Pi^C)$ becomes perfect? When does that require C to be exponentially large, and when is polynomial-size C enough? Can we exploit such insights to choose C in a targeted manner?

We believe that these are interesting research questions. We are not so sure about the significance of our answers so far. Certainly, we are nowhere near answering the last question, i. e., it is unclear how (and whether at all) our results so far can be made useful in practice. Our hope is that other HSDIP researchers will find our questions and partial answers inspiring, leading to interesting discussions and, eventually, better progress on this subject.

After preliminaries (notations, Π^C compilation), we make a few simple observations about the size of C depending on the value of h^*. We then introduce a "direct" characterization of $h^+(\Pi^C)$, in terms of equations, that does not need to go via a compilation step (as a side effect, this also yields a somewhat novel view on h^+). Towards an analysis of "tractable fragments", i. e., planning sub-classes in which polynomial-size C suffices to render $h^+(\Pi^C)$ perfect, we introduce a first such fragment similar to fork causal graphs, where size-2 conjunctions suffice. We present results comparing $h^+(\Pi^C)$ to alternative partial delete relaxation methods, namely red-black planning (Katz, Hoffmann, and Domshlak 2013b; 2013a; Katz and Hoffmann 2013) and fluent merging (van den Briel, Kambhampati, and Vossen 2007; Seipp and Helmert 2011). We close the paper ("conclusion") with a number of wild speculations as to what might be interesting to investigate in the future.

[1] To be fair, it should be said that, while the above proof is given by both Haslum (2012) and Keyder et al. (2012), Haslum also gives an alternative proof via convergence of iterative relaxed plan refinement. The latter proof, however, involves excluding all flawed relaxed plans one-by-one, which does not seem to be any more directly illuminating regarding the practical power of $h^+(\Pi^C)$.

Preliminaries

In difference to prior works on Π^C, we use an FDR framework. Planning tasks are tuples $\Pi = (V, A, I, G)$ of variables, actions, initial state, and goal, each action a being a pair $(pre(a), \mathit{eff}(a))$ as usual. We consider uniform costs only (i. e., all action costs are 1). We refer to variable/value pairs as facts, and we perceive (partial) variable assignments as sets of facts. The set of all facts in a planning task is denoted F. We will from now on assume this setup tacitly, i. e., we won't repeat it in formal claims etc.

We say that a set X of facts is *consistent* if there does not exist a variable v so that X contains more than one value for v. Otherwise, we say that X is *contradictory*.

When we talk about heuristic functions h, we mean their value $h(I)$ in the initial state (i. e., for the moment we do not consider rendering$h^+(\Pi^C)$ perfect across all states). By $h(\Pi')$, we denote a heuristic function for Π whose value is given by applying h in a modified task Π'. It is sometimes of advantage to make explicit that h is a heuristic computed on Π itself; we will denote that by $h(\Pi)$.

The delete relaxation in FDR, and thus h^+ in our setup, is defined by interpreting states as fact sets allowed to be contradictory, and where applying action a to state s yields the outcome state $s \cup \mathit{eff}(a)$; the initial state is the same as before. Intuitively, this just means that we are interpreting the effect list $\mathit{eff}(a)$ exactly like the add lists in STRIPS.

The Π^C compilation and its relatives are based on introducing π-*fluents* of the form π_c, each of which represents a conjunction c of facts. In the context of FDR, π_c is a Boolean variable; we will treat it like a STRIPS fact, e. g., we write $\pi_c \in s$ if π_c is true in s, and $\pi_c \notin s$ otherwise. We identify conjunctions with fact sets. For fact sets X, we use the shorthand $X^C = X \cup \{\pi_c \mid c \in C \wedge c \subseteq X\}$. In other words, X^C consists of the set of facts X itself, together with all facts π_c representing conjunctions $c \in C$ such that $c \subseteq X$. With this, Π^C can be defined as follows:

Definition 1 (The Π^C compilation) *Given a set C of conjunctions, Π^C is the planning task (V^C, A^C, I^C, G^C), where $V^C = V \cup \{\pi_c \mid c \in C\}$, and A^C contains an action $a^{C'}$ for every pair $a \in A$, $C' \subseteq C$ such that*

- *for all $c' \in C'$, $\mathit{eff}(a) \cap c' \neq \emptyset$, and $\mathit{eff}(a) \cup c'$ is consistent.*

Here, $a^{C'}$ is given by

- $pre(a^{C'}) = (pre(a) \cup \bigcup_{c' \in C'} (c' \setminus \mathit{eff}(a)))^C$, *and*
- $\mathit{eff}(a^{C'}) = \mathit{eff}(a) \cup \{\pi_{c'} \mid c' \in C'\}$.

This definition, apart from using FDR instead of STRIPS, diverges from Haslum's (2012) in three ways. We do not demand C' to be "downward closed", i. e., to contain all c' subsumed by C'; we do not automatically include $\pi_{c'}$ facts relying on non-deleted preconditions; and we do not include any delete effects. None of these changes have any consequences for the results we present. The first just introduces some superfluous actions, the second change means that we need to include these $\pi_{c'}$ facts explicitly into C', and the third change is made as such effects are irrelevant to h^+ which is our exclusive focus here.

We denote by $C^m := \{c \subseteq F \mid |c| \leq m\}$ the set of all size-$\leq m$ conjunctions. We denote Π^C with $C = C^m$ by Π^{Cm}. We will often consider Π^{Cm} only, abstracting from the ability of Π^C to choose an arbitary C. The underlying intuition/hypothesis is that, in most cases, this abstraction level will suffice to determine the desired distinction between polynomial-size C and exponentially large C.

We will sometimes employ regression-based characterizations of h^* and h^+. The *regression of fact set g over action a*, $R(g, a)$, is defined if $\mathit{eff}(a) \cap g \neq \emptyset$ and $\mathit{eff}(a) \cup g$ is consistent.[2] If $R(g, a)$ is defined, then $R(g, a) = (g \setminus \mathit{eff}(a)) \cup pre(a)$; otherwise, we write $R(g, a) = \bot$.

Obviously, $h^* = h(G)$ where $h(g)$, for a set g of facts, is the function that satisfies $h(g) =$

$$\begin{cases} 0 & g \subseteq I \\ 1 + \min_{a \in A, R(g,a) \neq \bot} h(R(g, a)) & \text{otherwise} \end{cases} \quad (1)$$

Similarly, $h^+ = h(G)$ where $h(g)$, for a set g of facts, is the function that satisfies $h(g) =$

$$\begin{cases} 0 & g \subseteq I \\ 1 + \min_{a \in A, \mathit{eff}(a) \cap g \neq \emptyset} & \\ \quad h((g \setminus \mathit{eff}(a)) \cup pre(a)) & \text{otherwise} \end{cases} \quad (2)$$

Under the delete relaxation, a sub-goal g can be achieved through action a iff part of it is achieved by a's effect, regardless of any contradictions that may be present.

Remember finally that h^m is defined as $h^m = h(G)$ where $h(g)$, for a set g of facts, is the function that satisfies $h(g) =$

$$\begin{cases} 0 & g \subseteq I \\ 1 + \min_{a \in A, R(g,a) \neq \bot} h^m(R(g, a)) & |g| \leq m \\ \max_{g' \subseteq g, |g'| \leq m} h^m(g') & \text{otherwise} \end{cases} \quad (3)$$

The Size of C vs. the Value of h^*

A possible starting point for thinking about the size of C is comparing it to the value of h^*. A trivial observation is immediately made:

Proposition 1 *If $h^+(\Pi^C) < \infty$, then $h^+(\Pi^C) - h^+(\Pi) \leq |C|$.*

This holds simply because a relaxed plan needs to achieve every fact (including π-fluents) at most once. We get:

Proposition 2 *If, in a planning task family $\{\Pi_n\}$ whose size relates polynomially to n, h^* grows exponentially in n, then so must C in order to render $h^+(\Pi^C)$ perfect.*

Denoting by F_n the set of facts in Π_n, with Proposition 1 we have $|C| >= h^*(\Pi_n) - F_n$, showing this claim.

Proposition 2 opens the question whether there exist cases with polynomial h^*, but where super-polynomial growth of C is needed nevertheless. The answer is a qualified "yes":

[2] It is sometimes required also that $(g \setminus \mathit{eff}(a)) \cup pre(a)$ is consistent. The two definitions are equivalent as contradictory subgoals will be unreachable anyhow. We use the simpler definition as it is closer to h^+ and its relatives.

Proposition 3 *There exist planning task families $\{\Pi_n\}$ whose size relates polynomially to n, where h^* grows polynomially in n for solvable tasks, but where (unless **P=NP**) C must grow super-polynomially in n in order to render $h^+(\Pi^C)$ perfect.*

Proof: Simply encode SAT into a planning task whose size relates polynomially to the number n of clauses, and where a plan consists of choosing a value for each variable, then evaluating that all clauses are satisfied. Then h^* grows polynomially in n for solvable tasks. Assume that polynomial-sized C suffices to render $h^+(\Pi^C)$ perfect. As relaxed plan existence is equivalent to $h^1 < \infty$, we could then in polynomial time decide whether or not $h^* = h^+(\Pi^C) < \infty$, yielding **P=NP**. ∎

Proposition 3 is only a "qualified" yes because its setup is not fair: Whereas we require h^* to grow polynomially only on solvable tasks (ignoring the ∞ cases), we require $h^+(\Pi^C)$ to be perfect *everywhere*, including the ∞ cases. For solvable SAT instances, $h^+(\Pi^C)$ might very well get perfect with small C already – or, at least, the current proof makes no statement about that.

Open Question 1 *Do there exist families of solvable tasks $\{\Pi_n\}$ whose size relates polynomially to n, where h^* grows polynomially in n, but where C must grow super-polynomially in n to render $h^+(\Pi^C)$ perfect? Most extremely, where on top of this h^* can be computed in polynomial time?*

We conjecture that the answer to this one is "yes", but our proof attempts so far did not succeed. Note here that the simple proof of Proposition 3 above relies crucially on needing to test only whether $h^+(\Pi^C) = \infty$, which can be done in polynomial time. On solvable tasks, as demanded in Open Question 1, perfect $h^+(\Pi^C)$ will be finite, so even for small C it is **NP**-hard to decide whether a given bound is met. For illustration: Say that, in the proof of Proposition 3, we introduce a "side route" in the SAT encoding, rendering unsat cases solvable but via a longer plan. Then we can still read off sat vs. unsat from perfect $h^+(\Pi^C)$, but we can no longer do so in polynomial time, so do not get a contradiction to the hardness of SAT.

Characterizing $h^+(\Pi^C)$ w/o Compilation

Trying to lead proofs about $h^+(\Pi^C)$, it can be annoying that one always has to do the mapping from original task to compiled task first. To make do without this, we now characterize $h^+(\Pi^C)$ directly in terms of the original planning task. Focusing on Π^{Cm} only for the moment, $h^+(\Pi^{Cm})$ can be understood as the following hybrid of h^m and h^+:

Definition 2 (Marrying h^m with h^+: h^{m+}) *The* critical path delete relaxation heuristic h^{m+} *is defined as $h^{m+} := h^{m+}(\{G\})$, where $h^{m+}(\mathcal{G})$, for a set $\mathcal{G}$ of fact sets, is the function that satisfies $h^{m+}(\mathcal{G}) =$*

$$\begin{cases} 0 & \forall g \in \mathcal{G} : g \subseteq I \\ 1 + \min_{a \in A, \emptyset \neq \mathcal{G}' \subseteq \{g \in \mathcal{G} \mid R(g,a) \neq \bot\}} & \\ \quad h^{m+}((\mathcal{G} \setminus \mathcal{G}') \cup \{\bigcup_{g \in \mathcal{G}'} R(g,a)\}) & \forall g \in \mathcal{G} : |g| \leq m \\ h^{m+}(\bigcup_{g \in \mathcal{G}} \{g' \subseteq g \mid |g'| \leq m\}) & \text{otherwise} \end{cases}$$

The underlying idea here is that the delete relaxation can be understood as allowing to achieve sub-goals *separately*: We worry only about the part of the sub-goal we can support, not about other parts that the same action may contradict. For $m = 1$, this means to ignore "delete lists" altogether as the same action never both supports and contradicts a single fact. For $m > 1$, we have to adequately (non-contradictingly) achieve all size-m sub-goals. That generalization is exactly the one made by $h^+(\Pi^{Cm})$. Definition 2 captures this by splitting up the goal (initially, the global goal fact set of the planning task) into all size-$\leq m$ sub-goals in the bottom case. For any given action in the middle case, these sub-goals are regressed separately, so each must be achieved non-contradictingly but contradictions across sub-goals are ignored.

There are two important subtleties in Definition 2, which distinguish it from what we would have to write in order to capture Π^C_{ce} instead of Π^C. First, we hand over the *union* $\bigcup_{g \in \mathcal{G}'} R(g,a)$ of the regressed sub-goals, forcing achievement of all these conditions conjunctively, like in Π^C when selecting C', for $a^{C'}$, to correspond to the set of conjunctions $\mathcal{G}'$. In particular, taking the union will give rise to *cross-dependencies* arising from several size-$\leq m$ sub-goals $g \in \mathcal{G}'$ ("cross-context π-fluents" in the parlance of Keyder et al. (2012)). To capture Π^C_{ce}, we can instead hand over each sub-goal $g \in \mathcal{G}'$ separately. Second, in the minimization, we minimize over pairs of action a and achieved sub-goal set $\mathcal{G}'$, as opposed to minimizing only over a and forcing $\mathcal{G}'$ to be maximal, i. e., setting $\mathcal{G}' = \{g \in \mathcal{G} \mid R(g,a) \neq \bot\}$. The latter would be suitable for capturing Π^C_{ce}, where there is no point in leaving out a "possible benefit" of the action a. In Π^C, that is not so because larger $\mathcal{G}'$ may give rise to additional cross-dependencies. For example, if $\mathit{eff}_a = \{p\}$ and $\mathcal{G} = \{\{p, q_1\}, \{p, q_2\}\}$ where q_1 and q_2 are impossible to achieve together, then $\mathcal{G}' = \{\{p, q_1\}, \{p, q_2\}\}$ leads to the unsolvable sub-goal $\{\{q_1, q_2\}\}$, while $\mathcal{G}' = \{\{p, q_1\}\}$ leads to the sub-goal $\{\{q_1\}, \{p, q_2\}\}$ which is solvable because we can achieve each of q_1 and $\{p, q_2\}$ separately.

To prove that Definition 2 does indeed capture $h^+(\Pi^{Cm})$, we start with the simple case $m = 1$, which will be employed below in the proof for the general case:

Theorem 1 $h^+ = h^{1+}$.

Proof: We show that, for $m = 1$, the h^{m+} equation simplifies to Equation 2. For $m = 1$, the bottom case just splits $\mathcal{G}$ up into its single goal facts. Hence the internal structure of $\mathcal{G}$ – the fact subsets it contains – does not matter; it matters only which facts are contained in any of these fact subsets. We can thus perceive $\mathcal{G}$ as a set goal facts, equivalently re-writing the equation to:

$$\begin{cases} 0 & \forall g \in \mathcal{G} : g \in I \\ 1 + \min_{a \in A, \emptyset \neq \mathcal{G}' \subseteq \{g \in \mathcal{G} \mid R(\{g\},a) \neq \bot\}} & \\ \quad h^{1+}((\mathcal{G} \setminus \mathcal{G}') \cup \bigcup_{g \in \mathcal{G}'} R(\{g\},a)) & \text{otherwise} \end{cases}$$

For a single goal fact $g \in \mathcal{G}'$, $R(\{g\}, a)$ is defined iff $g \in \mathit{eff}(a)$. Thus we can re-write the above to:

$$\begin{cases} 0 & \forall g \in \mathcal{G} : g \in I \\ 1 + \min_{a \in A, \emptyset \neq \mathcal{G}' \subseteq \mathcal{G} \cap \mathit{eff}(a)} & \\ \quad h^{1+}((\mathcal{G} \setminus \mathcal{G}') \cup \bigcup_{g \in \mathcal{G}'} R(\{g\},a)) & \text{otherwise} \end{cases}$$

Next, consider the regressed goal $(\mathcal{G}\setminus\mathcal{G}')\cup\bigcup_{g\in\mathcal{G}'} R(\{g\},a)$. For each $g\in\mathcal{G}'$, $R(\{g\},a)=pre(a)$. Thus the regressed goal is $(\mathcal{G}\setminus\mathcal{G}')\cup pre(a)$, giving us:

$$\begin{cases} 0 & \forall g\in\mathcal{G}: g\in I \\ 1+\min_{a\in A,\emptyset\neq\mathcal{G}'\subseteq\mathcal{G}\cap \mathit{eff}(a)} & \\ \quad h^{1+}((\mathcal{G}\setminus\mathcal{G}')\cup pre(a)) & \text{otherwise} \end{cases}$$

Observe that there is no point in choosing $\mathcal{G}'\subset\mathcal{G}\cap \mathit{eff}(a)$, i. e., using a to achieve a strict subset of its possible benefit $\mathcal{G}\cap \mathit{eff}(a)$, because that can only lead to a larger sub-goal $(\mathcal{G}\setminus\mathcal{G}')\cup pre(a)$. So we equivalently obtain:

$$\begin{cases} 0 & \forall g\in\mathcal{G}: g\in I \\ 1+\min_{a\in A,\emptyset\neq\mathcal{G}'=\mathcal{G}\cap \mathit{eff}(a)} & \\ \quad h^{1+}((\mathcal{G}\setminus\mathcal{G}')\cup pre(a)) & \text{otherwise} \end{cases}$$

With minimal re-writing, this turns into:

$$\begin{cases} 0 & \mathcal{G}\subseteq I \\ 1+\min_{a\in A,\emptyset\neq\mathcal{G}\cap \mathit{eff}(a)} & \\ \quad h^{1+}((\mathcal{G}\setminus \mathit{eff}(a))\cup pre(a)) & \text{otherwise} \end{cases}$$

This last equation is obviously equivalent to Equation 2, proving the claim. ∎

For $m=1$, $\Pi^{Cm}=\Pi$ so $h^+=h^+(\Pi^{Cm})$ and by Theorem 1 we get $h^+(\Pi^{Cm})=h^{m+}(\Pi)$ as desired. We now generalize this to arbitrary m:

Theorem 2 $h^+(\Pi^{Cm})=h^{m+}(\Pi)$.

Proof Sketch: By Theorem 1, for any Π we have $h^+(\Pi)=h^{1+}(\Pi)$. Applying this to $\Pi:=\Pi^{Cm}$, we get $h^+(\Pi^{Cm})=h^{1+}(\Pi^{Cm})$. It thus suffices to prove that $h^{1+}(\Pi^{Cm})=h^{m+}(\Pi)$. This is straightforward (but notationally cumbersome) based on comparing two equations, characterizing $h^{1+}(\Pi^{Cm})$ respectively $h^{m+}(\Pi)$.

For $h^{1+}(\Pi^{Cm})$, our equation (called *Equation I*) simply applies Definition 2 to Π^{Cm}:

$$\begin{cases} 0 & \forall g\in\mathcal{G}: g\subseteq I^C \\ 1+\min_{a^{C'}\in A^C,\emptyset\neq\mathcal{G}'\subseteq\{g\in\mathcal{G}\mid R(g,a^{C'})\neq\bot\}} & \\ \quad h^{m+}((\mathcal{G}\setminus\mathcal{G}')\cup\{\bigcup_{g\in\mathcal{G}'} R(g,a^{C'})\}) & \forall g\in\mathcal{G}: |g|\leq 1 \\ h^{m+}(\bigcup_{g\in\mathcal{G}}\{g'\subseteq g\mid |g'|=1\}) & \text{otherwise} \end{cases}$$

For $h^{m+}(\Pi)$, we need to do a little more work as we need to get rid of an irrelevant conceptual difference between Π^{Cm} and the equation defining h^{m+}: Whereas the latter splits up sub-goals only if their size is greater than m, Π^{Cm} always includes all possible π-fluents, even into sub-goals of size $\leq m$. Our new equation (called *Equation II*) modifies Definition 2 to do the same. We call $\mathcal{G}$ *completed* if, for all $g\in\mathcal{G}$, every $g'\subseteq g$ with $|g'|\leq m$ is contained in $\mathcal{G}$ as well:

$$\begin{cases} 0 & \forall g\in\mathcal{G}: g\subseteq I \\ 1+\min_{a\in A,\emptyset\neq\mathcal{G}'\subseteq\{g\in\mathcal{G}\mid R(g,a)\neq\bot\}} & \\ \quad h^{m+}((\mathcal{G}\setminus\mathcal{G}')\cup\{\bigcup_{g\in\mathcal{G}'} R(g,a)\}) & \\ \qquad\qquad \mathcal{G}\text{ is completed and } \forall g\in\mathcal{G}: |g|\leq m & \\ h^{m+}(\bigcup_{g\in\mathcal{G}}\{g'\subseteq g\mid |g'|\leq m\}) & \text{otherwise} \end{cases}$$

This is equivalent because we only add subsumed sub-goals.

Viewing each of Equations I and II as a tree whose root node is the "initializing call" containing the goal of the planning task, we show that the two trees are isomorphic. Namely, using the suffixes [I] and [II] to identify the tree, whenever the middle case applies we have:

$$(*)\ \mathcal{G}[I]=\{\{\pi_g\}\mid g\in\mathcal{G}[II]\}$$

To understand this intuitively, consider Equation II. This works on size-$\leq m$ sub-goals. Equation I works on singleton π-fluents representing size-$\leq m$ sub-goals. The original goal G gets split up into size-$\leq m$ subsets in II, vs. the π-fluents G^C in I, so we have (*). A sub-goal $g[I]=\pi_g$ in I can be regressed through $a^{C'}$ iff a achieves part of g and contradicts none of it; the same condition is applied in II. So the set $\mathcal{G}'$ of sub-goals tackled by a in II corresponds via (*) with that tackled by $a^{C'}$ in I. Finally, with $\mathcal{G}'$ having (*), the regressed sub-goal in I collects $pre(a)$ and $g\setminus \mathit{eff}(a)$ for all $\pi_g\in\mathcal{G}'$; the same is done in II, so (*) is preserved, concluding the proof. ∎

We remark that, from known results about $h^+(\Pi^{Cm})$, Theorem 2 implies that both $h^m\leq h^{m+}$ and $h^+\leq h^{m+}$: The marriage of h^m with h^+ yields a heuristic stronger than each of its sources, as one would expect.

Regarding dead-end detection power, it is easy to see that $h^{m+}=\infty$ iff $h^m=\infty$, i. e., like for $m=1$, the dead-end detection power of h^{m+} is the same as that of the corresponding critical-path heuristic.[3]

The above can be generalized to deal with arbitray C, i. e., to compute $h^{C+}=h^+(\Pi^C)$ using arbitrary Π^C instead of Π^{Cm}: In the bottom case of Definition 2, instead of splitting up into all size-m subsets, split up into the sets $c\in C$ (adapting the condition for the middle case accordingly to $\forall g\in\mathcal{G}\exists c\in C: g\subseteq c$).

Despite these niceties, we can't help but record:

Open Question 2 *What is this good for?*

We see two potential uses: (a) as a more direct way to *formulate* $h^+(\Pi^C)$ and thus ease leading proofs about its properties; and (b) as a more direct way to *compute* $h^+(\Pi^C)$, not necessitating a compilation step and thus being more efficient. Regarding (a), we haven't found any use case yet. Regarding (b), the most immediate idea is to extract "h^{2FF}" from a planning graph in a similar manner as for h^{FF} from a relaxed planning graph, considering pairs of sub-goal facts instead of single facts, following the correspondence between the equations characterizing h^{2+} vs. h^{1+}. However, there is no need for these equations to come up with h^{2FF}, and indeed Alcazar et al. (2013) already devised, implemented, and tested a variant of this idea, simply from the perspective of extending h^{FF} to correspond to Π^{C2}. As Alcazar et al. also already pointed out, "$h^{m\text{FF}}$" for arbitrary m can be computed from h^m respectively from an m-planning graph maintaining size-m mutexes. From that perspective, the main value of our work here is providing a theory background towards understanding and extending that technique.

[3] Similarly, for any C whose largest conjunction has size m, $h^+(\Pi^C)=\infty$ only if $h^m=\infty$.

It appears straightforward to extend $h^{m\text{FF}}$ to arbitrary conjunction sets C. A more tricky question, that might be answered using our formalization, is how exactly $h^{m\text{FF}}$ relates to the previous techniques $h^{\text{FF}}(\Pi^C)$ vs. $h^{\text{FF}}(\Pi^C_{ce})$.

Causal Graphs et al.

We now get back to the core motivation of this work, "scratching the itch". The aim is to understand under what circumstances "small" (i. e., polynomial-size) C is enough to render $h^+(\Pi^C)$ perfect. As an approach towards answering that question, we have taken the line of *identifying causal graph (CG) fragments (plus restrictions on the DTGs as needed) where $h^+(\Pi^{C2})$ is perfect*. In other words, adopt distinction lines as in many previous works on tractability, and see how far they carry when using only fact pairs.

The restriction to fact pairs is a bit arbitrary and mainly practically motivated. In particular, Keyder et al.'s (2012) implementation of semi-relaxed plan heuristics uses a subset of fact pairs. Then again, using all fact pairs in that implementation typically is infeasible, so we're still on the idealized side in our theory. In any case, the far more limiting fact here is that we got stopped in tracks right at the beginning. Having in mind initially to kill fork CGs quickly and then move on to more interesting quarters, we ended up spending lots of time racking our brains about even very small extensions to fork CGs, and indeed quite some time about fork CGs themselves.

What follows is thus a very simple fragment that we did manage to analyze. We remark that the proof is derived from a proof for (a generalization of) the VisitAll domain, for which also selecting all fact pairs is enough to render $h^+(\Pi^C)$ perfect.

We presume the reader is familiar with fork causal graphs. We will denote them here as planning tasks with a single "root variable" x, and with n "leaf" variables $y_1, \ldots, y_n$. The actions moving x do not have any preconditions on variables other than x, while the actions moving y_i may have preconditions on both y_i and x. So far, this is the standard fork CG setup. We impose the additional restriction, for all y_i, that y_i is Boolean and that there is only a single action affecting y_i. This essentially means that achieving the goal for y_i comes down to reaching a particular node in DTG_x (namely the one forming the precondition for y_i). We furthermore impose the restriction that x has no own goal, i. e., solving the task is just about moving the leaves into place (we will show later on that this restriction can be lifted, at least partially), and that every action moving x has a precondition on x.

We assume WLOG that initially each y_i is false, that the goal for each y_i is to be true (if y_i has no goal we can remove it without affecting either of h^* or $h^+(\Pi^C)$), and that the action for each y_i does have a precondition on x (else y_i moves independently and can be removed affecting h^* and $h^+(\Pi^C)$ in exactly the same way) and no precondition on y_i (that precondition could only be $y_i = False$, which is already true anyhow and thus affects neither h^* nor $h^+(\Pi^C)$).

We denote the DTG of x as a graph $DTG_x = (N, E)$ where the nodes N are the x values and the edges E correspond to the actions moving x. We denote by $n_i \in N$ the precondition on x of the action moving y_i, and by N_y the union of all n_i, i. e., those nodes we need to reach. We denote by n_0 the initial value of x. We denote facts $x = n$ simply by n, and we denote facts $y_i = True$ simply by y_i. We denote actions moving x by $go(d, d')$, and actions moving y_i by $do(i)$.

We refer to the class of planning tasks just described as *simple forks with binary leaves*.

Theorem 3 *$h^+(\Pi^{C2})$ is perfect for simple forks with binary leaves.*

The proof of this theorem is via a series of lemmas. First, it is easy to see that h^2-mutexes are recognized by Π^{C2}:

Lemma 1 *If $h^2(\{p, q\}) = \infty$, then $\pi_{p,q}$ is unreachable in Π^{C2}.*

This is simply because Π^{C2} captures support paths for all pairs of facts, just like h^2 does.

The following lemmas are basically concerned with paths, through the graph (N, E), that must be present in a relaxed plan for Π^{C2}. In the proofs, we will not explicitly distinguish the compiled actions in Π^{C2} from the original actions they are based on; instead, we will just talk about what preconditions are needed (will be present in Π^{C2}) if the original action a is to add a particular π-fluent π_c, i. e., if the corresponding conjunction c is added into the set C' for the compiled action $a^{C'}$.

The first lemma is a simple observation about achieving a pair of facts of the form "have y_i and now at n". Namely, to get that pair, we first need to get y_i and then move along a path to n:

Lemma 2 *Let Π be a simple fork with binary leaves, and let $\vec{a} = \langle a_1, \ldots, a_m \rangle$ be any sequence of actions applicable in Π^{C2}. Let s_k be the state that results from executing the prefix $\langle a_1, \ldots, a_k \rangle$. If $\pi_{y_i,n} \in s_k$, then $\langle a_1, \ldots, a_k \rangle$ contains a subsequence of actions that form a directed path in (N, E) from n_i to n.*

Proof: By induction on k. The base case, $k = 0$, is trivial, as it does not contain any $\pi_{y_i,n}$. Assume the claim holds for all $j < k$. The induction step proves it holds for k as well.

If $\pi_{y_i,n} \in s_k$, this either means that $\pi_{y_i,n} \in s_{k-1}$ (covered by induction hypothesis), or that a_k adds $\pi_{y_i,n}$. Say first that $a_k = do(i)$. Then $\pi_{y_i,n}$ can only be added if the compiled action has the precondition $\pi_{n_i,n}$ (only the y_i part can be added, n must have been true beforehand already). With Lemma 1, we must have $n_i = n$ or else the compiled action's precondition would be unreachable, in contradiction to applicability. But then, the directed path from n_i to n is empty and the claim holds trivially.

Say now that $a_k = go(d, d')$. Then $\pi_{y_i,n}$ can only be added if $a_k = go(d, n)$ (only the n part can be added, y_i must have been true beforehand already). But that compiled action has the precondition $\pi_{y_i,d}$, so by induction hypothesis $\langle a_1, \ldots, a_{k-1} \rangle$ contains a subsequence of actions that form a directed path from n_i to d in (N, E). Adding a_k to that subsequence forms the desired path from n_i to n. ∎

Our next lemma exploits the previous observation to show that any relaxed plan for Π^{C2} must, for every pair of the

target nodes N_y, contain a directed path between these two nodes in some order:[4]

Lemma 3 *Let Π be a simple fork with binary leaves, and let $\vec{a} = \langle a_1, \ldots, a_m \rangle$ be a relaxed plan for Π^{C2}. Then, for every $n_i \neq n_j \in N_y$, there is a subsequence of actions in $\vec{a}$ that form a directed path in (N, E) either from n_i to n_j or from n_j to n_i.*

Proof: The goal in Π^{C2} contains π_{y_i,y_j}. The only compiled actions which can achieve this are (1) $do(i)$ with precondition π_{y_j,n_i} (only the y_i part can be added, y_j must have been true beforehand already), or (2) $do(j)$ with precondition π_{y_i,n_j} (only the y_j part can be added, y_i must have been true beforehand already). Thus $\vec{a}$ must contain either of these two compiled actions. If $\vec{a}$ contains (i) $do(i)$ with precondition π_{y_j,n_i}, then by Lemma 2 $\vec{a}$ contains a subsequence of actions that form a directed path in (N, E) from n_j to n_i, showing the claim. Similarly for (2). ∎

We are now finally ready to prove Theorem 3 itself, by exploiting Lemma 3 in an argument as to how a relaxed plan can move through DTG_x:

Proof:[of Theorem 3] Let Π be a simple fork with binary leaves, and let $\vec{a} = \langle a_1, \ldots, a_m \rangle$ be a relaxed plan for Π^{C2}. It suffices to prove that there exists a subsequence of $\vec{a}$ that is a plan for Π.

By Lemma 3, for each pair of nodes $n_i, n_j \in N_y$, $\vec{a}$ contains a path from from n_i to n_j or vice versa. Hence, the graph

$$T_y = \langle N_y, \{(n_i, n_j) \mid \text{a path from } n_i \text{ to } n_j \text{ is in } \vec{a}\} \rangle$$

(or a subgraph of it, should $\vec{a}$ happen to contain paths in both directions between some pairs of nodes) is a tournament graph, and therefore must contain a Hamiltonian path, i.e., a directed path that visits every node (exactly once, in the graph T_y).[5] Furthermore, $\vec{a}$ must contain a path from n_0 to every n_i, as otherwise it could not achieve y_i. Hence, a subsequence of $\vec{a}$ must form a contiguous path, $n_0, n_{i_1}, \ldots, n_{i_n}$, through the nodes in N_y. Although in $\vec{a}$ the $do(i)$ actions can be applied at any time point after passing through n_i, whereas a plan for Π must apply $do(i)$ exactly when it is at n_i, the summed-up cost for applying all these actions is the same, proving the claim. ∎

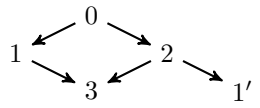
Having concluded this proof, the immediate question is whether all the restrictions on Π, such as the root variable having no own goal, and every action moving it having a precondition, are necessary.

[4]As a reminder: Being a relaxed plan for Π^{C2} is the same as being a plan for Π^{C2}, as we do not include any delete effects in our definition of the compilation here. We include the "relaxed" in the hope that being explicit is clearer.

[5]A *tournament graph* on n nodes is any directed graph obtained by assigning a direction to every edge in a complete undirected graph of size n. The proof that such graphs must contain a Hamiltonian path is due to Rédei (1934). A proof can be found in, e.g., the textbook by Moon (1968), or at `http://en.wikipedia.org/wiki/Tournament_(graph_theory)`.

Some of the restrictions can be relaxed. For example, if the root variable x has a goal value, n_x, and $n_x \neq n_i$ for all $n_i \in N_y$, the theorem still holds. The goal of Π^{C2} includes π_{y_i,n_x}, for all $n_i \in N_y$, so by Lemma 2 any relaxed plan $\vec{a}$ contains a path from n_i to n_x. Thus, adding n_x to the graph T_y in the proof above still leaves it a tournament graph, and because all edges between n_x and other nodes are (or can be chosen to be) directed towards n_x, this node can appear last in the Hamiltonian path.

The restriction to a single root variable "target node" per leaf variable (i.e., a single precondition $x = n_i$ common to actions that achieve $y_i = True$), on the other hand, is indeed necessary: if there are two options for achieving a fact y_i, we can construct a graph which allows relaxed plans in Π^{C2} to cheat, by achieving pairs π_{y_i,y_j} and π_{y_i,y_l} on separate branches of a directed tree. As a simple example, suppose there are three leaf variables, y_1, y_2 and y_3, and DTG_x is the following graph:



where 0 is the initial state of x, action $do(i)$ has precondition $x = i$, but there is an additional action $do'(1)$ with precondition $x = 1'$ (and effect $y_1 = True$). In Π^{C2}, the relaxed plan $go(0,1), do(1), go(1,3), do(3)$ achieves π_{y_1,y_3}, $go(0,2), do(2), go(2,3), do(3)$ achieves π_{y_2,y_3} and $go(0,2), do(2), go(2,1'), do'(1)$ achieves π_{y_1,y_2}, and hence the concatenation of all three achieves the goal. But the real problem has no solution.

Many of the other questions surrounding the restrictions in Theorem 3 do appear easy to answer, while a few may be hard. The reason we do not yet have answers is that most of our time was spent considering a slighly more general fragment than fork causal graphs, namely a more direct generalization of the VisitAll domain, where moving the root variable may have arbitrary side effects on subsets of Boolean leaf variables. This setting, it turned out, is substantially more complex to analyze.

In any case, the real open question remains:

Open Question 3 *Can the approach of analyzing CG-based tractable fragments ever be brought up to a level suitable for targeted selection of C in practice?*

Our idea here is, if for sufficiently large/many fragments of planning we know exactly which C are needed to render $h^+(\Pi^C)$ perfect, then we may be able to use these as "building blocks" for selection methods with a strong theory justification. In particular, for fact-pair cases, the idea would be to not necessarily consider all fact pairs but just the minimal subsets required.

Our speed of progress so far, and counter-examples identified for very simple fragments of planning, suggests that this approach is challenging to say the least, doomed perhaps. But the jury is still out. We speculate some more in the conclusion, and for now get back to some actual results:

$h^+(\Pi^C)$ vs. Red-Black vs. Fluent Merging

A different way to approach the power of $h^+(\Pi^C)$ is to compare it with other partial delete relaxation methods, i. e., alternative methods able to interpolate all the way between h^+ and h^*. Exactly two such alternative methods are known, at this time: *red-black planning* (Katz, Hoffmann, and Domshlak 2013b; 2013a; Katz and Hoffmann 2013) and *fluent merging* (van den Briel, Kambhampati, and Vossen 2007; Seipp and Helmert 2011). *Which of these approaches can simulate which other ones, i. e., compute an at least as good heuristic, with polynomial overhead?*

As fluent merging necessitates the use of 0-cost actions, in what follows we consider the arbitrary-costs case. The answers to the question we are posing are the same anyhow when assuming uniform costs (for those cases where that assumption is possible).

In fluent merging, we choose a subset $M \subseteq V$ of variables to merge, and replace them with a single variable v^M whose domain is the cross-product of the domains of $v \in M$.[6] Every action a that touches any $v \in M$ is then replaced by a set of actions resulting from the enumeration of possible precondition/effect values of v^M (i. e., states over M) that match a's precondition and effect. That is, we complete $\mathit{eff}(a)$ with an assignment p to the remaining variables, where p matches $pre(a)$; we complete $pre(a)$ with the same assignment p on variables not occuring in $\mathit{eff}(a)$, and with an arbitrary assignment on those variables that do occur in $\mathit{eff}(a)$. If M touches the goal, then an artificial goal value is introduced for v^M, reached by an artificial 0-cost action from every assignment to M that complies with the goal. Denote the resulting heuristic, i. e., the length of an optimal relaxed plan in the pre-merged task, by $h^{\text{Merge}+}$. It hasn't to our knowledge been noted before, but is obvious, that for $M = V$ we get $h^{\text{Merge}+} = h^*$ (so indeed this is an "interpolation method" in the sense above).

In red-black planning, we "delete-relax" only a subset V^R of the finite-domain state variables (the "red" ones), applying the original semantics to the remaining variables V^B (the "black" ones). That is, red variables accumulate their values ($\mathit{eff}(a)$ gets added to the state) while the black variables switch between their values ($\mathit{eff}(a)$ over-writes the state). Denote the resulting heuristic, i. e., the length of an optimal red-black plan, by $h^{\text{RB}+}$. Obviously, setting $V^B = \emptyset$ we get $h^{\text{RB}+} = h^+$, and setting $V^R = \emptyset$ we get $h^{\text{RB}+} = h^*$. The known "tractable fragments" (i. e., polynomial-time satisficing red-black plan generation) require (a) a fixed number of black variables each with fixed domain size, or (b) an acyclic *black causal graph* (projection of the causal graph onto V^B) where each black variable has only invertible value transitions in its DTG.

It is not completely clear what "polynomial overhead" should be taken to mean in this context. We choose to ignore the complexity of optimal partially-relaxed-plan generation, which makes sense as this underlies all three frameworks and will be approximated by satisficing partially-relaxed-plan generation just like in the standard delete relaxation. Given this, "polynomial overhead" for fluent merging means that $|M|$ is fixed; for $h^+(\Pi^C)$ we take it to mean that $|C|$ is polynomially bounded.[7] For red-black planning, we take "polynomial overhead" to mean "inside a known tractable fragment"; this is not fair as there may be yet unknown tractable fragments, but it is the best we can do for now.

As per this simulation framework, it turns out that all three approaches are orthogonal, with a single exception:

Theorem 4 *None of $h^+(\Pi^C)$, red-black planning, and fluent merging can simulate any other with polynomial overhead, except that $h^+(\Pi^C)$ simulates fluent merging on M when setting C to contain all fact conjunctions c over M (including c mentioning the same variable more than once).*

Proof Sketch: To see that red-black planning cannot simulate either of $h^+(\Pi^C)$ or fluent merging, it suffices to construct an example whose only "flaw" is small and easy to fix, but outside a known tractable fragment. This can be based, e. g., on having to buy a car, consuming a piece of gold, but the goal being to have both the car and the gold. Merging the two variables (car and gold) yields $h^{\text{Merge}+} = h^* = \infty$, and a single conjunction yields $h^+(\Pi^C) = \infty$. For $h^{\text{RB}+} = \infty$ we would need both variables to be black, yielding a cyclic causal graph; it is easy to scale variable domains and the number of variables so that neither $h^{\text{Merge}+} = h^*$ nor $h^+(\Pi^C) = h^*$ is affected.

$h^+(\Pi^C)$ cannot simulate red-black planning because there are planning tasks whose causal graphs are lines (in particular, DAGs) and all of whose variables are invertible, but where h^* is exponentially large. This is tractable for $h^{\text{RB}+} = h^*$ (using a succinct plan representation), but is not tractable for $h^+(\Pi^C)$ by Proposition 2.

Fluent merging cannot simulate red-black planning because sometimes painting a single variable black suffices whereas we would need to merge all variables to obtain $h^{\text{Merge}+} = h^*$. One such example is "star-shaped switches", where a robot starts in the middle node of a star graph, has to move to every leaf node turning on a switch, and has to be back in the middle at the end. Painting the robot variable black obviously gives $h^{\text{RB}+} = h^*$. However, if M leaves out a single variable then $h^{\text{Merge}+} < h^*$. This is obvious for the robot variable. If switch variable $v \notin M$, then a relaxed plan can solve v_M as appropriate (switching all other switches on and moving back to the middle), then move outwards to v's node and switch it on, but not move back to the middle as the two goals "other-switches-on-and-robot-at-middle" as well as "switch-v-on" are both already true.

The same example shows that fluent merging cannot simulate $h^+(\Pi^C)$, as by Theorem 3 we have $h^+(\Pi^{C2}) = h^*$.

Consider finally the only positive result. Denoting the pre-merged task by Π^M, our proof considers an optimal

[6] Note that this is a restricted version of the technique, merging only a single subset of variables. One can instead merge several subsets, potentially with overlaps between them. We restrict outselves to the simpler variant in what follows.

[7] We ignore the exponential growth of Π^C in $|C|$ because (a) this can be largely fixed using Π^C_{ce} (Keyder, Hoffmann, and Haslum 2012), and (b) it appears that alternate methods for computing approximations of $h^+(\Pi^C)$, not going via a compilation, can avoid that blow-up altogether, cf. our comments below Open Question 2.

relaxed plan $\vec{a} = \langle a_1^{C'_1}, \ldots, a_n^{C'_n} \rangle$ for Π^C, and shows that we can transform $\vec{a}$ step-by-step into a relaxed plan $\vec{a}^M = \langle a_1^M, \ldots, a_n^M \rangle$ for Π^M based on the same actions a_i from the original planning task. Then $\vec{a}^M$ has the same cost as $\vec{a}$, implying that $h^+(\Pi^M) \leq h^+(\Pi^C)$ as we need to prove.

While that idea sounds simple, spelling it out was unexpectedly cumbersome, taking us a few iterations and ending up being a full page long in this format (perhaps there are simpler proofs). Omitting the details, consider the structure of relaxed plans in Π^C vs. Π^M. In the latter, the M-states visited form a tree, the root being the initial state, actions a_i^M connecting to any M-state already visited. But what is the structure in Π^C? While it is easy to see that each C'_i contains at most one full assignment to M (otherwise the precondition would contain an unreachable mutex pair, of different values for the same variable), nothing forces the C'_i to include any full assignment. So the structure of $\vec{a}$ is less rigid as that of relaxed plans in Π^M, making commitments only where necessary. Our proof shows how to instantiate these partial commitments to full commitments, extending each C'_i to correspond exactly to a full assignment to M: At the initial state, the commitment is full already. Every action i must have a preceding action $r(i)$ adding the π-fluent corresponding to $a_i^{C'_i}$'s entire set of precondition facts. Assuming that $C'_{r(i)}$ has been fully committed already, we can extend the partial commitment made by C'_i accordingly. ∎

We would like to note that, for once, there are no open questions left here. Except of course what would happen were we to identify further tractable fragments of red-black planning, and whether more general variants of fluent merging (several variable subsets with potential overlaps) make a difference. As that is not "open enough" for the spirit of this paper, we now get into open questions for real:

Some Wild Speculations (aka "Conclusion")

Disregarding issues such as exponential separations between the different variants of Π^C, or the role of mutex pruning (removing actions with known mutexes in the precondition) in all this, let us focus our speculation on the questions we started out with: Can we characterize more accurately the circumstances under which $h^+(\Pi^C)$ becomes perfect? When does that require C to be exponentially large, and when is polynomial-size C enough?

Our most direct answer to these questions is Theorem 3, tractability of a restricted fork fragment. While our original plan had been to extend that result "bottom-up", proving tractability of different/ever larger fragments, the effort it took to analyze even slightly larger fragments suggests that perhaps a "top-down" approach might be more suitable:

Open Question 4 *Can we identify easily testable sufficient conditions for some subset D of conjunctions to* not *be required for rendering $h^+(\Pi^C)$ perfect?*

For example, if there is no undirected CG path between two variables, then presumably we do not need to include conjunctions involving both. But what if there is no *directed* CG path between them?

Or perhaps we can make progress by considering particular benchmark domains:

Open Question 5 *In which IPC benchmarks is h^{2+}, i. e., $h^+(\Pi^C)$ with C being all fact pairs, perfect? In those domains where it isn't, how does the search topology (local minima, exit distances, unrecognized dead-ends) differ from that known for h^+ (Hoffmann 2005)?*

A major source of speculations is whether we can somehow identify structural criteria – based on whatever notions, not necessarily efficiently testable – under which $h^+(\Pi^C)$ becomes perfect. Let us start with the most plausible one:

Open Question 6 *Say that a task Π is "m-decomposable" if there exists a partitioning $\{V_i\}$ of its variables whose largest V_i has size m, and where the length of an optimal plan for Π is equal to the sum of lengths of optimal plans for the projections onto V_i. Is h^{m+} perfect?*

While that is more like a conjecture than an open question, its practical use is doubtful: If we have m-decomposability, then basically we can split up the planning task into its pieces and have no need for a global heuristic addressing the whole task. Unless, of course, we don't actually know what the decomposition is, only its size; but that seems unlikely to happen in practice (?)

We close the paper with what are probably our two most speculative open questions. One is basically an attempt to answer Open Question 2:

Open Question 7 *Can syntactical criteria be identified which imply that the h^{m+} equation simplifies to Equation 1?*

We haven't got any idea how to do this; it should be said though that we did not spend much time trying.

Finally, thinking about the size of conjunctions needed often corresponds to thinking about "the value of how many variables we need to remember in order to avoid cheating". The "remembering" here corresponds to deleted values that are required again later on. The number of variables affected in this way intuitively corresponds to a "level of interference". For example, in VisitAll (and other fork-like domains) the only variable whose value we need to remember is the robot (the root of the fork); in puzzles, by contrast, achieving a desired value may typically involve deleting arbitrarily many other desired values. Taking "level of interference" m to be the maximal number of variables affected while achieving a target value, plus one for the target variable itself, the question is:

Open Question 8 *Say that a task Π has level of interference m (whatever that means, exactly). Is h^{m+} perfect?*

We are looking forward to some answers in the future, apologize for posing so many un-answered questions, and thank you for not having laid the paper aside before reaching this sentence.

Acknowledgments. We thank the anonymous reviewer (aka Malte Helmert) for many detailed and helpful comments.

References

Alcázar, V.; Borrajo, D.; Fernández, S.; and Fuentetaja, R. 2013. Revisiting regression in planning. In Rossi, F., ed., *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI'13)*, 2254–2260. AAAI Press.

Haslum, P. 2009. $h^m(P) = h^1(P^m)$: Alternative characterisations of the generalisation from $h^{\max}$ to h^m. In Gerevini, A.; Howe, A.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*, 354–357. AAAI Press.

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

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

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

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

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

Keyder, E.; Hoffmann, J.; and Haslum, P. 2012. Semi-relaxed plan heuristics. In Bonet, B.; McCluskey, L.; Silva, J. R.; and Williams, B., eds., *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*, 128–136. AAAI Press.

Moon, J. W. 1968. *Topics on Tournaments*. Holt, Rinehart and Winston, Inc. `http://www.gutenberg.org/ebooks/42833`.

Rédei, L. 1934. Ein kombinatorischer satz. *Acta. Litt. Szeged* 7:39–43.

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

van den Briel, M.; Kambhampati, S.; and Vossen, T. 2007. Fluent merging: A general technique to improve reachability heuristics and factored planning. In *ICAPS 2007 Workshop on Heuristics for Domain-Independent Planning: Progress, Ideas, Limitations, Challenges*.

Pushing the Limits of Partial Delete Relaxation: Red-Black DAG Heuristics

Michael Katz
IBM Haifa Research Labs
Haifa, Israel
katzm@il.ibm.com

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

Abstract

Red-black planning is a systematic approach to partial delete relaxation, taking into account *some* of the delete effects: Red variables take the relaxed (value-accumulating) semantics, while black variables take the regular semantics. Prior work on red-black plan heuristics has identified a powerful tractable fragment requiring the *black causal graph* – the projection of the causal graph onto the black variables – to be a DAG; but all implementations so far use a much simpler fragment where the black causal graph is required to not contain any arcs at all. We close that gap here, and we design techniques aimed at making red-black plans executable, short-cutting the search. Our experiments show that these techniques can yield significant improvements on those IPC benchmarks where non-trivial DAG black causal graphs exist.

Introduction

The *delete relaxation*, where state variables accumulate their values rather than switching between them, has played a key role in the success of satisficing planning systems, e. g. (Bonet and Geffner 2001; Hoffmann and Nebel 2001; Richter and Westphal 2010). Still, the delete relaxation has well-known pitfalls, for example the fundamental inability to account for moves back and forth (as done, e. g., by vehicles in transportation). It has thus been an actively researched question from the outset how to take *some* deletes into account, e. g. (Fox and Long 2001; Gerevini, Saetti, and Serina 2003; Helmert 2004; Helmert and Geffner 2008; Baier and Botea 2009; Cai, Hoffmann, and Helmert 2009; Haslum 2012; Keyder, Hoffmann, and Haslum 2012). Herein, we continue the most recent attempt, *red-black planning* (Katz, Hoffmann, and Domshlak 2013b; 2013a; Katz and Hoffmann 2013) where a subset of *red* state variables takes on the relaxed value-accumulating semantics, while the other *black* variables retain the regular semantics.

Katz et al. (2013b) introduced the red-black framework and conducted a theoretical investigation of tractability. Following up on this (2013a), they devised practical *red-black plan heuristics*, non-admissible heuristics generated by repairing fully delete-relaxed plans into red-black plans. Observing that this technique often suffers from dramatic overestimation incurred by following arbitrary decisions taken in delete-relaxed plans, Katz and Hoffmann (2013) refined the approach to rely less on such decisions, yielding a more flexible algorithm delivering better search guidance.

The *black causal graph* is the projection of the causal graph onto the black variables only. Both Katz et al. (2013a) and Katz and Hoffmann (2013) exploit, in theory, a tractable fragment characterized by *DAG black causal graphs*, but confine themselves to *arc-empty black causal graphs* – no arcs at all – in practice. Thus current red-black plan heuristics are based on a simplistic, almost trivial, tractable fragment of red-black planning. We herein close that gap, designing *red-black DAG heuristics* exploiting the full tractable fragment previously identified. To that end, we augment Katz and Hoffmann's implementation with a DAG-planning algorithm (executed several times within every call to the heuristic function). We devise some enhancements targeted at making the resulting red-black plans executable in the real task, stopping the search if they succeed in reaching the goal. In experiments on the relevant IPC benchmarks, we find that the gained informativity often pays off, reducing search and improving overall performance.

Background

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

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

Π is a **finite-domain representation (FDR)** planning task if $V^{\mathsf{R}} = \emptyset$, and is a **monotonic finite-domain representation (MFDR)** planning task if $V^{\mathsf{B}} = \emptyset$. Plans for MFDR tasks (i. e., for delete-relaxed tasks) can be generated in polynomial time. A key part of many satisficing planning systems is based on exploiting this property for deriving heuristic estimates, via delete-relaxing the task at hand. Generalizing this to red-black planning, the **red-black relaxation** of an FDR task Π relative to V^{R} is the RB task $\Pi^{*+}_{V^{\mathsf{R}}} = \langle V \setminus V^{\mathsf{R}}, V^{\mathsf{R}}, A, I, G\rangle$. A plan for $\Pi^{*+}_{V^{\mathsf{R}}}$ is a **red-black plan** for Π, and the length of a shortest possible red-black plan is denoted $h^{*+}_{V^{\mathsf{R}}}(\Pi)$. For arbitrary states s, $h^{*+}_{V^{\mathsf{R}}}(s)$ is defined via the RB task $\langle V \setminus V^{\mathsf{R}}, V^{\mathsf{R}}, A, s, G\rangle$. If $V^{\mathsf{R}} = V$, then red-black plans are **relaxed plans**, and $h^{*+}_{V^{\mathsf{R}}}$ coincides with the optimal delete relaxation heuristic h^+.

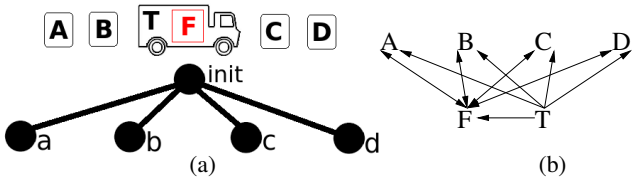


Figure 1: An example (a), and its causal graph (b).

In Figure 1, truck T needs to transport each package $X \in \{A, B, C, D\}$ to its respective goal location $x \in \{a, b, c, d\}$. The truck can only carry one package at a time, encoded by a Boolean variable F ("free"). A real plan has length 15 (8 loads/unloads, 7 drives), a relaxed plan has length 12 (4 drives suffice as there is no need to drive back). If we paint (only) T black, then $h^{*+}_{V^{\mathsf{R}}}(I) = 15$ as desired, but red-black plans may not be applicable in the real task, because F is still red so we can load several packages consecutively. Painting T *and* F black, that possibility disappears.[1]

Tractable fragments of red-black planning have been identified using standard structures. The **causal graph** CG_Π of Π is a digraph with vertices V. An arc (v, v') is in CG_Π if $v \neq v'$ and there exists an action $a \in A$ such that $(v, v') \in [\mathcal{V}(\mathsf{eff}(a)) \cup \mathcal{V}(\mathsf{pre}(a))] \times \mathcal{V}(\mathsf{eff}(a))$. The **domain transition graph** $DTG_\Pi(v)$ of a variable $v \in V$ is a labeled digraph with vertices $\mathcal{D}(v)$. The graph has an arc (d, d') induced by action a if $\mathsf{eff}(a)[v] = d'$, and either $\mathsf{pre}(a)[v] = d$ or $v \notin \mathcal{V}(\mathsf{pre}(a))$. The arc is labeled with its **outside condition** $\mathsf{pre}(a)[V \setminus \{v\}]$ and its **outside effect** $\mathsf{eff}(a)[V \setminus \{v\}]$.

The **black causal graph** CG^{B}_Π of Π is the sub-graph of CG_Π induced by V^{B}. An arc (d, d') is **relaxed side effects invertible**, **RSE-invertible** for short, if there exists an arc (d', d) with outside condition $\phi' \subseteq \phi \cup \psi$ where ϕ and ψ are the outside condition respectively outside effect of (d, d'). A variable v is RSE-invertible if all arcs in $DTG_\Pi(v)$ are RSE-invertible, and an RB task is RSE-invertible if all its black variables are. Prior work on red-black plan heuristics proved that plan generation for RSE-invertible RB tasks with **DAG** (acyclic) black causal graphs is tractable, but used the much simpler fragment restricted to **arc-empty** black causal graphs in practice. In Figure 1, both T and F are RSE-invertible; if we paint only T black then the black causal graph is arc-empty, and if we paint both T and F black then the black causal graph is (not arc-empty but) a DAG.

[1]Indeed, all optimal red-black plans (but not some non-optimal ones) then are real plans. We will get back to this below: As we shall see, the ability to increase red-black plan applicability is a main advantage of our red-black DAG heuristics over the simpler red-black plan heuristics devised in earlier work.

Algorithm : REDBLACKPLANNING(Π, R^+)

main
- $\pi \leftarrow \langle\rangle$
- **while** $R \not\supseteq R^+$
  - **do**
    - $A' = \{a \in A \mid \mathsf{pre}(a) \subseteq B \cup R, \mathsf{eff}(a) \cap (R^+ \setminus R) \neq \emptyset\}$
    - Select $a \in A'$
    - **if** $\mathsf{pre}(a)[V^{\mathsf{B}}] \not\subseteq I[\![\pi]\!]$
      - **then** $\pi \leftarrow \pi \circ$ ACHIEVE($\mathsf{pre}(a)[V^{\mathsf{B}}]$)
    - $\pi \leftarrow \pi \circ \langle a \rangle$
- **if** $G[V^{\mathsf{B}}] \not\subseteq I[\![\pi]\!]$
  - **then** $\pi \leftarrow \pi \circ$ ACHIEVE($G[V^{\mathsf{B}}]$)
- **return** π

procedure ACHIEVE(g)
- $I^{\mathsf{B}} \leftarrow I[\![\pi]\!][V^{\mathsf{B}}]; G^{\mathsf{B}} \leftarrow g$
- $A^{\mathsf{B}} \leftarrow \{a^{\mathsf{B}} \mid a \in A, a^{\mathsf{B}} = \langle \mathsf{pre}(a)[V^{\mathsf{B}}], \mathsf{eff}(a)[V^{\mathsf{B}}]\rangle,$ $\mathsf{pre}(a) \subseteq B \cup R\}$
- $\Pi^{\mathsf{B}} \leftarrow \langle V^{\mathsf{B}}, A^{\mathsf{B}}, I^{\mathsf{B}}, G^{\mathsf{B}}\rangle$
- **return** DAGPLANNING(Π^{B})

Figure 2: Katz and Hoffmann's (2013) red-black planning algorithm (abbreviated; for explanations see text).

Red-Black DAG Heuristics

As indicated, we augment Katz and Hoffmann's (2013) implementation with a DAG-planning algorithm. To provide the context, Figure 2 shows (the main parts of) Katz and Hoffmann's pseudo-code. The algorithm assumes as input the set R^+ of preconditions and goals on red variables in a fully delete-relaxed plan, i. e., $R^+ = G[V^{\mathsf{R}}] \cup \bigcup_{a \in \pi^+} \mathsf{pre}(a)[V^{\mathsf{R}}]$ where π^+ is a relaxed plan for Π. It then successively selects achieving actions for R^+, until all these red facts are true. Throughout the algorithm, R *denotes the set of red facts already achieved by the current red-black plan prefix* π*;* B *denotes the set of black variable values that can be achieved using only red outside conditions from* R. We have omitted the (simple) maintenance of R and B here as it is not needed to understand the present paper.

For each action $a \in A'$ selected to achieve new facts from R^+, and for the global goal condition at the end, there may be black variables that do not have the required values. For example, say we paint T and F black in Figure 1. Then R^+ will have the form $\{A = T, A = a, B = T, B = b, C = T, C = c, D = T, D = d\}$. In the initial state, A' will contain only load actions. Say we execute $a =$load$(A, init)$, entering $A = T$ into R and thus including unload(A, a) into A' in the next iteration. Trying to execute that action, we find that its black precondition $T = a$ is not satisfied. The call to ACHIEVE($\{T = a\}$) is responsible for rectifying this.

ACHIEVE(g) creates a task Π^{B} over Π's black variables, asking to achieve g. As Katz and Hoffmann showed, Π^{B} is solvable, has a DAG causal graph, and has strongly connected DTGs (when restricting to actions a where $\mathsf{pre}(a) \subseteq I[\![\pi]\!]$). From this and Theorem 4.4 of Chen and Gimenez (2010), it directly follows that a plan for Π^{B}, *in a succinct*

Algorithm : DAGPLANNING(Π^{B})

main
  $\pi^{\mathsf{B}} \leftarrow \langle\rangle$
  for $i = n$ **downto** 1
    // Denote $\pi^{\mathsf{B}} = \langle a_1, \ldots, a_k \rangle$
    $d \leftarrow I[v_i]$
    for $j = 1$ **to** k
      do **if** $\mathsf{pre}(a_j)[v_i]$ is defined **then** $\pi_j \leftarrow \pi_{v_i}(d, \mathsf{pre}(a_j)[v_i])$; $d \leftarrow \mathsf{pre}(a_j)[v_i]$
    $\pi_{k+1} \leftarrow \langle\rangle$
    if $G[v_i]$ is defined **then** $\pi_{k+1} \leftarrow \pi_{v_i}(d, G[v_i])$
    $\pi^{\mathsf{B}} \leftarrow \pi_1 \cdot \langle a_1 \rangle \cdot \ldots \cdot \pi_k \cdot \langle a_k \rangle \cdot \pi_{k+1}$
  return π^{B}

Figure 3: Planning algorithm for FDR tasks Π^{B} with DAG causal graph $CG_{\Pi^{\mathsf{B}}}$ and strongly connected DTGs. $v_1, \ldots, v_n$ is an ordering of variables V consistent with the topology of $CG_{\Pi^{\mathsf{B}}}$. $\pi_v(d, d')$ denotes an action sequence constituting a shortest path in $DTG_v(\Pi)$ from d to d'.

plan representation, can be generated in polynomial time.

The "succinct plan representation" just mentioned consists of recursive macro actions for pairs of initial-value/other-value within each variable's DTG; it is required as plans for Π^{B} may be exponentially long. Chen and Gimenez' algorithm handling these macros involves the exhaustive enumeration of shortest paths for the mentioned value pairs in all DTGs, and it returns highly redundant plans moving precondition variables back to their initial value in between every two requests. For example, if a truck unloads two packages at the same location, then it is moved back to its start location in between the two unload actions.

Katz and Hoffmann (2013) shunned the complexity of DAG planning, and considered Π^{B} with arc-empty causal graphs, solving which is trivial. In our work, after exploring a few options, we decided to use the simple algorithm in Figure 3: Starting at the leaf variables and working up to the roots, the partial plan π^{B} is augmented with plan fragments bringing the supporting variables into place (a similar algorithm was mentioned, but not used, by Helmert (2006)).

Proposition 1 *The algorithm* DAGPLANNING(Π^{B}) *is sound and complete, and its runtime is polynomial in the size of* Π^{B} *and the length of the plan* π^{B} *returned.*

Note here that the length of π^{B} is worst-case exponential in the size of Π^{B}, and so is the runtime of DAGPLANNING(Π^{B}). We trade the theoretical worst-case efficiency of Chen and Gimenez' algorithm against the practical advantage of not having to rely on exhaustive computation of shortest paths – anew for every call of DAGPLANNING, with "initial values" and DTGs from Π^{B} – for input tasks Π^{B} that typically have small plans (achieving the next action's black preconditions) anyhow.[2]

[2]One could estimate DAG plan length (e. g., using Helmert's (2006) causal graph heuristic), computing a red-black plan *length estimate* only. But that would forgo the possibility to actually execute DAG red-black plans, which is a key advantage in practice.

Unlike the macro-based algorithm of Chen and Gimenez, our DAGPLANNING algorithm does not superfluously keep switching supporting variables back to their initial values. But it is not especially clever, either: If variable v_0 supports two otherwise independent leaf variables v_1 and v_2, then the sub-plans for v_1 and v_2 will be inserted sequentially into π^{B}, losing any potential for synergies in the values of v_0 required. We developed a more flexible algorithm addressing that weakness through using a partially-ordered π^{B}, but that algorithm resulted in significantly worse empirical performance, so we do not include it here.

Enhancing Red-Black Plan Applicability

One crucial advantage of red-black plans, over fully-delete relaxed plans, is that they have a much higher chance of actually working for the original planning task. This is especially so for the more powerful DAG red-black plans we generate here. In Figure 1, as already mentioned, if we paint just T black then the red-black plan *might* work; but if we paint both T and F black – moving to a non-trivial DAG black causal graph – then *every optimal red-black plan definitely works*. A simple possibility for exploiting this, already implemented in Katz and Hoffmann's (2013) earlier work, is to *stop search* if the red-black plan generated for a search state s is a plan for s in the original task.

There is a catch here, though – the red-black plans we generate are not optimal and thus are not guaranteed to execute in Figure 1. In our experiments, we observed that the red-black plans often were not executable due to simple reasons. We fixed this by augmenting the algorithms with the two following applicability enhancements.

(1) Say that, as above, $R^+ = \{A = T, A = a, B = T, B = b, C = T, C = c, D = T, D = d\}$ and REDBLACKPLANNING started by selecting load$(A, init)$. Unload(A, a) *might* be next, but the algorithm might just as well select load$(B, init)$. With T and F black, load$(B, init)$ has the black precondition $F = true$. Calling ACHIEVE$(\{F = true\})$ will obtain that precondition using unload$(A, init)$. Note here that variable A is red so the detrimental side effect is ignored. The same phenomenon may occur in any domain with renewable resources (like transportation capacity). We tackle it by giving a preference to actions $a \in A'$ getting whose black preconditions does not involve deleting R^+ facts already achieved beforehand. To avoid excessive overhead, we approximate this by recording, in a pre-process, which red facts may be deleted by moving each black variable, and prefer an action if none of its black preconditions may incur any such side effects.

(2) Our second enhancement pertains to the DTG paths chosen for the black precondition variables in DAGPLANNING (after REDBLACKPLANNING has already selected the next action). The red outside conditions are by design all reached (contained in R), but we can prefer paths whose red outside conditions are "active", i. e., true when executing the current red-black plan prefix in the real task. (E.g., if a capacity variable is red, then this will prefer loads/unloads that use the actual capacity instead of an arbitrary one.) In some special cases, non-active red outside conditions can be easily fixed by inserting additional supporting actions.

Domain		Coverage							Evaluations hFF/Own						DLS	hFF/DLS		Coverage DL			Eval hFF/DL		
		hFF	K12	K13	EAS	ELS	DAS	DLS	K12	K13	EAS	ELS	DAS	DLS	Init	Plan	Time	-S	-(1,2)	-(2)	-S	-(1,2)	-(2)
Barman	20	17	18	13	**20**	**20**	**20**	**20**	3.4	1.6	6	67.8	6	67.8	0	0.9	56.8	**20**	**20**	**20**	6	67.8	67.8
Driverlog	20	20	20	20	20	20	20	20	1.3	2	1	1.1	1.1	1	1	1	1	19	**20**	**20**	0.9	1	1
Elevators	20	**20**	18	17	**20**	18	**20**	**20**	1.2	1.5	1.2	1.6	1.5	5920	20	1.1	15.5	18	**20**	**20**	1.4	2911	5920
Gripper	20	20	20	20	20	20	20	20	1	0.7	4.2	1	344	344	20	1	1	20	20	20	3.7	344	344
Rovers	29	29	29	29	29	29	29	29	1.1	1.2	1.2	1.2	1.4	1.4	1	1	0.8	29	29	29	1.1	1.4	1.4
Tidybot	20	13	10	**16**	12	13	12	13	2.2	1.3	1.2	1.1	1.2	1.1	0	1	0.8	12	13	12	1	1.1	1.1
Transport	20	10	10	10	11	11	**20**	**20**	0.6	1.2	1	0.9	3.4	3071	20	1.5	8.3	15	16	**20**	0.8	33.2	3071
Trucks	30	**19**	15	14	18	18	18	18	0.5	0.9	1.1	1.1	0.5	0.5	0	1	0.5	18	18	18	0.5	0.5	0.5
Sum	179	148	140	139	150	149	159	**160**							62			151	156	159			

Table 1: Experiments results. Ratios: median over instances solved by both planners involved. Explanations see text.

Experiments

The experiments were run on Intel Xeon CPU E5-2660 machines, with time (memory) limits of 30 minutes (2 GB). We ran *all IPC STRIPS benchmark instances whose causal graphs have at least one directed arc (v, v') between RSE-invertible variables v and v', with no backwards arc (v', v)*. These are exactly the tasks for which there exists a choice of black variables so that (a) the resulting red-black planning task is inside the tractable fragment, and (b) the black causal graph is a non-arc-empty DAG. The domains/instances where that happens are as shown in Table 1. For IPC'08 domains also used in IPC'11, we used only the IPC'11 version. For simplicity, we consider uniform costs throughout (i. e., we ignore action costs where specified).

We compare our **D**AG heuristics against Katz and Hoffmann's (2013) arc-**E**mpty ones, and against two variants of Keyder et al.'s (2012) partial delete relaxation heuristics: **K12** is best in their published experiments, **K13** is best in more recent (yet unpublished) experiments. **S** stops the search if a red-black plan works for the original planning task. Our baseline is the **hFF** heuristic implemented in Fast Downward. All configurations run greedy best-first search with lazy evaluation and a second open list for states resulting from preferred operators (Helmert 2006). All red-black heuristics return the same preferred operators as **hFF**: This enhances comparability; we found that changing the preferred operators was typically not beneficial anyway.

Katz and Hoffmann explored a variety of *painting strategies*, i. e., strategies for selecting the black variables. We kept this simple here because, as we noticed, there actually is little choice, at least when accepting the rationale that we should paint black as many variables as possible: In all our domains except Tidybot, there are at most 2 possible paintings per task. To illustrate, consider Figure 1: We can paint T and F black, or paint T and the packages black. All other paintings either do not yield a DAG black causal graph, or are not set-inclusion maximal among such paintings. We thus adopted only Katz and Hoffmann's 3 basic strategies, ordering the variables either by incident arcs (**A**), or by conflicts (**C**), or by causal graph level (**L**), and iteratively painting variables red until the black causal graph is a DAG (Katz and Hoffmann's original strategies continue until that graph is arc-**E**mpty). **A** works best for Katz and Hoffmann's heuristics (both here and in their experiments), and **L** works best for ours, so we show data for these two.

Consider Table 1 from left to right. Our red-black DAG heuristics have the edge in coverage, thanks to excelling in Transport and being reliably good across these domains. For search space size (number of state evaluations, i. e., calls to the heuristic function), there are smallish differences in half of the domains, and huge differences in the other half: Under the **L** painting strategy, in Barman the arc-**E**mpty heuristic already does well, and in Elevators, Gripper, and Transport our new **D**AG heuristic excels. A look at the "DLS Init" column, i. e., the number of instances solved by **S** in the initial state (without any search), shows that the latter 3 domains are exactly those where the superior applicability of DAG red-black plans makes the difference. Column "hFF/DLS Plan" shows that the plans found using **DLS** (even when constructed within the heuristic by **S**) are about as good as those found using **hFF**. Column "hFF/DLS Time" shows that the search space reductions do pay off in total runtime, except in Gripper (where all except **K12** and **K13** terminate in split seconds). The maximum speed-ups are 413 in Barman, 7.5 in Driverlog, 722 in Elevators, 12.9 in Tidybot, and 683 in Transport (none in the other domains); the maximum slow-down factors are 1.2 in Barman, 25 in Driverlog, 4.8 in Rovers, 4 in Tidybot, and 625 in Trucks.

The remainder of Table 1 sheds light on the contribution of stop search in **DLS**: **-S** switches stop search off, **-(1,2)** leaves it on but switches both applicability enhancements off, **-(2)** switches only enhancement (2) off.[3] We see that stop search helps even in domains (Driverlog, Tidybot) not solved directly in the initial state, and we see that the superior coverage in Transport is half due to the more informed heuristic, and half due to stop search with enhancement (1).

To further investigate the effect of stop search, we generated additional instances of the three domains that are fully solved directly in the initial state. Table 2 summarizes the results for additional 100 instances of increasing size in each of the three domains, namely Elevators, Gripper, and Transport. Focusing on our best performer, we compare **DLS** to both switching stop search off and our base **FF** heuristic. Note that **DLS** still solves all, even extremely large[4] instances directly in the initial state. Switching **S** off drastically reduces coverage without improving plan length. Comparing to **FF**, the picture is similar. Due to a large number of evaluations, even the much faster heuristic times out

[3] The data for **-(1)** is left out of Table 1 as it is equal to that for **-(1,2)**. Enhancement (2) has more impact in other (non-DAG) domains, especially NoMystery and Zenotravel, where **S** solves 8 respectively 19 more tasks directly with (2) than without it.

[4] Largest Gripper instance has 3942 balls.

		Coverage			Evals hFF/Own		Plan hFF/Own		Time hFF/Own	
		FF	DLS	DL	DLS	DL	DLS	DL	DLS	DL
Elevators	100	38	**100**	31	**10788.00**	1.28	**1.12**	1.06	**28.33**	0.63
Gripper	100	47	**100**	52	**148778.00**	68.59	1.00	1.00	**34.14**	0.75
Transport	100	30	**100**	26	**4150.00**	0.79	**1.44**	1.10	**13.78**	0.53
Sum	300	115	**300**	109	**10788.00**	1.28	**1.12**	1.06	**28.33**	0.63

Table 2: Experiments results. Coverage, evaluations, plan length, and total time for generated instances. Ratios: median over instances solved by both planners involved.

before finding a solution. Interestingly, in Gripper, **DL** has linear search space[5]. However, even in such case, due to the costly per node evaluation time of **DL**, large enough instances are not solved under the 30 minutes time bound.

Conclusion

Our work provides one more step on the road towards systematic interpolation between delete-relaxed and non-relaxed (real) planning. Our experience, as reflected by the presented experiments, suggests that the key advantage of heavy interpolation may lie in the ability to produce approximate plans that are very close to real plans (or that already *are* real plans). We believe that further progress will be achieved by developing search methods exploiting that ability in a targeted manner, for example by using partially relaxed plans to initialize plan-space searches (e. g., (Nguyen and Kambhampati 2001; Gerevini, Saetti, and Serina 2003)).

References

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

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

Cai, D.; Hoffmann, J.; and Helmert, M. 2009. Enhancing the context-enhanced additive heuristic with precedence constraints. In Gerevini, A.; Howe, A.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*, 50–57. AAAI Press.

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

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

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

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

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

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

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

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

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

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

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

Keyder, E.; Hoffmann, J.; and Haslum, P. 2012. Semi-relaxed plan heuristics. In Bonet, B.; McCluskey, L.; Silva, J. R.; and Williams, B., eds., *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*, 128–136. AAAI Press.

Nguyen, X., and Kambhampati, S. 2001. Reviving partial order planning. In Nebel, B., ed., *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI-01)*, 459–464. Seattle, Washington, USA: Morgan Kaufmann.

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

[5] So does FF heuristic when using enhanced hill climbing search and helpful actions in FF planning system.

Landmarks in Oversubscription Planning

Vitaly Mirkis and **Carmel Domshlak**
Technion, Haifa, Israel.
{mirkis@tx}{dcarmel@ie}.technion.ac.il

Abstract

In the basic setup of oversubscription planning (OSP), the objective is to achieve an as valuable as possible subset of goals within a fixed allowance of the total action cost (Smith 2004). Continuing from the recent successes in exploiting logical goal-reachability landmarks in classical planning, we develop a framework for exploiting such landmarks in heuristic-search OSP. We show how standard landmarks of certain classical planning tasks can be compiled into the OSP task of interest, resulting in an equivalent OSP task with a lower budget, and thus with a smaller search space. We then show how such landmark-based task enrichment can be combined in a mutually stratifying way with the *BFBB* search used for OSP planning. Our empirical evaluation confirms the effectiveness of the proposed landmark-based budget reduction scheme.

INTRODUCTION

In most general terms, deterministic planning is a problem of finding paths in large-scale yet concisely represented state-transition systems. In what these days is called classical planning (Fikes and Nilsson 1971), the task is to find an as *cost-effective* path as possible to a *goal-satisfying* state. In contrast, in what Smith (Smith 2004) baptized as "oversubscription" planning (OSP), the task is to find an as *goal-effective* (or *valuable*) state as possible via a *cost-satisfying* path. In other words, the hard constraint of classical planning translates to only preference in OSP, and the hard constraint of OSP translates to only preference in classical planning. Finally, in "optimal" classical planning and OSP, the tasks are further constrained to finding only *most* cost-effective paths and *most* goal-effective states, respectively.

Together, classical planning and OSP constitute the most fundamental variants of deterministic planning, with many other variants of deterministic planning being defined in terms of mixing and relaxing the two. For instance, "net-benefit" planning tries to achieve both (classical) cost-effectiveness of the path and (OSP) goal-effectiveness of the end-state by additively combining the two measures, but at the same time, it relaxes the hard constraints of (classical) goal-satisfaction and (OSP) cost-satisfaction (Sanchez and Kambhampati 2005; Baier, Bacchus, and McIlraith 2007; Bonet and Geffner 2008; Benton, Do, and Kambhampati 2009; Coles and Coles 2011; Keyder and Geffner 2009). Another popular setup is "cost-bounded" planning, in which both (classical) goal-satisfaction and (OSP) cost-satisfaction are pursued, but both (classical) cost-effectiveness of the path and (OSP) goal-effectiveness of the end-state are relaxed/ignored (Thayer and Ruml 2011; Thayer et al. 2012; Haslum 2013; Haslum and Geffner 2001; Hoffmann et al. 2007; Gerevini, Saetti, and Serina 2008; Nakhost, Hoffmann, and Müller 2012).

While OSP has been advocated over the years on par with classical planning, so far, the theory and practice of the latter have been studied and advanced much more intensively. The remarkable success and continuing progress of heuristic-search solvers for classical planning is one notable example. Primary enablers of this success are the advances in domain-independent approximations, or heuristics, of the cost needed to achieve a goal state from a given state. With our focus here on optimal planning, two classes of approximation techniques have been found especially useful in the context of optimal classical planning: those based on state-space *abstractions* (Edelkamp 2001; Haslum et al. 2007; Helmert, Haslum, and Hoffmann 2007; Katz and Domshlak 2010a) and these based on logical *landmarks* for goal reachability (Karpas and Domshlak 2009; Helmert and Domshlak 2009; Domshlak, Katz, and Lefler 2012; Bonet and Helmert 2010; Pommerening and Helmert 2013).

Considering OSP as heuristic search, a question is then whether some similar-in-spirit (yet possibly mathematically different) approximation techniques can be developed for heuristic-search OSP. Recently, the authors provided the first affirmative answer to this question in the context of abstractions by developing the actual notion of OSP abstractions, investigating the complexity of working with them for the purpose of heuristic approximation, and demonstrating empirically that using OSP abstraction heuristics within a best-first branch-and-bound (*BFBB*) search can be extremely effective in practice (Mirkis and Domshlak 2013). In contrast, the prospects of goal-reachability landmarks in heuristic-search OSP have not been investigated yet.

This is precisely the contribution of this paper: First, we introduce and study ε-landmarks, the logical properties of OSP plans that achieve valuable states. We show that ε-

Copyright © 2014, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

landmarks correspond to regular landmarks of certain classical planning tasks that can be (straightforwardly) derived from the OSP tasks of interest. We then show how such ε-landmarks can be compiled into the OSP task of interest, resulting in an equivalent OSP task, but with a stricter cost satisfaction constraint, and thus with a smaller effective search space. Finally, we show how such landmark-based task enrichment can be combined in a mutually stratifying way with the *BFBB* search used for OSP planning, resulting in an incremental procedure that interleaves search and landmark discovery. The entire framework is independent of the OSP planner specifics, and in particular, of the heuristic functions it employs. Our empirical evaluation on a large set of OSP tasks confirms the effectiveness of the proposed approach.

PRELIMINARIES

Since both OSP and classical planning tasks are discussed in the paper, we use a formalism that is based on the standard STRIPS formalism with non-negative operator costs (cf. (Helmert and Domshlak 2009)), extended to OSP in line with the notation of our earlier paper on OSP (Mirkis and Domshlak 2013).

Planning Tasks. A *planning task structure* is given by a pair $\langle V, O\rangle$, where V is a finite set of propositional state variables, and O is a finite set of operators. State variables are also called propositions or facts. A state $s \in 2^V$ is a subset of facts, representing the propositions which are currently true. Each operator $o \in O$ is associated with preconditions $\mathsf{pre}(o) \subseteq V$, add effects $\mathsf{add}(o) \subseteq V$, and delete effects $\mathsf{del}(o) \subseteq V$. Applying an operator o in s results in state $(s \setminus \mathsf{del}(o)) \cup \mathsf{add}(o)$, which we denote as $s[\![o]\!]$. The notation is only defined if o is applicable in s, i.e., if $\mathsf{pre}(o) \subseteq s$. Applying a sequence $\langle o_1, \ldots, o_k\rangle$ of operators to a state is defined inductively as $s[\![\epsilon]\!] := s$ and $s[\![\langle o_1, \ldots, o_k\rangle]\!] := (s[\![\langle o_1, \ldots, o_{k-1}\rangle]\!])[\![o_k]\!]$.

A *classical planning task* $\Pi = \langle V, O; I, G, cost\rangle$ extends its structure $\langle V, O\rangle$ with an initial state $I \subseteq V$, a goal $G \subseteq V$, and a real-valued, nonnegative operator cost function $cost : O \to \mathbb{R}^{0+}$. An operator sequence π is called an s-plan if it is applicable in s, and $G \subseteq s[\![\pi]\!]$. The cost of s-plan π is $cost(\pi) := \sum_{o\in\pi} cost(o)$, and π is optimal if its cost is minimal among all s-plans. The objective in classical planning is to find an I-plan of as low cost as possible or prove that no I-plan exists. Optimal classical planning is devoted to searching for optimal I-plans only.

An *oversubscription planning (OSP) task* $\Pi = \langle V, O; I, cost, u, b\rangle$ extends its structure $\langle V, O\rangle$ with four components: an initial state $I \subseteq V$ and an operator cost function $cost : O \to \mathbb{R}^{0+}$ as above, plus a succinctly represented and efficiently computable state value function $u : S \to \mathbb{R}^{0+}$, and a cost budget $b \in \mathbb{R}^{0+}$. In what follows, we assume $u(s) = \sum_{v\in s} u(v)$, i.e., the value of state s is the sum of (mutually independent) values of propositions which are true in s. Conceptually, our results equally apply to general value functions, but the complexity of certain construction steps may vary between different families of value functions.

In OSP, an operator sequence π is called an s-plan if it is applicable in s, and $\sum_{o\in\pi} cost(o) \leq b$. While even an empty operator sequence is an s-plan for any state s, the objective in OSP is to find an I-plan that achieves as valuable a state as possible. By $\widehat{u}(\pi)$ we refer to the value of the end-state of π, that is, $\widehat{u}(\pi) = u(s[\![\pi]\!])$. *Optimal OSP* is devoted to searching for optimal I-plans only: An s-plan π is optimal if $\widehat{u}(\pi)$ is maximal among all the s-plans.

Heuristics. The two major ingredients of any heuristic-search planner are its search algorithm and heuristic function. In classical planning, the heuristic is typically a function $h : 2^V \to \mathbb{R}^{0+} \cup \{\infty\}$, with $h(s)$ estimating the cost $h^*(s)$ of optimal s-plans. A heuristic h is admissible if it is *lower-bounding*, i.e., $h(s) \leq h^*(s)$ for all states s. All common heuristic search algorithms for optimal classical planning, such as A^*, require admissible heuristics.

In OSP, a heuristic is a function $h : 2^V \times \mathbb{R}^{0+} \to \mathbb{R}^{0+}$, with $h(s, b)$ estimating the value $h^*(s, b)$ of optimal s-plans under cost budget b. A heuristic h is admissible if it is *upper-bounding*, i.e., $h(s, b) \geq h^*(s, b)$ for all states s and all cost budgets b. Here as well, search algorithms for optimal OSP, such as best-first branch-and-bound (*BFBB*) discussed later on in detail, require admissible heuristics.

Landmarks in Classical Planning. For a state s in a classical planning task Π, a landmark is a property of operator sequences that is satisfied by all s-plans (Hoffmann, Porteous, and Sebastia 2004). For instance, a fact landmark for a state s is a fact that is true at some point in every s-plan. Several admissible landmark heuristics have been shown as extremely effective in optimal classical planning (Karpas and Domshlak 2009; Helmert and Domshlak 2009; Bonet and Helmert 2010; Pommerening and Helmert 2013). These heuristics use extended notions of landmarks which are subsumed by *disjunctive action landmarks*. Each such landmark is a set of operators such that every s-plan contains at least one action from that set. In what follows we consider this popular notion of landmarks, and simply refer to disjunctive action landmarks for a state s as s-landmarks. For ease of presentation, most of our discussion will take place in the context of landmarks for the initial state of the task, and these will simply be referred to as *landmarks (for* Π*)*.

Deciding whether an operator set $L \subset O$ is a landmark for classical planning task Π is PSPACE-hard (Porteous, Sebastia, and Hoffmann 2001). Therefore, all landmark heuristics employ methods for landmark discovery that are polynomial-time, sound, but incomplete. In what follows we assume access to such a procedure; the actual way the landmarks are discovered is tangential to our contribution. For a set $\mathcal{L}$ of s-landmarks, a *landmark cost function* $lcost : \mathcal{L} \to \mathbb{R}^{0+}$ is admissible if $\sum_{L\in\mathcal{L}} lcost(L) \leq h^*(s)$. For a singleton set $\mathcal{L} = \{L\}$, $lcost(L) := \min_{o\in L} cost(o)$ is a natural admissible landmark cost function, and it extends directly to non-singleton sets of pairwise disjoint landmarks. For more general sets of landmarks, $lcost$ can be devised (in polynomial time) via operator cost partitioning (Katz and Domshlak 2010b), either given $\mathcal{L}$ (Karpas and Domshlak 2009), or within the actual process of generating $\mathcal{L}$ (Helmert and Domshlak 2009).

"BRING ME SOMETHING" LANDMARKS

While landmarks play an important role in (both satisficing and optimal) classical planning, so far they have not been exploited in OSP. At first glance, this is probably no surprise: Since landmarks must hold in all plans, and the empty operator sequence is always a plan for any OSP task, the notion of landmark does not seem useful here.

Having said that, consider the anytime "output improvement" property of the forward-search branch-and-bound algorithms used for heuristic-search OSP. The empty plan is not interesting there not only because it is useless, but also because it is "found" by the search algorithm right at the get-go. In general, at all stages of the search, anytime algorithms like *BFBB* maintain the best-so-far solution π, and prune all branches that promise value lower or equal to $\widehat{u}(\pi)$. Hence, in principle, such algorithms may benefit from information about properties that are "satisfied by all plans with value larger than x." Unfortunately, it is not yet clear how the machinery for discovering classical planning landmarks can be adapted to discovery of such "value landmarks" while preserving polynomial-time complexity on general OSPs and arbitrary lower bounds x.

Looking at what is needed and what is available, our goal here is to exploit this machinery as it is. While the value of different s-plans in an OSP task Π varies between zero and the value of the optimal s-plan (which may also be zero), let an *ε-landmark* for state s be any property that is satisfied by any s-plan π that achieves *something valuable*. For instance, with the disjunctive action landmarks we use here, if $L \subseteq O$ is an ε-landmark for s, then every s-plan π with $\widehat{u}(\pi) > 0$ contains an operator from L. In what follows, unless stated otherwise, we focus on ε-landmarks for (the initial state of) Π.

Given an OSP task $\Pi = \langle V, O; I, cost, u, b\rangle$, let a classical planning task $\Pi_\varepsilon = \langle V_\varepsilon, O_\varepsilon; I_\varepsilon, cost_\varepsilon, G_\varepsilon\rangle$ be constructed as $V_\varepsilon = V \cup \{g\}$, $I_\varepsilon = I$, $G_\varepsilon = \{g\}$, and $O_\varepsilon = O \cup O_g$, where, for each proposition v with $u(v) > 0$, O_g contains an operator o_v with $\mathsf{pre}(o_v) = \{v\}$, $\mathsf{add}(o_v) = \{g\}$, $\mathsf{del}(o_v) = \emptyset$, and $cost_\varepsilon(o_v) = 0$. For all the original operators $o \in O$, $cost_\varepsilon(o) = cost(o)$. In other words, Π_ε extends the structure of Π with a set of zero-cost actions such that applying any of them indicates achieving a positive value in Π. In what follows, we refer to Π_ε as the *ε-compilation* of Π. Constructing Π_ε from Π is trivially polynomial time, and it allows us to discover ε-landmarks for Π using the standard machinery for classical planning landmark discovery.

Theorem 1 *For any OSP task Π, any landmark L for Π_ε such that $L \subseteq O$ is an ε-landmark for Π.*

With Theorem 1 in hand,[1] we can now derive ε-landmarks for Π using any method for classical planning landmark extraction, such as that employed by the LAMA planner (Richter, Helmert, and Westphal 2008) or the LM-Cut family of techniques (Helmert and Domshlak 2009; Bonet and Helmert 2010). However, at first glance, the discriminative power of knowing "what is needed to achieve *something* valuable" seems to be negligible when it comes to deriving effective heuristic estimates for OSP. The good news is that, in OSP, such information can be effectively exploited in a slightly different way.

[1] Due to space limitations, all proofs are delegated to a full technical report (Mirkis and Domshlak 2014).

BFBB ($\Pi = \langle V, O; I, cost, u, b\rangle$)
  open := **new** max-heap ordered by $f(n) = h(s[n], b - g(n))$
  open.insert(make-root-node(I))
  closed:= $\emptyset$; best-cost:= $\emptyset$
  initialize best solution $n^* := I$
  while not open.empty()
    n := open.pop-max()
    if $f(n) \leq u(s[n^*])$: **break**
    if $s[n] \notin$ closed **or** $g(n) <$ best-cost($s[n]$):
      closed:= closed $\cup \{s[n]\}$
      best-cost($s[n]$) := $g(n)$
      foreach $o \in O(s[n])$:
        n' := make-node($s[n][\![o]\!]$)
        if $g(n') > b$ **or** $f(n') \leq u(s[n^*])$: **continue**
        if $u(s[n']) > u(s[n^*])$: update $n^* := n'$
        open.insert(n')
  return n^*

Figure 1: Best-first branch-and-bound (*BFBB*) search for OSP

ε-Landmarks and Budget Reduction

In the same way that A^* constitutes a canonical heuristic-search algorithm for optimal classical planning, anytime *best-first branch-and-bound (BFBB)* probably constitutes such an algorithm for optimal OSP.[2] Figure 1 depicts a pseudo-code description of *BFBB*. $s[n]$ there denotes the state associated with search node n. In *BFBB* for OSP, a node n with maximum evaluation function $h(s[n], b - g(n))$ is selected from the OPEN list. The duplicate detection and reopening mechanisms in *BFBB* are similar to those in A^*. In addition, *BFBB* maintains the best solution n^* found so far and uses it to prune all generated nodes evaluated no higher than $u(s[n^*])$. Likewise, complying with the semantics of OSP, all generated nodes n with cost-so-far $g(n)$ higher than the problem's budget b are also immediately pruned. When the OPEN list becomes empty or the node n selected from the list promises less than the lower bound, *BFBB* returns (the plan associated with) the best solution n^*, and if h is admissible, i.e., the h-based pruning of the generated nodes is sound, then the returned plan is guaranteed to be optimal.

Now, consider a schematic example of searching for an optimal plan for an OPS task Π with budget b, using *BFBB* with an admissible heuristic h. Suppose that there is only one sequence of (all unit-cost) operators, $\pi = \langle o_1, o_2, \ldots, o_{b+1}\rangle$, applicable in the initial state of Π, and that the only positive value state along π is its end-state. While clearly no value higher than zero can be achieved

[2] *BFBB* is also extensively used for net-benefit planning (Benton, van den Briel, and Kambhampati 2007; Coles and Coles 2011; Do et al. 2007), as well as some other variants of deterministic planning (Bonet and Geffner 2008; Brafman and Chernyavsky 2005).

in Π under the given budget of b, the search will continue beyond the initial state, unless $h(I,\cdot)$ counts the cost of all the $b+1$ actions of π. Now, suppose that $h(I,\cdot)$ counts only the cost of $\{o_i,\ldots,o_{b+1}\}$ for some $i>0$, but $\{o_1\},\{o_2\},\ldots,\{o_{i-1}\}$ are all discovered to be ε-landmarks for Π. Given that, suppose that we modify Π by (a) setting the cost of operators $o_1,o_2,\ldots,o_{i-1}$ to zero, and (b) *reducing the budget* to $b-i+1$. This modification seems to preserve the semantics of Π, while on the modified task, *BFBB* with the same heuristic h will prune the initial state and thus establish without any search that the empty plan is an optimal plan for Π. Of course, the way Π is modified in this example is as simplistic as the example itself. Yet, this example does motivate the idea of *landmark-based budget reduction* for OSP, as well as illustrates the basic idea behind the *generically sound* task modifications that we discuss next.

Let $\Pi=\langle V,O;I,cost,u,b\rangle$ be an OSP task, $\mathcal{L}=\{L_1,\ldots,L_n\}$ be a set of pairwise disjoint ε-landmarks for Π, and $lcost$ be an admissible landmark cost function from $\mathcal{L}$. Given that, a new OSP task $\Pi_\mathcal{L}=\langle V_\mathcal{L},O_\mathcal{L};I_\mathcal{L},cost_\mathcal{L},u_\mathcal{L},b_\mathcal{L}\rangle$ with budget $b_\mathcal{L}=b-\sum_{i=1}^n lcost(L_i)$ is constructed as follows. The set of variables $V_\mathcal{L}=V\cup\{v_{L_1},\ldots,v_{L_n}\}$ extends V with a new proposition per ε-landmark in $\mathcal{L}$. These new propositions are all initially true, and $I_\mathcal{L}=I\cup\{v_{L_1},\ldots,v_{L_n}\}$. The value function $u_\mathcal{L}=u$ remains unchanged—the new propositions do not affect the value of the states. Finally, the operator set is extended as $O_\mathcal{L}=O\cup\bigcup_{i=1}^n O_{L_i}$, with O_{L_i} containing an operator $\overline{o}$ for each $o\in L_i$, with $\mathsf{pre}(\overline{o})=\mathsf{pre}(o)\cup\{v_{L_i}\}$, $\mathsf{add}(\overline{o})=\mathsf{add}(o)$, $\mathsf{del}(\overline{o})=\mathsf{del}(o)\cup\{v_{L_i}\}$, and, importantly, $cost_\mathcal{L}(\overline{o})=cost(o)-lcost(L_i)$. In other words, $\Pi_\mathcal{L}$ extends the structure of Π by mirroring the operators of each ε-landmark L_i with their "$lcost(L_i)$ cheaper" versions, while ensuring that these cheaper operators can be applied no more than once along an operator sequence from the initial state. At the same time, introduction of these discounted operators for L_i is compensated for by reducing the budget by precisely $lcost(L_i)$, leading to effective equivalence between Π and $\Pi_\mathcal{L}$.

Theorem 2 *Let $\Pi=\langle V,O;I,cost,u,b\rangle$ be an OSP task, $\mathcal{L}$ be a set of pairwise disjoint ε-landmarks for Π, $lcost$ be an admissible landmark cost function from $\mathcal{L}$, and $\Pi_\mathcal{L}$ be the respective budget reducing compilation of Π. For every π for Π with $\widehat{u}(\pi)>0$, there is a plan $\pi_\mathcal{L}$ for $\Pi_\mathcal{L}$ with $\widehat{u}(\pi_\mathcal{L})=\widehat{u}(\pi)$, and vice versa.*

The above budget reducing compilation of Π to $\Pi_\mathcal{L}$ is clearly polynomial time. Putting things together, we can see that the compile-and-BFBB procedure depicted in Figure 2 (1) generates an ε-compilation Π_ε of Π, (2) uses off-the-shelf tools for classical planning to generate a set of landmarks $\mathcal{L}$ for Π_ε and an admissible landmark cost function $lcost$, and (3) compiles $(\mathcal{L},lcost)$ into Π, obtaining an OSP task $\Pi_\mathcal{L}$. The optimal solution for $\Pi_\mathcal{L}$ (and thus for Π) is then searched for using a search algorithm for optimal OSP such as *BFBB*.

Before we proceed to consider more general sets of landmarks, a few comments concerning the setup of Theorem 2

compile-and-BFBB ($\Pi=\langle V,O;I,cost,u,b\rangle$)
  Π_ε = ε-compilation of Π
  $\mathcal{L}$:= a set of landmarks for Π_ε
  $lcost$:= admissible landmark cost function for $\mathcal{L}$
  $\Pi_{\mathcal{L}^*}$:= budget reducing compilation of $(\mathcal{L},lcost)$ into Π
  n^* := BFBB($\Pi_{\mathcal{L}^*}$)
  return plan for Π associated with n^*

Figure 2: *BFBB* search with landmark-based budget reduction

are now probably in place. First, if the reduced budget $b_\mathcal{L}$ turns out to be lower than the cost of the cheapest action applicable in the initial state, then no search is needed, and the empty plan can be reported as optimal right away. Second, zero-cost landmarks are useless in our compilation as much as they are useless in deriving landmark heuristics for optimal planning. Hence, $lcost$ in what follows is assumed to be strictly positive. Third, having both o and $\overline{o}$ applicable at a state of Π_ε brings no benefits yet adds branching to the search. Hence, in our implementation, for each landmark $L_i\in\mathcal{L}$ and each operator $o\in L_i$, the precondition of o in $O_\mathcal{L}$ is extended with $\{\neg v_{L_i}\}$. It is not hard to verify that this extension[3] preserves the correctness of $\Pi_\mathcal{L}$ in terms of Theorem 2. Finally, if the value of the initial state is not zero, that is, the empty plan has some positive value, then ε-compilation Π_ε of Π will have no positive cost landmarks at all. However, this can easily be fixed by considering as "valuable" only facts v such that both $u(v)>0$ and $v\notin I$. For now we put this difficulty aside and assume that $\widehat{u}(\epsilon)=0$. Later, however, we come back to consider it more systematically.

Non-Disjoint ε-Landmarks

While the compilation $\Pi_\mathcal{L}$ above is sound for pairwise disjoint landmarks, this is not so for more general sets of ε-landmarks. For example, consider a planning task Π in which, for some operator o, we have $cost(o)=b$, $\widehat{u}(\langle o\rangle)>0$, and $\widehat{u}(\pi)=0$ for all other operator sequences $\pi\neq\langle o\rangle$. That is, a value greater than zero is achievable in Π, but only via the operator o. Suppose now that our set of ε-landmarks for Π is $\mathcal{L}=\{L_1,\ldots,L_n\}$, $n>1$, and that all of these ε-landmarks contain o. In this case, while the budget in $\Pi_\mathcal{L}$ is $b_\mathcal{L}=b-\sum_{i=1}^n lcost(L_i)$, the cost of the cheapest replica $\overline{o}$ of o, that is, the cost of the cheapest operator sequence achieving a non-zero value in Π, is $cost(o)-\min_{i=1}^n lcost(L_i)>b_\mathcal{L}$. Hence, no state with positive value will be reachable from $I_\mathcal{L}$ in $\Pi_\mathcal{L}$, and thus Π and $\Pi_\mathcal{L}$ are not "value equivalent" in the sense of Theorem 2.

Since non-disjoint landmarks can bring more information, and they are typical to outputs of standard techniques for landmark extraction in classical planning, we now present a different, slightly more involved, compilation that is both polynomial and sound for arbitrary sets of ε-landmarks. Let $\Pi=\langle V,O;I,cost,u,b\rangle$ be an OSP task, $\mathcal{L}=\{L_1,\ldots,L_n\}$ be a set of ε-landmarks for Π, and $lcost$ be an admissible landmark cost function from $\mathcal{L}$. For

[3]This modification requires augmenting our STRIPS-like formalism with negative preconditions, but this augmentation is straightforward.

each operator o, let $\mathcal{L}(o)$ denote the set of all landmarks in $\mathcal{L}$ that contain o. Given that, a new OSP task $\Pi_{\mathcal{L}^*} = \langle V_{\mathcal{L}^*}, O_{\mathcal{L}^*}; I_{\mathcal{L}^*}, cost_{\mathcal{L}^*}, u_{\mathcal{L}^*}, b_{\mathcal{L}^*}\rangle$ is constructed as follows. Similarly to $\Pi_{\mathcal{L}}$, we have $b_{\mathcal{L}^*} = b - \sum_{i=1}^{n} lcost(L_i)$, $V_{\mathcal{L}^*} = V \cup \{v_{L_1}, \ldots, v_{L_n}\}$, $I_{\mathcal{L}^*} = I \cup \{v_{L_1}, \ldots, v_{L_n}\}$, and $u_{\mathcal{L}^*} = u$. The operator set $O_{\mathcal{L}^*}$ extends O with two sets of operators:

- For each operator $o \in O$ that participates in some landmark from $\mathcal{L}$, $O_{\mathcal{L}^*}$ contains an action $\overline{o}$ with $\mathsf{pre}(\overline{o}) = \mathsf{pre}(o) \cup \{v_L \mid L \in \mathcal{L}(o)\}$, $\mathsf{add}(\overline{o}) = \mathsf{add}(o)$, $\mathsf{del}(\overline{o}) = \mathsf{del}(o) \cup \{v_L \mid L \in \mathcal{L}(o)\}$, $cost_{\mathcal{L}^*}(\overline{o}) = cost(o) - \sum_{L \in \mathcal{L}(o)} lcost(L)$.
- For each $L \in \mathcal{L}$, $O_{\mathcal{L}^*}$ contains an action $get(L)$ with $\mathsf{pre}(get(L)) = \{\neg v_L\}$, $\mathsf{add}(get(L)) = \{v_L\}$, $\mathsf{del}(get(L)) = \emptyset$, $cost_{\mathcal{L}^*}(get(L)) = lcost(L)$.

For example, let $\mathcal{L} = \{L_1, L_2, L_3\}$, $L_1 = \{a, b\}$, $L_2 = \{b, c\}$, $L_3 = \{a, c\}$, with all operators having the cost of 2, and let $lcost(L_1) = lcost(L_2) = lcost(L_3) = 1$. In $\Pi_{\mathcal{L}^*}$, we have $V_{\mathcal{L}^*} = V \cup \{v_{L_1}, v_{L_2}, v_{L_3}\}$ and $O_{\mathcal{L}^*} = O \cup \{\overline{a}, \overline{b}, \overline{c}, get(L_1), get(L_2), get(L_3)\}$, with, e.g., $\mathsf{pre}(\overline{a}) = \mathsf{pre}(a) \cup \{v_{L_1}, v_{L_3}\}$, $\mathsf{add}(\overline{a}) = \mathsf{add}(a)$, $\mathsf{del}(\overline{a}) = \mathsf{del}(a) \cup \{v_{L_1}, v_{L_3}\}$, and $cost_{\mathcal{L}^*}(\overline{a}) = 0$, and, for $get(L_1)$, $\mathsf{pre}(get(L_1)) = \mathsf{del}(get(L_1)) = \emptyset$, $\mathsf{add}(get(L_1)) = \{v_{L_1}\}$, and $cost_{\mathcal{L}^*}(get(L_1)) = 1$.

Theorem 3 *Let $\Pi = \langle V, O; I, cost, u, b\rangle$ be an OSP task and $\Pi_{\mathcal{L}^*}$ a budget reducing compilation of Π. For every π for Π with $\widehat{u}(\pi) > 0$, there is a plan $\pi_{\mathcal{L}^*}$ for $\Pi_{\mathcal{L}^*}$ with $\widehat{u}(\pi_{\mathcal{L}^*}) = \widehat{u}(\pi)$, and vice versa.*

ε-LANDMARKS & INCREMENTAL *BFBB*

As we discussed earlier, if the value of the initial state is not zero, i.e., the empty plan has some positive value, then the basic ε-compilation Π_ε of Π will have no positive cost landmarks at all. In passing we noted that this small problem can be remedied by considering as "valuable" only facts v such that both $u(v) > 0$ and $v \notin I$. We now consider this aspect of OSP more closely, and show how ε-landmarks discovery and incremental revelation of plans by *BFBB* can be combined in a mutually stratifying way.

Let $\Pi = \langle V, O; I, cost, u, b\rangle$ be the OSP task of our interest, and suppose we are *given* a set of plans $\pi_1, \ldots, \pi_n$ for Π. If so, then we are no longer interested in searching for plans that "achieve something," but in searching for plans that achieve *something beyond what $\pi_1, \ldots, \pi_n$ already achieve*. For $1 \leq i \leq n$, let $s_i = I[\![\pi_i]\!]$ be the end-state of π_i, and for any set of propositions $s \subseteq V$, let $\mathsf{goods}(s) \subseteq s$ be the set of all facts $v \in s$ such that $u(v) > 0$. If a new plan π with end-state s achieves something beyond what $\pi_1, \ldots, \pi_n$ already achieve, then $\mathsf{goods}(s) \setminus \mathsf{goods}(s_i) \neq \emptyset$ for *all* $1 \leq i \leq n$.

We now put this observation to work. Given an OSP task $\Pi = \langle V, O; I, cost, u, b\rangle$ and a set of reference states $S_{\text{ref}} = \{s_1, \ldots, s_n\}$ of Π, let a classical planning task $\Pi_{(\varepsilon, S_{\text{ref}})} = \langle V_\varepsilon, O_\varepsilon; I_\varepsilon, G_\varepsilon, cost_\varepsilon\rangle$ be constructed as follows. The variable set $V_\varepsilon = V \cup \{x_1, \ldots, x_n, search, collect\}$ extends V with a new proposition per state in S_{ref}, plus two auxiliary control variables. In the initial state, all the new variables but *search* are false, i.e., $I_\varepsilon = I \cup \{search\}$, and the goal is

inc-compile-and-BFBB ($\Pi = \langle V, O; I, cost, u, b\rangle$)
  initialize global variables:
    $n^* := I$ *// best solution so far*
    $S_{\text{ref}} := \{I\}$ *// current reference states*
  loop:
    $\Pi_{(\varepsilon, S_{\text{ref}})}$ = $(\varepsilon, S_{\text{ref}})$-compilation of Π
    $\mathcal{L}$:= a set of landmarks for $\Pi_{(\varepsilon, S_{\text{ref}})}$
    $lcost$:= admissible landmark cost function from $\mathcal{L}$
    $\Pi_{\mathcal{L}^*}$:= budget reducing compilation of $(\mathcal{L}, lcost)$ into Π
    if inc-BFBB($\Pi_{\mathcal{L}^*}, S_{\text{ref}}, n^*$) = *done*:
      return plan for Π associated with n^*

inc-BFBB (Π, S_{ref}, n^*)
  open := **new** max-heap ordered by $f(n) = h(s[n], b - g(n))$
  open.insert(make-root-node(I))
  closed:= $\emptyset$ best-cost:= $\emptyset$;
  while not open.empty()
    n := open.pop-max()
    if $\mathsf{goods}(s[n]) \not\subseteq \mathsf{goods}(s')$ for all $s' \in S_{\text{ref}}$:
      $S_{\text{ref}} := S_{\text{ref}} \cup \{s[n]\}$
      if termination criterion: **return** *updated*
    if $f(n) \leq u(s[n^*])$: **break**
      // . . .
      // similar to BFBB *in Figure 1*
  return *done*

Figure 3: Iterative *BFBB* with landmark enhancement

$G_\varepsilon = \{x_1, \ldots, x_n\}$. The operator set O_ε contains three sets of operators: First, each operator $o \in O$ is represented in O_ε by an operator $\overline{o}$, with the only difference between o and $\overline{o}$ (including cost) being that $\mathsf{pre}(\overline{o}) = \mathsf{pre}(o) \cup \{search\}$. We denote this set of new operators $\overline{o}$ by $\overline{O}$. Second, for each $s_i \in S_{\text{ref}}$ and each value-carrying fact g that is *not* in s_i, i.e., for each $g \in \mathsf{goods}(V) \setminus s_i$, O_ε contains a zero-cost action $o_{i,g}$ with $\mathsf{pre}(o_{i,g}) = \{g, collect\}$, $\mathsf{add}(o_{i,g}) = \{x_i\}$, $\mathsf{del}(o_{i,g}) = \emptyset$. Finally, O_ε contains a zero-cost action *finish* with $\mathsf{pre}(finish) = \emptyset$, $\mathsf{del}(finish) = \{search\}$, and $\mathsf{add}(finish) = \{collect\}$.

It is easy to verify that (1) the goal G_ε cannot be achieved without applying the *finish* operator, (2) the operators $\overline{o}$ can be applied only before *finish*, and (3) the subgoal achieving operators $o_{i,g}$ can be applied only after *finish*. Hence, the first part of any plan for $\Pi_{(\varepsilon, S_{\text{ref}})}$ determines a plan for Π, and the second part "verifies" that the end-state of that plan achieves a subset of value-carrying propositions $\mathsf{goods}(V)$ that is included in no state from S_{ref}.[4]

Theorem 4 *Let $\Pi = \langle V, O; I, cost, u, b\rangle$ be an OSP task, $S_{ref} = \{s_1, \ldots, s_n\} \subseteq 2^V$ be a subset of Π's states, and L be a landmark for $\Pi_{(\varepsilon, S_{ref})}$ such that $L \subseteq \overline{O}$. For any plan π for Π such that $\mathsf{goods}(I[\![\pi]\!]) \setminus \mathsf{goods}(s_i) \neq \emptyset$ for all $s_i \in S_{ref}$, π contains an instance of at least one operator from $L' = \{o \mid \overline{o} \in L\}$.*

Theorem 4 allows us to define an iterative version of *BFBB*, successive iterations of which correspond to running the regular *BFBB* on successively more informed $(\varepsilon, S_{\text{ref}})$-compilations of Π, with the states discovered at iteration i

[4]This "plan in two parts" technique appears to be helpful in many planning formalism compilations; see, e.g., (Keyder and Geffner 2009).

making the $(\varepsilon, S_{\text{ref}})$-compilation used at iteration $i+1$ more informed. The respective procedure inc-compile-and-BFBB is depicted in Figure 3. This procedure maintains a set of reference states S_{ref} and the best solution so far n^*, and loops over calls to inc-BFBB, a modified version of *BFBB*. At each iteration of the loop, inc-BFBB is called with an $(\varepsilon, S_{\text{ref}})$-compilation of Π, created on the basis of the *current* S_{ref} and n^*, and it is provided with access to both S_{ref} and n^*. The reference set S_{ref} is then extended by inc-BFBB with all the non-redundant value-carrying states discovered during the search, and n^* is updated if the search discovers nodes of higher value.

If and when the OPEN list becomes empty or the node n selected from the list promises less than the lower bound, inc-BFBB returns an indicator, $done$, that the best solution n^* found so far, across the iterations of inc-compile-and-BFBB, is optimal. In that case, inc-compile-and-BFBB leaves its loop and extracts that optimal plan from n^*. However, inc-BFBB may also terminate in a different way, if a certain complementary termination criterion is satisfied. The latter criterion comes to assess whether the updates to S_{ref} performed in the current session of *BFBB* warrant updating the $(\varepsilon, S_{\text{ref}})$-compilation and restarting the search.[5] If terminated this way, inc-BFBB returns a respective indicator, and inc-compile-and-BFBB goes into another iteration of its loop, with the *updated* S_{ref} and n^*.

EMPIRICAL EVALUATION

We have implemented a prototype heuristic-search OSP solver on top of the Fast Downward planner (Helmert 2006). The implementation included[6]:

- $(\varepsilon, S_{\text{ref}})$-compilation of OSP tasks Π for arbitrary sets of reference states S_{ref};
- Generation of disjunctive action landmarks for $(\varepsilon, S_{\text{ref}})$-compilations using the LM-Cut procedure (Helmert and Domshlak 2009) of Fast Downward;
- The incremental *BFBB* procedure inc-compile-and-BFBB from the previous section, with the search termination criterion being satisfied (only) if the examined node n improves over current value lower bound; and
- An additive abstraction heuristic from the framework of Mirkis and Domshlak (Mirkis and Domshlak 2013), incorporating (i) an ad hoc action cost partition over k projections of the planning task onto connected subsets of ancestors of the respective k goal variables in the causal graph, and (ii) a value partition that associates the value of each goal (only) with the respective projection. The size of each projection was limited to 1000 abstract states.

After some preliminary evaluation, we also added two (optimality preserving) enhancements to the search. First, the auxiliary variables of our compilations increase the dimensionality of the problem, and this is well known to negatively affect the quality of the projection abstractions. Hence, we devised the projections with respect to the *original* OSP problem Π, and the open list was ordered as if the search is done on the original problem, that is, by $h(s[n]^{\downarrow V},\ b - g(n) + \sum_{v_L \notin s[n]} lcost(L))$, where $s[n]^{\downarrow V}$ is the projection of the state $s[n]$ on the variables of the original OSP task Π. This change in heuristic evaluation is sound, as Theorem 3 in particular implies that any admissible heuristic for Π is also an admissible heuristic for $\Pi_{\mathcal{L}^*}$, and vice versa. Second, when a new node n is generated, we check whether $g(n) + \sum_{v_L \notin s[n]} lcost(L) \ \geq\ g(n') + \sum_{v_L \notin s[n']} lcost(L)$, for some previously generated node n' that corresponds to the same state of the original problem Π, i.e., $s[n']^{\downarrow V} = s[n]^{\downarrow V}$. If so, then n is pruned right away. Optimality preservation of this enhancement is established in (Mirkis and Domshlak 2014).

Since, unlike classical and net-benefit planning, OSP lacks standard benchmarks for comparative evaluation, we have cast in this role the STRIPS classical planning domains from the International Planning Competitions (IPC) 1998-2006. This "translation" to OSP was done by associating a separate unit-value with each sub-goal. The evaluation included the regular *BFBB* planning for Π, solving Π using landmark-based compilation via compile-and-BFBB, and the straightforward setting of inc-compile-and-BFBB described above. All three approaches were evaluated under the blind heuristic and the additive abstraction heuristic as above.

Figure 4 depicts the results of our evaluation in terms of expanded nodes on all the aforementioned IPC tasks for which we could determine offline the minimal cost needed to achieve all the goals in the task. Each task was approached under four different budgets, corresponding to 25%, 50%, 75%, and 100% of the minimal cost needed to achieve all the goals in the task, and each run was restricted to 10 minutes. Figures 4(a) and 4(b) compare the performance of *BFBB* and compile-and-BFBB with blind (a) and abstraction (b) heuristics. Figures 4(c) and 4(d) provide a similar comparison between *BFBB* and inc-compile-and-BFBB. [7]

As Figure 4 shows, the results are satisfactory. With no informative heuristic guidance at all, the number of nodes expanded by compile-and-BFBB was typically much lower than the number of nodes expanded by *BFBB*, with the difference reaching three orders of magnitude more than once. Of the 760 task/budget pairs behind Figure 4a, 81 pairs were solved by compile-and-BFBB with no search at all (by proving that no plan can achieve value higher than that of the initial state), while, unsurprisingly, only 4 of these tasks were solved with no search by *BFBB*.

As expected, the value of landmark-based budget reduction is lower when the search is equipped with a meaningful heuristic (Figure 4b). Yet, even with our abstraction heuristic in hand, the number of nodes ex-

[5] While the optimality of the algorithm holds for *any* such termination condition, the latter should greatly affect the runtime efficiency of the algorithm.

[6] We are not aware of any other domain-independent planner for optimal OSP.

[7] We do not compare here between the running times, but the per-node CPU time overhead due to landmark-based budget reduction was $\leq 10\%$.

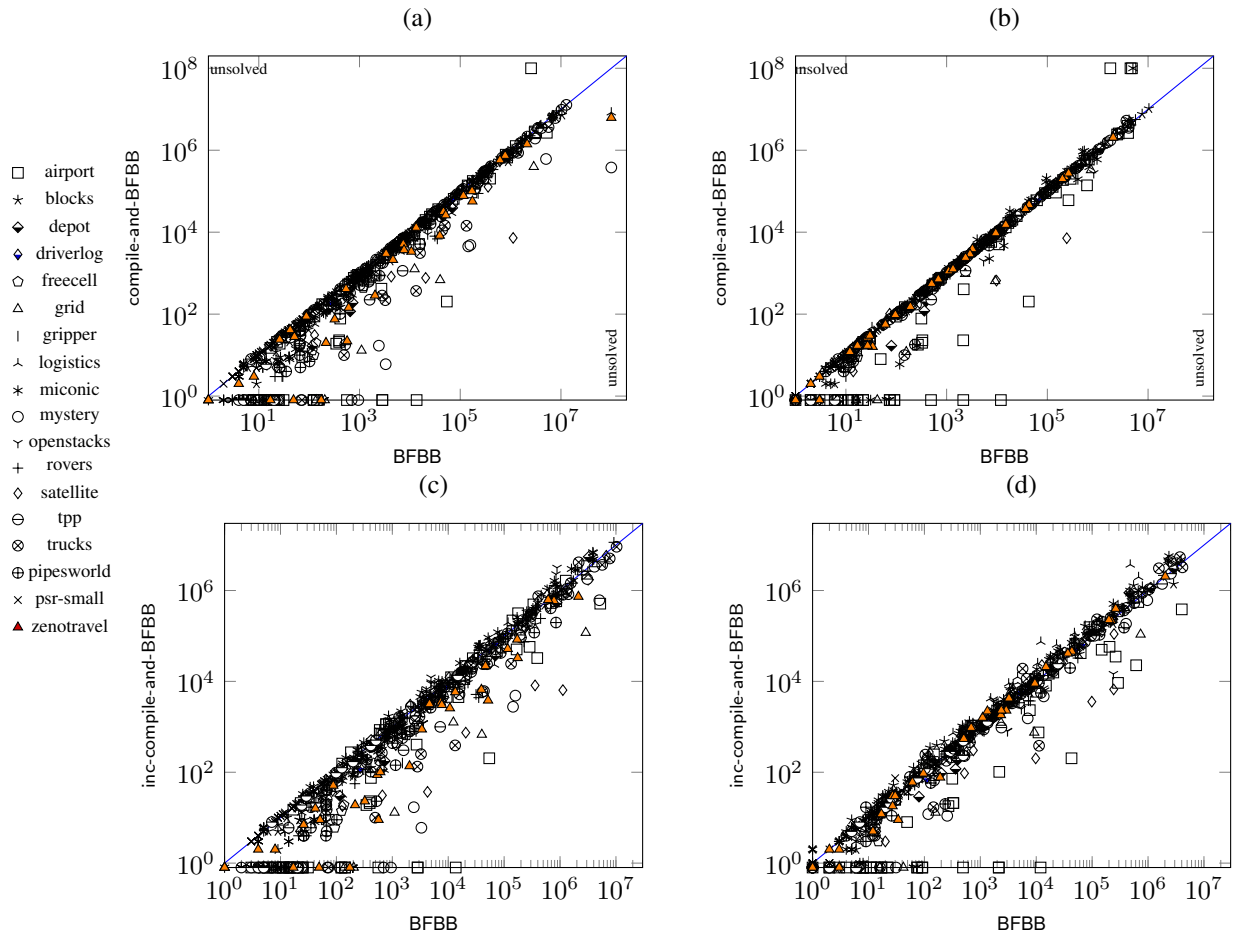


Figure 4: Comparative view of empirical results in terms of expanded nodes: (a) & (b) *BFBB* vs. compile-and-BFBB, (c) & (d) *BFBB* vs. inc-compile-and-BFBB, with blind ((a) & (c)) and additive projections ((b) & (d)) heuristics

panded by compile-and-BFBB was often substantially lower than the number of nodes expanded by *BFBB*. Here, *BFBB* and compile-and-BFBB solved with no search 39 and 85 task/budget pairs, respectively. Finally, despite the rather ad hoc setting of our incremental inc-compile-and-BFBB procedure, switching from compile-and-BFBB to inc-compile-and-BFBB was typically beneficial, though much deeper investigation and development of inc-compile-and-BFBB is obviously still required.

References

Baier, J. A.; Bacchus, F.; and McIlraith, S. A. 2007. A heuristic search approach to planning with temporally extended preferences. In *IJCAI*, 1808–1815.

Benton, J.; Do, M.; and Kambhampati, S. 2009. Anytime heuristic search for partial satisfaction planning. *AIJ* 173(5-6):562–592.

Benton, J.; van den Briel, M.; and Kambhampati, S. 2007. A hybrid linear programming and relaxed plan heuristic for partial satisfaction planning problems. In *ICAPS*, 34–41.

Bonet, B., and Geffner, H. 2008. Heuristics for planning with penalties and rewards formulated in logic and computed through circuits. *AIJ* 172(12–13):1579–1604.

Bonet, B., and Helmert, M. 2010. Strengthening landmark heuristics via hitting sets. In *ECAI*, 329–334.

Brafman, R., and Chernyavsky, Y. 2005. Planning with goal preferences and constraints. In *ICAPS*, 182–191.

Coles, A. J., and Coles, A. 2011. LPRPG-P: Relaxed plan heuristics for planning with preferences. In *ICAPS*, 26–33.

Do, M. B.; Benton, J.; van den Briel, M.; and Kambhampati, S. 2007. Planning with goal utility dependencies. In *IJCAI*, 1872–1878.

Domshlak, C.; Katz, M.; and Lefler, S. 2012. Landmark-enhanced abstraction heuristics. *AIJ* 189:48–68.

Edelkamp, S. 2001. Planning with pattern databases. In *ECP*, 13–24.

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

Gerevini, A.; Saetti, A.; and Serina, I. 2008. An approach to efficient planning with numerical fluents and multi-criteria plan quality. *AIJ* 172(8–9):899–944.

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

Haslum, P.; Botea, A.; Helmert, M.; Bonet, B.; and Koenig, S. 2007. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *AAAI*, 1007–1012.

Haslum, P. 2013. Heuristics for bounded-cost search. In *ICAPS*, 312–316.

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

Helmert, M.; Haslum, P.; and Hoffmann, J. 2007. Flexible abstraction heuristics for optimal sequential planning. In *ICAPS*, 200–207.

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

Hoffmann, J.; Gomes, C. P.; Selman, B.; and Kautz, H. A. 2007. SAT encodings of state-space reachability problems in numeric domains. In *IJCAI*, 1918–1923.

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

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

Katz, M., and Domshlak, C. 2010a. Implicit abstraction heuristics. *JAIR* 39:51–126.

Katz, M., and Domshlak, C. 2010b. Optimal admissible composition of abstraction heuristics. *AIJ* 174:767–798.

Keyder, E., and Geffner, H. 2009. Soft goals can be compiled away. *JAIR* 36:547–556.

Mirkis, V., and Domshlak, C. 2013. Abstractions for oversubscription planning. In *ICAPS*, 153–161.

Mirkis, V., and Domshlak, C. 2014. Landmarks in oversubscription planning. Technical Report IE/IS-2014-01, Technion.

Nakhost, H.; Hoffmann, J.; and Müller, M. 2012. Resource-constrained planning: A Monte Carlo random walk approach. In *ICAPS*, 181–189.

Pommerening, F., and Helmert, M. 2013. Incremental LM-Cut. In *ICAPS*, 162–170.

Porteous, J.; Sebastia, L.; and Hoffmann, J. 2001. On the extraction, ordering, and usage of landmarks in planning. In *ECP*, 37–49.

Richter, S.; Helmert, M.; and Westphal, M. 2008. Landmarks revisited. In *AAAI*, 975–982.

Sanchez, R., and Kambhampati, S. 2005. Planning graph heuristics for selecting objectives in over-subscription planning problems. In *ICAPS*, 192–201.

Smith, D. 2004. Choosing objectives in over-subscription planning. In *ICAPS*, 393–401.

Thayer, J. T., and Ruml, W. 2011. Bounded suboptimal search: A direct approach using inadmissible estimates. In *IJCAI*, 674–679.

Thayer, J. T.; Stern, R. T.; Felner, A.; and Ruml, W. 2012. Faster bounded-cost search using inadmissible estimates. In *ICAPS*, 270–278.

Adding Local Exploration to Greedy Best-First Search in Satisficing Planning

Fan Xie and **Martin Müller** and **Robert Holte**
Computing Science, University of Alberta
Edmonton, Canada
{fxie2,mmueller,rholte}@ualberta.ca

Abstract

Greedy Best-First Search (GBFS) is a powerful algorithm at the heart of many state of the art satisficing planners. One major weakness of GBFS is its behavior in so-called uninformative heuristic regions (UHRs) - parts of the search space in which no heuristic provides guidance towards states with improved heuristic values.

This work analyzes the problem of UHRs in planning in detail, and proposes a two level search framework as a solution. In *Greedy Best-First Search with Local Exploration (GBFS-LE)*, a local exploration is started within a global GBFS whenever the search seems stuck in UHRs.

Two different local exploration strategies are developed and evaluated experimentally: *Local GBFS (LS)* and *Local Random Walk Search (LRW)*. The two new planners LAMA-LS and LAMA-LRW integrate these strategies into the GBFS component of LAMA-2011. Both are shown to yield clear improvements in terms of both coverage and search time on standard International Planning Competition benchmarks, especially for domains that are proven to have unbounded or large UHRs.[1]

Introduction

In the latest International Planning Competition IPC-2011 (García-Olaya, Jiménez, and Linares López 2011), the planner LAMA-2011 (Richter and Westphal 2010) was the clear winner of the sequential satisficing track, by both measures of coverage and plan quality. LAMA-2011 finds a first solution using Greedy Best-First Search (GBFS) (Bonet and Geffner 2001; Helmert 2006) with popular enhancements such as Preferred Operators, Deferred Evaluation (Richter and Helmert 2009) and Multi-Heuristic search (Richter and Westphal 2010). Solutions are improved using restarting weighted A*.

GBFS always expands a node n that is closest to a goal state according to a heuristic h. While GBFS makes no guarantees about solution quality, it can often find a solution quickly. The performance of GBFS strongly depends on the quality of h. Misleading or uninformative heuristics can massively increase its running time.

Copyright © 2014, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

[1] Another version of this paper is published in AAAI 2014 (Xie, Müller, and Holte 2014)

The main focus of this paper is on one such problem with GBFS: uninformative heuristic regions (UHRs), which includes *local minima* and *plateaus*. A *local minimum* is a state with minimum h-value within a local region which is not a global minimum. A *plateau* is an area of the state space where all states have the same heuristic value. GBFS, because of its open list, can get stuck in multiple UHRs at the same time.

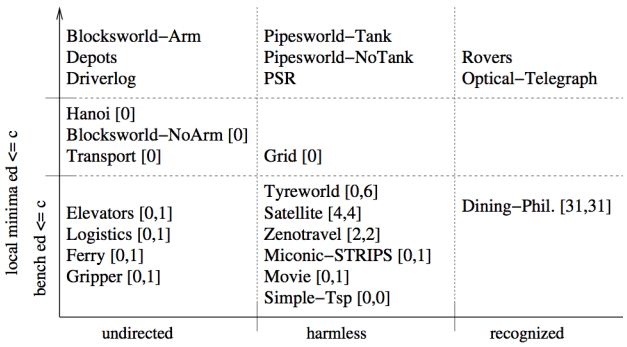


Figure 1: Overview of h^+ topology (Hoffmann 2011). Domains with unrecognized dead ends are not shown.

Hoffmann has studied the problem of UHRs for the case of the optimal relaxation heuristic h^+ (Hoffmann 2005; 2011). He classified a large number of planning benchmarks, shown in Figure 1, according to their *maximum exit distance* from plateaus and local minima, and by whether dead ends exist and are recognized by h^+. The current work proposes local exploration to improve GBFS. The focus of the analysis is on domains with a large or even unbounded maximum exit distance for plateaus and local minima, but without unrecognized dead ends. In these domains, there exists a plan from each state in an UHR (with $h^+ < \infty$).

As an example, the IPC domain 2004-notankage has no dead ends, but contains unbounded plateaus and local minima (Hoffmann 2011). Instance #21 shown in Figure 2 serves to illustrate a case of bad search behavior in GBFS due to UHRs. The figure plots the current minimum heuristic value h_{min} in the closed list on the x-axis against the log-scale cumulative search time needed to first reach h_{min}. The solid line is for GBFS with h^{FF}. The two huge increases

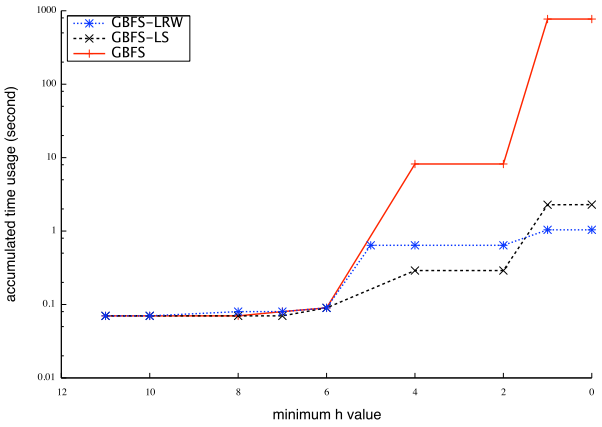


Figure 2: Cumulative search time (in seconds) of GBFS, GBFS-LS and GBFS-LRW with h^{FF} for first reaching a given h_{min} in 2004-notankage #21.

in search time, with the largest (763 seconds) for the step from $h_{min} = 2$ to $h_{min} = 1$, correspond to times when the search is stalled in multiple UHRs. Since the large majority of overall search time is used to inefficiently find an escape from UHRs, it seems natural to try switching to a secondary search strategy which is better at escaping. Such ideas have been tried several times before. This related work is reviewed and compared in the next section.

The current paper introduces a framework which adds a local search algorithm to GBFS in order to improve its behavior in UHRs. Two such algorithms, *local GBFS (LS(n))* and *local random walks (LRW(n))*, are designed to find quicker escapes from UHRs, starting from a node n within the UHRs. The main contributions of this work are:

- An analysis of the problem of UHRs in GBFS, and its consequences for limiting the performance of GBFS in current benchmark problems in satisficing planning.
- A new search framework, Greedy Best-First Search with Local Exploration (GBFS-LE), which runs a separate local search whenever the main global GBFS seems to be stuck. Two concrete local search algorithms, local GBFS (*LS*) and local random walks (*LRW*), are shown to be less sensitive to UHRs and when incorporated into GBFS are shown to outperform the baseline by a substantial margin over the IPC benchmarks.
- An analysis of the connection between Hoffmann's theoretical results on local search topology (Hoffmann 2005; 2011) and the performance of adding local exploration into GBFS.

The remainder of the paper is organized as follows: after a brief review of previous work on strategies for escaping from UHR, the new search framework GBFS-LE is introduced, compared with related work, and evaluated experimentally on IPC domains. A discussion of possible future work includes perspectives for addressing the early mistakes problem within GBFS-LE.

Search Strategies for Escaping UHRs

There are several approaches to attack the UHR problem. Better quality heuristics (Hoffmann and Nebel 2001; Helmert 2004; Helmert and Geffner 2008) can shrink the size of UHRs, as can combining several heuristics (Richter and Westphal 2010; Röger and Helmert 2010). Additional knowledge from heuristic computation or from problem structure can be utilized in order to escape from UHRs. Examples are helpful actions (Hoffmann and Nebel 2001) and explorative probes (Lipovetzky and Geffner 2011). The third popular approach is to develop search algorithms that are less sensitive to flaws in heuristics. Algorithms which add a global exploration component to the search, which is especially important for escaping from unrecognized *dead ends*, include restarting (Nakhost and Müller 2009; Coles, Fox, and Smith 2007) and non-greedy node expansion (Valenzano et al. 2014; Imai and Kishimoto 2011; Xie et al. 2014). The current paper focuses on another direction: adding a local exploration component to the globally greedy GBFS algorithm.

The planner Marvin adds machine-learned plateau-escaping macro-actions to enforced hill-climbing (Coles and Smith 2007). YAHSP constructs macro actions from FF's relaxed planning graph (Vidal 2004). Identidem adds exploration by expanding a sequence of actions chosen probabilistically, and proposes a framework for escaping from local minima in local-search forward-chaining planning (Coles, Fox, and Smith 2007). Arvand (Nakhost and Müller 2009) uses random walks to explore quickly and deeply. Arvand-LS (Xie, Nakhost, and Müller 2012) combines random walks with local greedy best-first search, while Roamer (Lu et al. 2011) adds exploration to LAMA-2008 by using fixed-length random walks. Nakhost and Müller's analysis (2012) shows that while random walks outperform GBFS in many plateau escape problems, they fail badly in domains such as Sokoban, where a precise action sequence must be found to escape. However, while escaping from UHRs has been well studied in the context of these local search based planners, there is comparatively little research on how to use search for escaping UHRs in the context of GBFS. This paper begins to fill this gap.

GBFS-LE: GBFS with Local Exploration

The new technique of *Greedy Best-First Search with Local Exploration (GBFS-LE)* uses local exploration whenever a global GBFS (G-GBFS) seems stuck. If G-GBFS fails to improve its minimum heuristic value h_{min} for a fixed number of node expansions, then GBFS-LE runs a small local search for exploration, *LocalExplore(n)*, from the best node n in a global-level open list. Algorithm 1 shows GBFS-LE. *STALL_SIZE* and *MAX_LOCAL_TRY*, used at Line 24, are parameters which control the tradeoff between global search and local exploration.

The main change from GBFS is the call to *LocalExplore(n)* at Line 26 whenever there has been no improvement in heuristic value over the last *STALL_SIZE* node expansions.

Two local exploration strategies were tested. The first is

Algorithm 1 GBFS-LE

Input Initial state I, goal states G
Parameter STALL_SIZE, MAX_LOCAL_TRY
Output A solution plan

```
1: if h(I) < ∞ then
2:    (open, h_min) ← ([I], h(I))
3: end if
4: stalled ← 0; nuLocalTry ← 0
5: while open ≠ ∅ do
6:    n ← open.remove_min(){FIFO tie-breaking}
7:    if n ∈ G then
8:       return plan from I to n
9:    end if
10:   closed.insert(n)
11:   for each v ∈ successors(n) do
12:      if v ∉ closed then
13:         if h(v) < ∞ then
14:            open.insert(v, h(v))
15:            if h_min > h(v) then
16:               h_min ← h(v)
17:               stalled ← 0; nuLocalTry ← 0
18:            else
19:               stalled ← stalled + 1
20:            end if
21:         end if
22:      end if
23:   end for
24:   if stalled = STALL_SIZE
      and nuLocalTry < MAX_LOCAL_TRY then
25:      n ← open.peek_min()
26:      LocalExplore(n){local GBFS or random walks}
27:      stalled ← 0; nuLocalTry ← nuLocalTry + 1
28:   end if
29: end while
```

Algorithm 2 $LS(n)$, *local GBFS*

Input state n, goal states G, h_{min} {global variable}, $open$, $closed$
Parameter LSSIZE

```
1: local_open ← [n]
2: h_improved ← false
3: for i = 1 to LSSIZE do
4:    if local_open = ∅ then
5:       return
6:    end if
7:    n ← local_open.remove_min() {FIFO
      tie-breaking}
8:    if n ∈ G then
9:       return plan from I to n
10:   end if
11:   closed.insert(n)
12:   for each v ∈ successors(n) do
13:      if v ∉ closed then
14:         if h(v) < ∞ then
15:            local_open.insert(v, h(v))
16:            if h_min > h(v) then
17:               h_min ← h(v)
18:               h_improved ← true
19:            end if
20:         end if
21:      end if
22:   end for
23:   if h_improved then
24:      break
25:   end if
26: end for
27: merge(open,local_open)
28: return
```

local GBFS search starting from node n, ***LocalExplore(n)*** = ***LS(n)***, which shares the closed list of G-GBFS, but maintains its own separate open list *local_open* that is cleared before each local search. $LS(n)$, as shown in Algorithm 2, succeeds if it finds a node v with $h(v) < h_{min}$ at Line 16 before it exceeds the ***LSSIZE*** limit. In any case, the remaining nodes in *local_open* are merged into the global open list. A local search tree grown from a single node n is much more focused and grows deep much more quickly than the global open list in G-GBFS. It also restricts the search to a single plateau, while G-GBFS can get stuck when exploring many separate plateaus simultaneously. Both G-GBFS and ***LS(n)*** use a first-in-first-out tie-breaking rule, since last-in-first-out did not work well: it often led to long aimless walks within a UHR.

The second local exploration strategy tested is local random walk search, ***LocalExplore(n) = LRW(n)***, as shown in Algorithm 3. The implementation of random walks from the Arvand planner (Nakhost and Müller 2009; Nakhost et al. 2011) is used. $LRW(n)$ runs up to a pre-set number of random walks starting from node n, and evaluates only the endpoint of each walk using h^{FF}. All intermediate states are checked for whether they are goal states. Like $LS(n)$, $LRW(n)$ succeeds if it finds a node v with $h(v) < h_{min}$ within its exploration limit at Line 15. In this case, v is added to the global open list, and the path from n to v is stored for future plan extraction. In case of failure, unlike ***LS(n)***, no information is kept.

Parameters, as in Arvand-2011, are expressed as a tuple $(len\_walk, e\_rate, e\_period, WalkType)$ (Nakhost and Müller 2009). Random walk length scaling is controlled by an initial walk length of len_walk, an extension rate of e_rate and an extension period of $NUMWALKS * e\_period$. This is very different from Roamer, which uses fixed length random walks. $WalkType$ defines two different strategies for action selecting at Line 8: Monte Carlo Helpful Actions (***MHA***), which bias random walks by helpful actions, and pure random (***PURE***). For example, in configuration $(1, 2, 0.1, MHA)$ all random walks use the MHA walk type, and if h_{min} does not improve for $NUMWALKS * 0.1$ random walks, then the length of walks, len_walk, which starts at 1, will be doubled. LRW was tested with the following two configurations: $(1, 2, 0.1, MHA)$, which is used with preferred operators, and $(1, 2, 0.1, PURE)$.

The example of Figure 2 is solved much faster, in around

Algorithm 3 $LRW(n)$, *local random walk*

Input state n, goal states G, h_{min} {global variable}, *open*
Parameter LSSIZE

```
for i = 1 to LSSIZE do
    s ← n
    for j = 1 to LENGTH_WALK do
        A ← ApplicableActions(s)
        if A = ∅ then
            break
        end if
        a ← SelectAnActionFrom(A)
        s ← apply(s, a)
        if s ∈ G then
            open.insert(s, h(s))
            return
        end if
    end for
    if h(s) < h_min then
        open.insert(s, h(s))
        break
    end if
end for
return
```

1 second, by both GBFS-LS and GBFS-LRW, while GBFS needs 771 seconds. The three algorithms built exactly the same search trees until they first achieved the minimum h-value 6. The local GBFS in GBFS-LS, because it could focus on one branch, found a 5 step path that decreases the minimum h-value using only 10 expansions. The h-values along the path were 6, 7, 7, 6 and 4, showing an initial increase before decreasing. h-values along GBFS-LRW's path also increased before decreasing. In contrast, GBFS gets stuck in multiple separate h-plateaus since it needs to expand over 10000 nodes with h-value 6, which were distributed in many different parts of the search tree. Only after exhausting these, it expands the first node with $h = 7$. In this example, the local explorations, which expand or visit higher h-value nodes earlier, massively speed up the escape from UHRs.

There are several major differences between GBFS-LS and GBFS-LRW. GBFS-LS keeps all the information gathered during local searches by copying its nodes into the global open list at the end. GBFS-LRW keeps only endpoints that improve h_{min} and the paths leading to them. This causes a difference in how often the local search should be called. For GBFS-LS, it is generally safe to do more local search, while over-use of local search in GBFS-LRW can waste search effort[2]. This suggests using more conservative settings for the parameters *MAX_LOCAL_TRY* and *LSSIZE* in *LRW(n)*. The two algorithms also explore UHRs very differently. *LS(n)* systematically searches the subtree of n, while *LRW(n)* samples paths leading from n sparsely but deeply.

[2]Each step in a random walk generates all children and randomly picks one, which is only slightly cheaper than one expansion by LS when Deferred Evaluation is applied.

Experimental Results

Experiments were run on a set of 2112 problems in 54 domains from the seven International Planning Competitions which are publicly available[3], using one core of a 2.8 GHz machine with 4 GB memory and 30 minutes per instance. Results for planners which use randomization are averaged over five runs. All planners are implemented on the Fast Downward code base FD-2011 (Helmert 2006). The translation from PDDL to SAS+ was done only once, and this common preprocessing time is not counted in the 30 minutes. Parameters were set as follows: *STALL_SIZE = 1000* for both algorithms. (*MAX_LOCAL_TRY*, *LSSIZE*) = (100, 1000) for GBFS-LS and (10, 100) for GBFS-LRW.

Local Search Topology for h^+

For the purpose of experiments on UHRs, the detailed classification by h^+ of Figure 1 can be coarsened into three broad categories:

- *Unrecognized-Deadend*: 195 problems from 4 domains with unrecognized dead ends: Mystery, Mprime, Freecell and Airport.
- *Large-UHR*: 383 problems from domains with UHRs which are large or of unbounded exit distance, but with recognized dead ends: column 3 in Figure 1, plus the top two rows of columns 1 and 2.
- *Small-UHR*: 669 problems from domains without UHRs, or with only small UHRs, corresponding to columns 1 and 2 in the bottom row of Figure 1.

Note, problems from these three categories are only a subset of the total 2112 problems. Only a part of the 54 domains were analyzed by Hoffmann (2011).

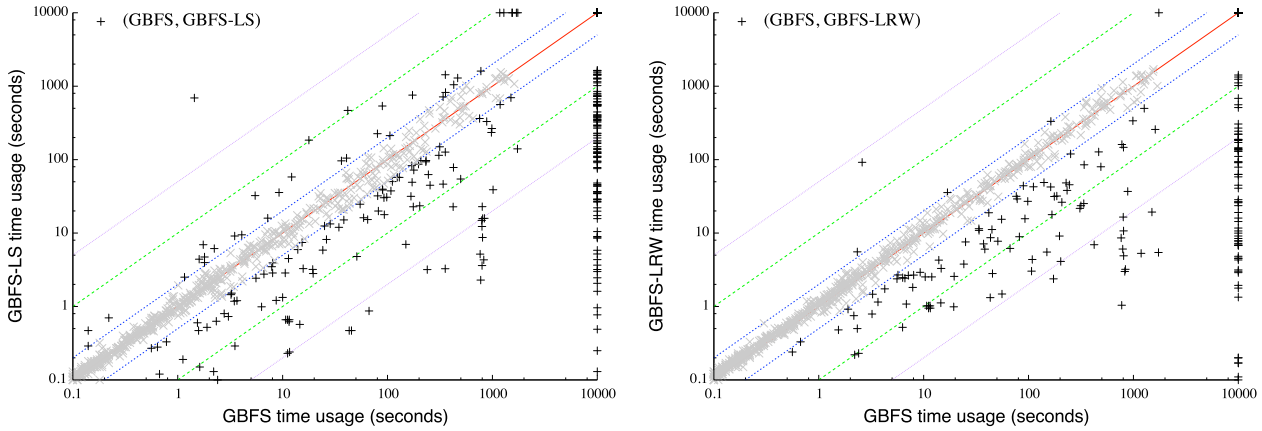
Performance of Baseline Algorithms

The baseline study evaluates GBFS, GBFS-LS and GBFS-LRW without the common planning enhancements of preferred operators, deferred evaluation and multi-heuristics. Three widely used planning heuristics are tested: FF (Hoffmann and Nebel 2001), causal graph (CG) (Helmert 2004) and context-enhanced additive (CEA) (Helmert and Geffner 2008). We use the distance-base versions for the three heuristics. They estimate the length of a solution path starting from the evaluated state. Table 1 shows the coverage on all 2112 IPC instances. Both GBFS-LS and GBFS-LRW outperform GBFS by a substantial margin for all 3 heuristics.

Heuristic	GBFS	GBFS-LS	GBFS-LRW
FF	1561	**1657**	1619.4
CG	1513	**1602**	1573.2
CEA	1498	1603	**1615.2**

Table 1: IPC coverage out of 2112 for GBFS with and without local exploration, and three standard heuristics.

[3]Our IPC test set does not include Hanoi, Ferry and Simple-Tsp from Figure 1.



(a) GBFS(X) vs GBFS-LS(Y)

(b) GBFS(X) vs GBFS-LRW(Y)

Figure 3: Comparison of time usage of the three baseline algorithms. 10000 corresponds to runs that timed out or ran out of memory. Results shown for one typical run of GBFS-LRW.

Benchmarks	GBFS	GBFS-LS	GBFS-LRW
UR-Deadend(195)	162	162(0.0%)	**169(3.7%)**
Large-UHR(383)	195	214(9.7 %)	**225(15.3%)**
Small-UHR(669)	634	637 (0.5%)	**641(1.1%)**

Table 2: Coverage comparison on the three domain categories for GBFS and GBFS-LE with h^{FF}. UR-Deadend is short for Unrecognized-Deadend. The same typical run in Figure 3 is used for GBFS-LRW. Numbers in parentheses show coverage improvements compared to GBFS.

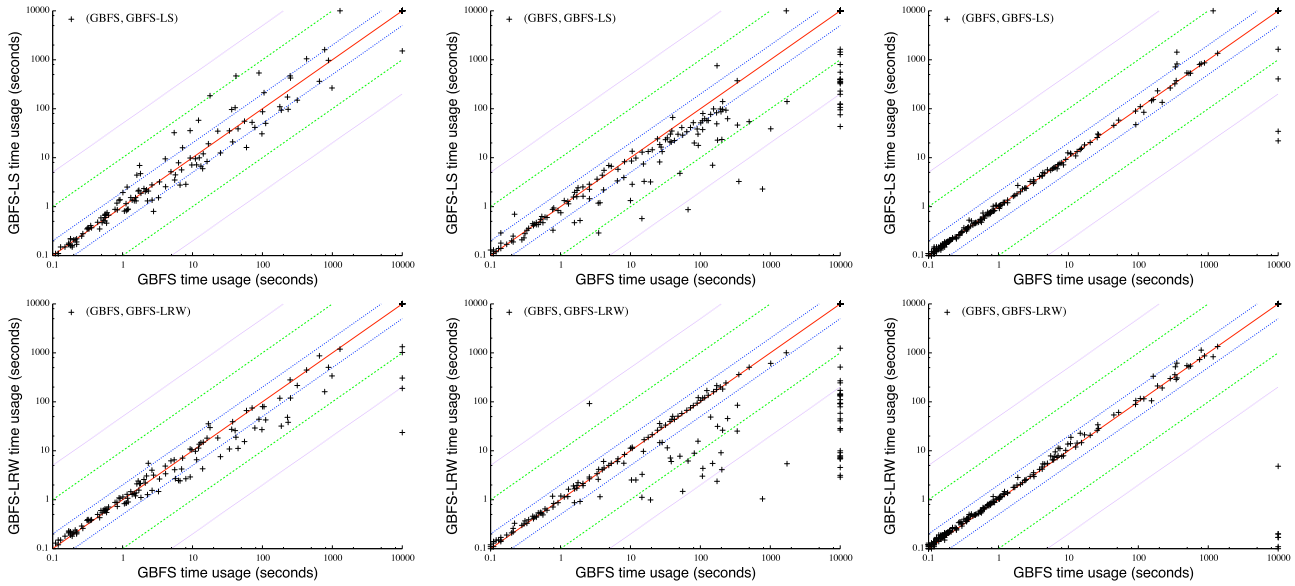
Figure 3 compares the time usage of the two proposed algorithms with GBFS using h^{FF} over all IPC benchmarks. Every point in the figure represents one instance, plotting the search time for GBFS on the x-axis against GBFS-LS (left) and GBFS-LRW (right) on the y-axis. Only problems for which both algorithms need at least 0.1 seconds are shown. Points below the main diagonal represent instances that the new algorithms solve faster than GBFS. For ease of comparison, additional reference lines indicate $2\times$, $10\times$ and $50\times$ relative speed. Data points within a factor of 2 are shown in grey in order to highlight the instances with substantial differences. Problems that were only solved by one algorithm within the 1800 second time limit are included at $x = 10000$ or $y = 10000$. Both new algorithms show substantial improvements in search time over GBFS.

Figure 4 restricts the comparison to Unrecognized-Deadend, Large-UHR and Small-UHR respectively. Table 2 shows the overall coverages. In Large-UHR, GBFS-LS and GBFS-LRW solve 19 (+9.7%) and 30 (+15.3%) more problems than GBFS (195/383) respectively. Both outperform GBFS in search time. However, in Small-UHR, GBFS-LS and GBFS-LRW only solve 3 (+0.5%) and 7 (+1.1%) more problems than GBFS (634/669), and there is very little difference in search time among the three algorithms. This result clearly illustrates the relationship between the size of UHRs and the performance of the two local exploration techniques. For Unrecognized-Deadend, GBFS-LS is slightly slower than GBFS with the same coverage (162/195), while GBFS-LRW is slightly faster and solves 7 (+3.7%) more problems. The effect of local exploration on the performance in the case of unrecognized dead-ends is not clear-cut.

Performance with Search Enhancements

Experiments in this section test the two proposed algorithms when three common planning enhancements are added: Deferred Evaluation, Preferred Operators and Multiple Heuristics. h^{FF} is used as the primary heuristic in all cases.

- *Deferred Evaluation* delays state evaluation and uses the parent's heuristic value in the priority queue (Richter and Helmert 2009). This technique is used in G-GBFS and *LS(n)*, but not in the endpoint-only evaluation of random walks in *LRW(n)*.
- The *Preferred Operators* enhancement keeps states reached via a preferred operator, such as helpful actions in h^{FF}, in an additional open list (Richter and Helmert 2009). An extra preferred open list is also added to $LS(n)$. Boosting with default parameter 1000 is used, and Preferred Operator first ordering is used for tie-breaking as in LAMA-2011 (Richter and Westphal 2010). In $LRW(n)$, preferred operators are used in form of the *Monte Carlo with Helpful Actions* (MHA) technique (Nakhost and Müller 2009), which biases random walks towards using operators which are often preferred.
- The *Multi-Heuristics* approach maintains additional open lists in which states are evaluated by other heuristic functions. Because of its proven strong performance in LAMA, the *Landmark count* heuristic h^{lm} (Richter, Helmert, and Westphal 2008) is used as the second heuristic. Both G-GBFS and *LS(n)* use a round robin strategy for picking the next node to expand. In Fast Downward, h^{lm} is calculated incrementally from the parent node. When Multi-Heuristics is applied to GBFS-LRW, the $LRW(n)$ part still uses h^{FF} because the path-dependent landmark



(a) Unrecognized-Deadend (b) Large-UHR (c) Small-UHR

Figure 4: Comparison of time usage of the three baseline algorithms over the three different categories. 10000 corresponds to runs that timed out or ran out of memory. Results shown for one typical run of GBFS-LRW, which is selected by comparing all 5 runs and picking the most typical one. They are all very similar.

computation was not implemented for random walks. When $LRW(n)$ finds an heuristically improved state s, GBFS-LRW evaluates and expands all states along the path to s in order to allow the path-dependent computation of $h^{lm}(s)$ in G-GBFS. Without Multi-Heuristics, only s itself is inserted to the open list.

Table 3 shows the experimental results on all IPC domains. Used as a single enhancement, Preferred Operators improves all three algorithms. Deferred Evaluation improves GBFS-LS and GBFS-LRW, but fails for GBFS, mainly due to plateaus caused by the less informative node evaluation (Richter and Helmert 2009). In GBFS-LS and GBFS-LRW, the benefit of faster search outweighs the weaker evaluation. Multi-Heuristics strongly improves GBFS and GBFS-LS, but is only a modest success in GBFS-LRW. This is not surprising since *LRW(n)* does not use h^{lm}, and in order to evaluate the new best states generated by *LRW(n)* with h^{lm} in G-GBFS, all nodes on the random walk path need to be evaluated, which degrades performance. When combining two enhancements, all three algorithms achieve their best performance with Preferred Operators plus Deferred Evaluation. Figure 5 compares the time usage of the three algorithms in this case.

Comparing State of the Art Planners in terms of Coverage and Search Time

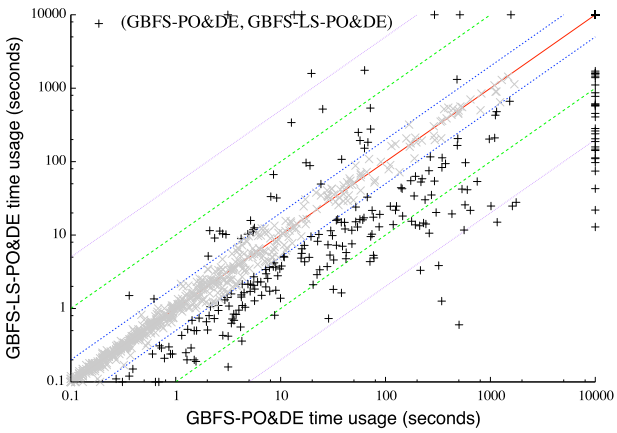
The final row in Table 3 shows coverage results when all three enhancements are applied. The performance comparisons in this section use this best known configuration in terms of coverage for three algorithms based on GBFS, GBFS-LS and GBFS-LRW, which closely correspond to the "coverage-only" first phase of the LAMA-2011 planner:

Enhancement	GBFS	GBFS-LS	GBFS-LRW
(none)	1561	1657	1619.4
PO	1826	1851	1827.4
DE	1535	1721	1635
MH	1851	1874	1688.4
PO + DE	1871	1889	1880.6
PO + MH	1850	1874	1854.2
DE + MH	1660	1764	1730.2
PO + DE + MH	**1913**	**1931**	**1925.4**

Table 3: Number of instances solved with search enhancements, out of 2112. PO = Preferred Operators, DE = Deferred Evaluation, MH = Multi-Heuristic.

- **LAMA-2011:** only the first GBFS iteration of LAMA is run, with deferred evaluation, preferred operators and multi-heuristics with h^{FF} and h^{lm} (Richter and Westphal 2010).
- **LAMA-LS:** Configured like LAMA-2011, but with GBFS replaced by GBFS-LS.
- **LAMA-LRW:** GBFS in LAMA-2011 is replaced by GBFS-LRW.

Table 4 shows the coverage results per domain. LAMA-LS has the best overall coverage, 18 more than LAMA-2011, closely followed by LAMA-LRW. LAMA-LS solves more problems in 7 of the 10 domains where LAMA and LAMA-LS differ in coverage. This number for LAMA-LRW is 7 out of 11. Although LAMA-LRW uses a randomized algorithm, our 5 runs for LAMA-LRW had quite stable results: 1927, 1924, 1926, 1924 and 1926. By comparison, adding the landmark count heuristic, which differentiates LAMA-



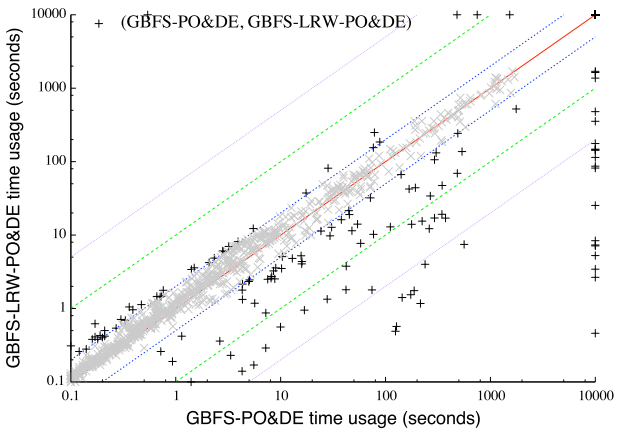
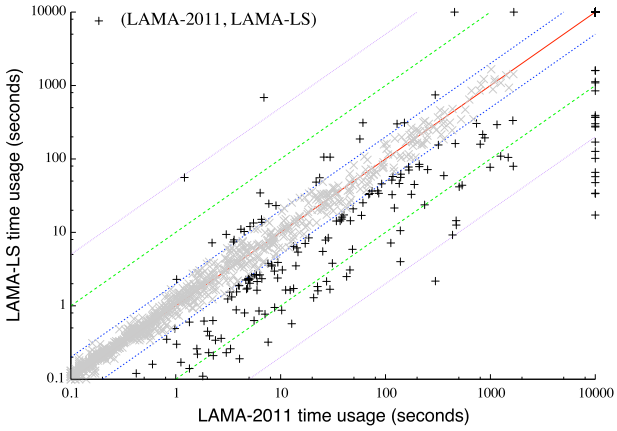
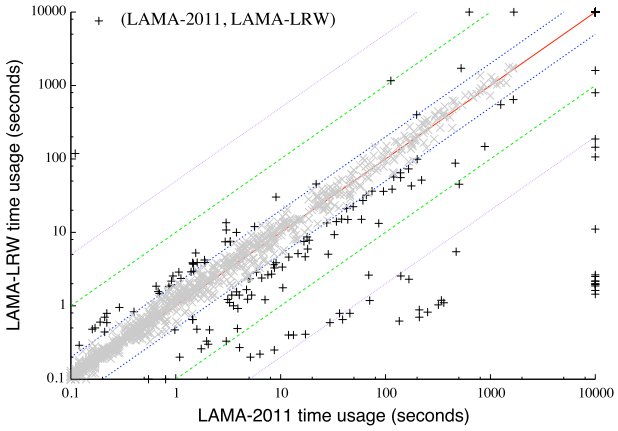


Figure 5: Comparison of time usage of the three baseline algorithms with Preferred Operators and Deferred Evaluation. 10000 corresponds to runs that timed out or ran out of memory. A typical single run of GBFS-LRW-PO&DE is shown.



(a) LAMA-2011(X) VS LAMA-LS(Y)



(b) LAMA-2011(X) VS LAMA-LRW(Y)

Figure 6: Comparison of time usage of LAMA-2011 with LAMA-LS and LAMA-LRW. A typical single run is used for LAMA-LRW.

2011 from other planners based on the Fast Downward code base, improves the coverage of LAMA-2011 by 42, from 1871 to 1913.

Using the same format as Figure 3 for baseline GBFS, Figure 6 compares the search time of the three planners over the IPC benchmark. Both LAMA-LS and LAMA-LRW show a clear overall improvement over LAMA-2011 in terms of speed. In Figure 7, the benefit of local exploration for search time in Large-UHR still holds even with all enhancements on. Both LAMA-LS and LAMA-LRW solve 12 more problems (4.1%) than LAMA-2011's 290/383 in Large-UHR, while in Small-UHR they solve 1 and 2 fewer problems respectively than LAMA-2011's 646/669. Table 5 compares coverages of the three planners over different categories.

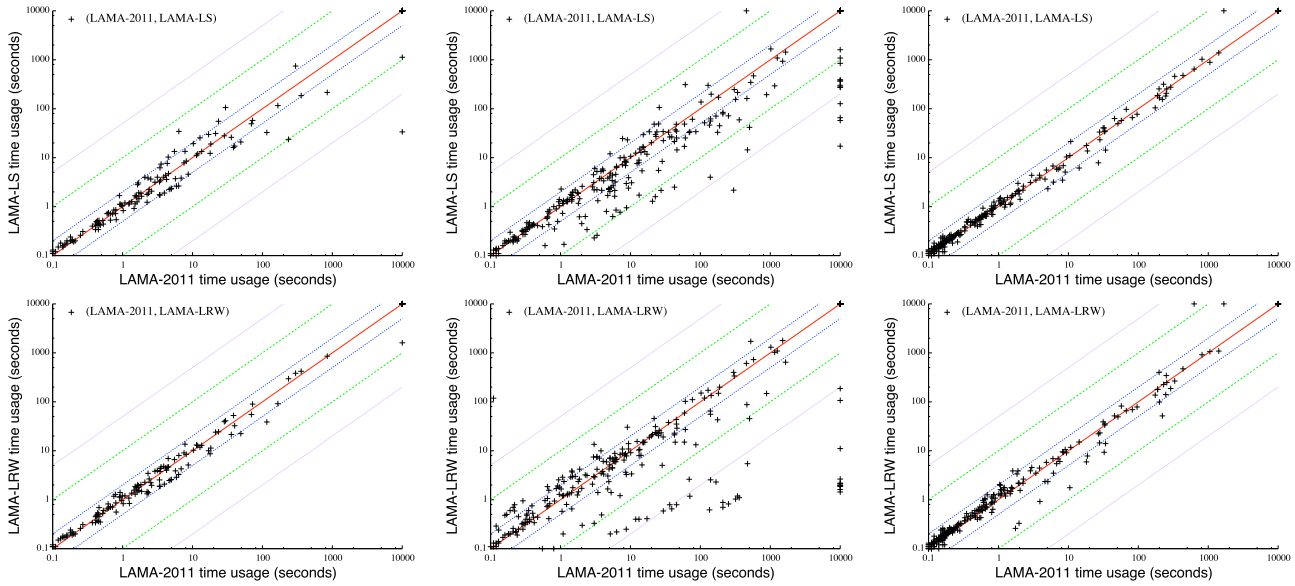
For further comparison, the coverage results of some other strong planners from IPC-2011 on the same hardware are: FDSS-2 solves 1912/2112, Arvand 1878.4/2112, Lama-2008 1809/2112, fd-auto-tune-2 1747/2112, and Probe 1706/1968 (failed on the ":derive" keyword in 144 problems).

Although the local explorations are inclined to increase the solution length, the influence is not clear-cut since they also solve more problems. The IPC-2011 style plan quality scores for LAMA-2011, LAMA-LS and LAMA-LRW are 1898.0, 1899.6 and 1900.5.

Conclusions and Future Work

While local exploration has been investigated before in the context of local search planners, it also serves to facilitate escaping from UHRs for greedy best-first search. The new framework of GBFS-LE, GBFS with Local Exploration, has been tested successfully in two different realizations, adding local greedy best-first search in GBFS-LS and random walks in GBFS-LRW.

Future work should explore more types of local search such as FF's enforced hill-climbing (Hoffmann and Nebel 2001), and try to combine different local exploration meth-



(a) Unrecognized-Deadend (b) Large-UHR (c) Small-UHR

Figure 7: Comparison of time usage of LAMA-2011 with LAMA-LS and LAMA-LRW over the three different categories. A typical single run is used for LAMA-LRW.

Domain	Size	LAMA-2011	LAMA-LS	LAMA-LRW
00-miconic-ful	150	**136**	**136**	135.6
02-depot	22	**20**	**20**	19.6
02-freecell	80	78	**79**	78 .2
04-airport-str	50	32	**34**	32.8
04-notankage	50	**44**	43	**44**
04-optical-tel	48	4	**6**	4
04-philosoph	48	39	47	**47.8**
04-satellite	36	**36**	35	35
06-storage	30	18	**23**	21
06-tankage	50	41	41	**42**
08-transport	30	29	**30**	**29.6**
11-floortile	20	**6**	5	**6**
11-parking	20	18	**20**	16.8
11-transport	20	16	16	**17**
All others	1396	1396	1396	1396
Total	2112	1913	1931	1925.4
Unsolved		199	181	186.6

Table 4: Domains with different coverage for the three planners. 33 domains with 100% coverage and 7 further domains with identical coverage for all planners are not shown.

Benchmarks	LAMA-2011	LAMA-LS	LAMA-LRW
UR-Deadend(195)	164	**166(1.2%)**	165(0.6%)
Large-UHR(383)	290	**302(4.1 %)**	**302(4.1%)**
Small-UHR(669)	**646**	645 (-0.2%)	641(-0.3%)

Table 5: Coverages over the three domain categories for LAMA-2011, LAMA-LS and LAMA-LRW. UR-Deadend is short for Unrecognized-Deadend. The same typical run in Figure 6 is used for LAMA-LRW. Numbers in parentheses show coverage improvements compared to LAMA-2011.

ods in a principled way. One open problem of GBFS-LE is that it does not have a mechanism for dealing with *unrecognized dead-ends*. Local exploration in GBFS-LE always starts from the heuristically most promising state in the global open list, which might be mostly filled with nodes from such dead-ends. In domains such as 2011-nomystery (Nakhost, Hoffmann, and Müller 2012), almost all exploration will occur within such dead ends and therefore be useless. It would be interesting to combine GBFS-LE with an algorithm for increased *global-level exploration*, such as DBFS (Imai and Kishimoto 2011) and Type-GBFS (Xie et al. 2014).

Acknowledgements

The authors wish to thank the anonymous referees for their valuable advice. This research was supported by GRAND NCE, a Canadian federally funded Network of Centres of Excellence, NSERC, the Natural Sciences and Engineering Research Council of Canada, and AITF, Alberta Innovates Technology Futures.

References

Bonet, B., and Geffner, H. 2001. Heuristic search planner 2.0. *AI Magazine* 22(3):77–80.

Coles, A., and Smith, A. 2007. Marvin: A heuristic search planner with online macro-action learning. *Journal of Artificial Intelligence Research* 28:119–156.

Coles, A.; Fox, M.; and Smith, A. 2007. A new local-search algorithm for forward-chaining planning. In Boddy, M. S.; Fox, M.; and Thiébaux, S., eds., *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS-2007)*, 89–96.

García-Olaya, A.; Jiménez, S.; and Linares López, C., eds. 2011. *The 2011 International Planning Competition*.

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

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

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

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

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

Hoffmann, J. 2011. Where ignoring delete lists works, part II: Causal graphs. In Bacchus, F.; Domshlak, C.; Edelkamp, S.; and Helmert, M., eds., *Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS-2011)*, 98–105.

Imai, T., and Kishimoto, A. 2011. A novel technique for avoiding plateaus of greedy best-first search in satisficing planning. In Burgard, W., and Roth, D., eds., *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence (AAAI-2011)*, 985–991.

Lipovetzky, N., and Geffner, H. 2011. Searching for plans with carefully designed probes. In Bacchus, F.; Domshlak, C.; Edelkamp, S.; and Helmert, M., eds., *Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS-2011)*, 154–161.

Lu, Q.; Xu, Y.; Huang, R.; and Chen, Y. 2011. The Roamer planner random-walk assisted best-first search. In García-Olaya, A.; Jiménez, S.; and Linares López, C., eds., *The 2011 International Planning Competition*, 73–76.

Nakhost, H., and Müller, M. 2009. Monte-Carlo exploration for deterministic planning. In Walsh, T., ed., *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI'09)*, 1766–1771.

Nakhost, H., and Müller, M. 2012. A theoretical framework for studying random walk planning. In Borrajo, D.; Felner, A.; Korf, R. E.; Likhachev, M.; López, C. L.; Ruml, W.; and Sturtevant, N. R., eds., *Proceedings of the Fifth Annual Symposium on Combinatorial Search (SOCS-2012)*, 57–64.

Nakhost, H.; Müller, M.; Valenzano, R.; and Xie, F. 2011. Arvand: the art of random walks. In García-Olaya, A.; Jiménez, S.; and Linares López, C., eds., *The 2011 International Planning Competition*, 15–16.

Nakhost, H.; Hoffmann, J.; and Müller, M. 2012. Resource-constrained planning: A Monte Carlo random walk approach. In McCluskey, L.; Williams, B.; Silva, J. R.; and Bonet, B., eds., *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS-2012)*, 181–189.

Richter, S., and Helmert, M. 2009. Preferred operators and deferred evaluation in satisficing planning. In Gerevini, A.; Howe, A. E.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS-2009)*, 273–280.

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

Richter, S.; Helmert, M.; and Westphal, M. 2008. Landmarks revisited. In Fox, D., and Gomes, C. P., eds., *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI-2008)*, 975–982.

Röger, G., and Helmert, M. 2010. The more, the merrier: Combining heuristic estimators for satisficing planning. In Brafman, R. I.; Geffner, H.; Hoffmann, J.; and Kautz, H. A., eds., *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS-2010)*, 246–249.

Valenzano, R.; Schaeffer, J.; Sturtevant, N.; and Xie, F. 2014. A comparison of knowledge-based GBFS enhancements and knowledge-free exploration. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS-2014)*.

Vidal, V. 2004. A lookahead strategy for heuristic search planning. In Zilberstein, S.; Koehler, J.; and Koenig, S., eds., *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS-2004)*, 150–160.

Xie, F.; Müller, M.; Holte, R.; and Imai, T. 2014. Type-based exploration with multiple search queues for satisficing planning. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI-2014)*.

Xie, F.; Müller, M.; and Holte, R. 2014. Adding local exploration to greedy best-first search in satisficing planning. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI-2014)*.

Xie, F.; Nakhost, H.; and Müller, M. 2012. Planning via random walk-driven local search. In McCluskey, L.; Williams, B.; Silva, J. R.; and Bonet, B., eds., *Proceeedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS-2012)*, 315–322.

Type-based Exploration with Multiple Search Queues for Satisficing Planning

Fan Xie and **Martin Müller** and **Robert Holte**
Computing Science, University of Alberta
Edmonton, Canada
{fxie2, mmueller, robert.holte}@ualberta.ca

Tatsuya Imai
Tokyo Institute of Technology
Tokyo, Japan
imai7@is.titech.ac.jp

Abstract

Utilizing multiple queues in Greedy Best-First Search (GBFS) has been proven to be a very effective approach to satisficing planning. Successful techniques include extra queues based on *Helpful Actions* (or *Preferred Operators*), as well as using *Multiple Heuristics*. One weakness of all standard GBFS algorithms is their lack of exploration. All queues used in these methods work as priority queues sorted by heuristic values. Therefore, misleading heuristics, especially early in the search process, can cause the search to become ineffective.

Type systems, as introduced for heuristic search by Lelis et al, are a development of ideas for exploration related to the classic stratified sampling approach. The current work introduces a search algorithm that utilizes type systems in a new way – for exploration within a GBFS multiqueue framework in satisficing planning.

A careful case study shows the benefits of such exploration for overcoming deficiencies of the heuristic. The proposed new baseline algorithm *Type-GBFS* solves almost 200 more problems than baseline GBFS over all International Planning Competition problems. Type-LAMA, a new planner which integrates Type-GBFS into LAMA-2011, solves 36.8 more problems than LAMA-2011.[1]

Introduction

In the latest International Planning Competition (IPC) IPC-2011 (García-Olaya, Jiménez, and Linares López 2011), the planner LAMA-2011 (Richter and Westphal 2010) was the clear winner of the sequential satisficing track, by both measures of coverage and plan quality. LAMA-2011 finds a first solution using Greedy Best-First Search (GBFS) (Bonet and Geffner 2001; Helmert 2006) with popular enhancements such as Preferred Operators (Richter and Helmert 2009), Deferred Evaluation (Richter and Helmert 2009) and Multi-Heuristic (Richter and Westphal 2010).

GBFS always expands a node n that is closest to a goal state according to a heuristic h. While GBFS makes no guarantees about solution quality, it can often find a solution quickly. GBFS's performance strongly depends on h. Misleading or uninformative heuristics can result in massive increases in the time and memory complexity of search.

Two of the three enhancements above, Preferred Operators and Multi-Heuristic, are implemented in a *Multiple Queue Search* framework (Helmert 2006). Separate priority queues are used to hold different sets of nodes, or keep them sorted according to different heuristics. Still, each queue is sorted based on some heuristic h, and is used in a greedy fashion by the search, which always expands a node with minimum h-value from one of the queues. This makes search vulnerable to the *misleading heuristic* problem, where it can stall in *bad subtrees*, which contain large local minima or plateaus but do not lead to a solution. Adding exploration to a search algorithm is one way to attack this problem.

Previous approaches to this problem of GBFS with misleading heuristics include K-BFS (Felner, Kraus, and Korf 2003), which expands the first k best nodes in a single priority queue and adds all their successors, and Diverse-BFS (Imai and Kishimoto 2011), which expands extra nodes with non-minimal h-values or at shallow levels of the search tree. Another simple algorithm is ϵ-GBFS (Valenzano et al. 2014), which expands a node selected uniformly at random from the open list with probability ϵ. All these algorithms add an element of exploration.

The current paper proposes and evaluates a simple yet very effective way of adding exploration based on a *type system* (Lelis, Zilles, and Holte 2013). The major contributions are:

1. An analysis of the weaknesses of previous simple exploration schemes. and a non-greedy approach to exploration based on a simple type system.
2. A search algorithm under the framework of multiple-queue search named *Type-GBFS* which uses a type system for exploration, and the corresponding planner *Type-LAMA*, which replaces the GBFS component of LAMA-2011 by Type-GBFS.
3. Detailed experiments on IPC benchmarks, which demonstrate that baseline Type-GBFS solves substantially more problems than baseline GBFS, and that this superiority also holds when adding most combinations of standard planning enhancements. Type-LAMA with all such en-

Copyright © 2014, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

[1] Another version of this paper is published in AAAI-2014 (Xie et al. 2014).

hancements outperforms LAMA-2011.

The discussion starts with a review of Multiple Queue Search and the problems caused by misleading heuristics in best-first algorithms. An analysis of ϵ-GBFS shows that its exploration suffers from closely following the distribution of h-values in the open list. A simple type system based on both g- and h-values of each node is proposed as a remedy and shown to lead to better exploration. Experiments on the baseline as well as state of the art planners confirm that this leads to significant progress in coverage.

Background: Multiple Queue Search and Early Mistakes in GBFS

Multiple Queue Search

Two of the most important enhancements for satisficing planning, Preferred Operators and Multi-Heuristics, are based on *Multiple Queue Search* (Helmert 2006; Röger and Helmert 2010). When more than one heuristic is used in the search algorithm, Multiple Queue Search uses one priority queue for each heuristic, and selects the next node to expand from these queues in a round-robin manner. Queue boosting (Helmert 2006) can be used to assign priorities to different queues. Once a node is expanded, all its successors are evaluated by all heuristics, and put into every queue with the corresponding value. For Preferred Operators (or Helpful Actions), one additional priority queue per heuristic is used, which contains only the successors generated by a preferred operator. The algorithm again selects nodes in a round-robin manner from all queues, but boosts the preferred operator queue(s). The search benefits both from focusing more on usually relevant actions, and from reaching greater depths more quickly because of the smaller effective branching factor in the preferred queues.

Early Mistakes caused by Misleading Heuristics

Early mistakes are mistakes in search direction at shallow levels of the search tree caused by sibling nodes being expanded in the wrong order. This happens when the root node of a *bad subtree*, which contains no solution or only hard-to-find solutions, has a lower heuristic value than a sibling which would lead to a quick solution.

The 2011-Nomystery domain from IPC-2011 is a typical example where delete-relaxation heuristics systematically make early mistakes (Nakhost, Hoffmann, and Müller 2012). In this transportation domain with limited non-replenishable fuel, delete-relaxation heuristics such as h^{FF} ignore the crucial aspect of fuel consumption, which makes the heuristic overoptimistic and misleading, and results in large unrecognized dead-ends in the search space. Bad subtrees in the search tree, which over-consume fuel early on, are searched exhaustively, before any good subtrees which consume less fuel and can lead to a solution are explored. As a result, while the random walk-based planner Arvand with its focus on exploration solved 19 out of 20 nomystery instances in IPC-2011, LAMA-2011 solved only 10.

Exploration bias in the Open List: Two Case Studies

Previous exploration methods in GBFS suffer from biasing their exploration heavily towards the neighborhood of nodes in the open list. In the case of early mistakes, the large majority of these nodes is in useless regions of the search space. Consider the nodes in the regular h^{FF} (Hoffmann and Nebel 2001) open list of LAMA-2011 while solving the problem 2011-nomystery #12. Figure 1(a) shows snapshots of their h-value distribution after 2,000, 10,000 and 50,000 nodes expanded. In the figure, the x-axis represents different heuristic values and the y-axis represents the number of nodes with a specific h value in the open list. The solution eventually found by LAMA-2011 goes through a single node n in this 50,000 node list, with $h(n) = 18$. This node is marked with an asterisk in the figure. Over 99% of the nodes in the open list have lower h-values, and will be expanded first, along with much of their subtrees. However, in this example, none of those nodes leads to a solution. The open list is flooded with a large number of useless nodes with undetected dead ends.

ϵ-GBFS (Valenzano et al. 2014) samples nodes uniformly over the whole open list. This is not too useful when entries are heavily clustered in bad subtrees. In the example above, ϵ-GBFS has a less than 1% probability to pick a node with h-value 18 or more in its exploration step, which itself is only executed with probability ϵ. Furthermore, the algorithm must potentially select several good successor nodes before making measurable progress towards a solution by finding an exit node with a lower h-value.

The instance 2011-nomystery #12, with 6 locations and 6 packages, has a relatively small search space, and both GBFS and ϵ-GBFS eventually solve it after exhaustively enumerating the dead ends. However, a larger problem like 2011-nomystery #19, with 13 locations and 13 packages, is completely out of reach for GBFS or ϵ-GBFS. This instance was solved by only 2 planners in IPC-2011. Figure 1(b) shows the h-value distribution in LAMA-2011's regular h^{FF} queue after 20,000, 100,000 and 500,000 nodes. The node with $h = 39$ from a solution found by Arvand-2011 (Nakhost and Müller 2009) is marked at the far right tail of the distribution in the figure.

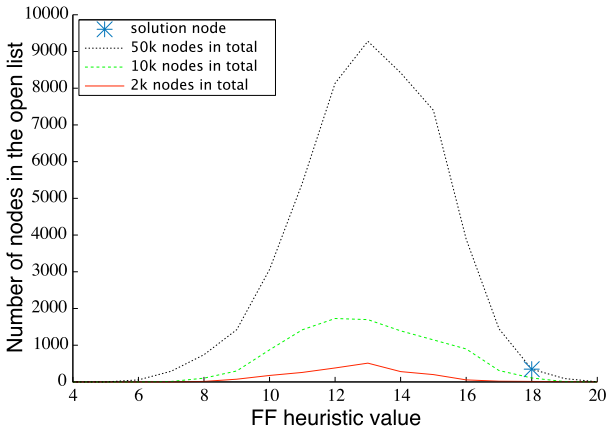
Adding Exploration via a Type System

Can the open list be sampled in a way that avoids the over-concentration on a cluster of very similar nodes? A *type system* (Lelis, Zilles, and Holte 2013), which is based on earlier ideas of stratified sampling (Chen 1992), is one possible approach.

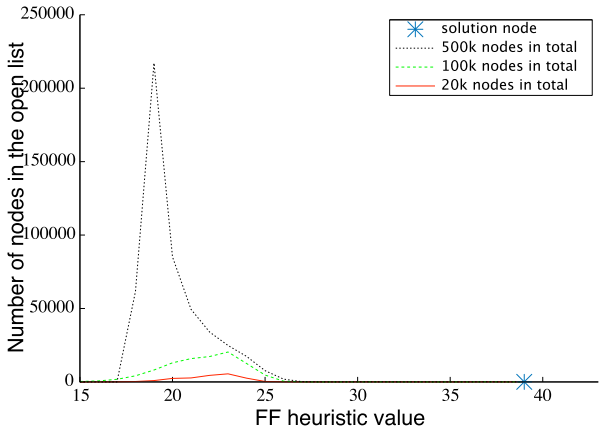
Type System

A *type system* is defined as follows:

Definition 1 *(Lelis, Zilles, and Holte 2013) Let S be the set of nodes in search space. $T = \{t_1, \ldots, t_n\}$ is a type system for S if T is a disjoint partitioning of S. For every $s \in S$, $T(s)$ denotes the unique $t \in T$ with $s \in S$.*



(a) h-values in nomystery #12



(b) h-values in nomystery #19

Figure 1: (a)(b): h-value distribution in the regular h^{FF} open list of LAMA-2011.

Types can be defined using any property of nodes. The simple type system used here defines the type of a node s in terms of its h-value for different heuristics h, and its g-value. A simple and successful choice is the pair $T(s) = (h^{FF}(s), g(s))$. The intuition behind such type systems is that they can roughly differentiate between nodes in different search regions, and help explore away from the nodes where GBFS gets stuck.

Figure 2(a) views a LAMA-2011 search of instance 2011-nomystery #19 through the lens of a (h^{FF}, g) type system. The horizontal x- and y-axes represent h^{FF}-values and g-values respectively. The number of nodes in the open list with a specific (h^{FF}, g) type is plotted on the vertical z-axis. The graph shows the frequency of each type in the regular h^{FF} open list of LAMA-2011 at the time when the open list first reaches 100,000 nodes. After initial rapid progress, search has stalled around a single huge peak. Most of the open list is filled with a large number of useless nodes.

Type-GBFS: Adding a Type System to GBFS

Type-GBFS uses a simple two level *type bucket* data structure tb which organizes its nodes in buckets according to their type. Type bucket-based node selection works as follows: First, pick a bucket b uniformly at random from among all the non-empty buckets. Then pick a node n uniformly at random from all the nodes in b. Type-GBFS alternately expands a node from from the regular open list O and from tb, and each new node is added to both O and tb.

Multi-Heuristic type systems $(h_1(s), h_2(s), ...)$ have been explored before using a *Pareto set* approach (Röger and Helmert 2010). The main differences of their approach are: 1) only *Pareto Optimal* buckets are selected; 2) the probability of selecting each Pareto optimal bucket is proportional to the number of nodes it contains; 3) only heuristics are used to define types, whereas the current approach also considers g and potentially any other relevant information; and 4) nodes in a bucket are selected deterministically in FIFO order, not uniformly at random.

Diverse Best-First Search (DBFS) (Imai and Kishimoto 2011) is another closely related high performance search algorithm which includes an exploration component. This two-level search algorithm uses a global open list O_G, a local open list O_L and a shared closed list.

Like Type-GBFS with the $(h_{FF}(s), g(s))$ type system, DBFS picks nodes based on their h- and g-values. There are three major differences between these algorithms. 1) DBFS performs a sequence of local searches while Type-GBFS defines a single global search; 2) DBFS uses g to restrict its node selection, while Type-GBFS can use g as part of its type system; 3) DBFS biases its node selection using h, while the type buckets in Type-GBFS sample uniformly over all types.

Algorithm 1 TypeBuckets.Insert(n)

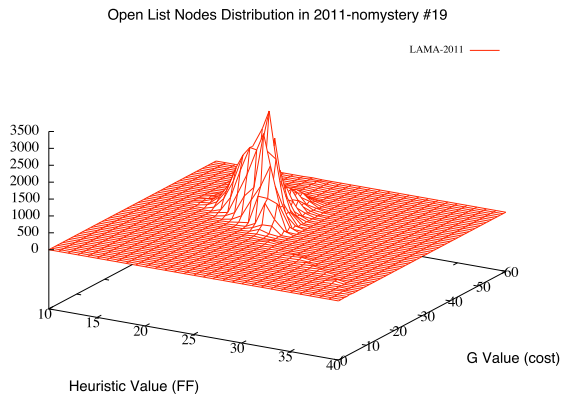
Input TypeBuckets b, node n

```
1: t ← T(n)
2: if not b.types.Contains(t) then
3:     b.types.Insert(t)
4: end if
5: b.nodes[t].Insert(n)
6: return
```

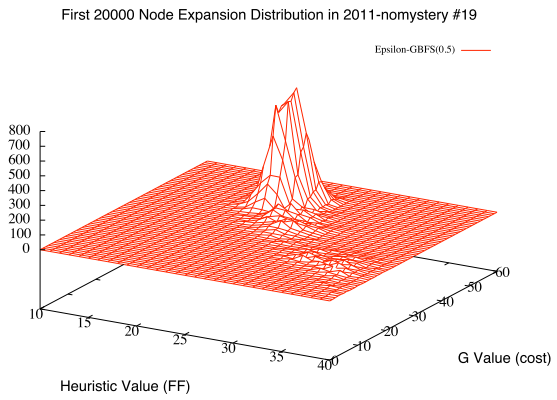
Algorithm 2 TypeBuckets.SelectNode()

Input TypeBuckets b
Output Node n

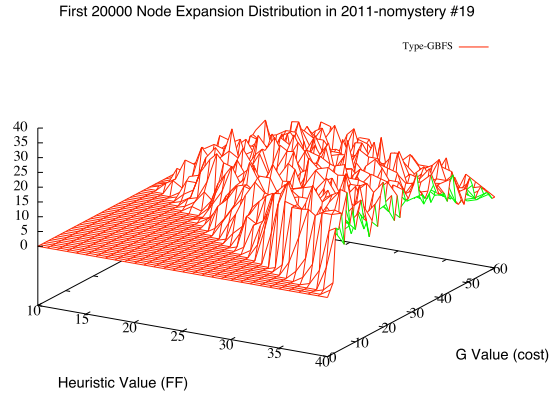
```
1: t ← b.types.SelectRandom()
2: n ← b.nodes[t].SelectRandom()
3: b.nodes[t].Remove(n)
4: if b.nodes[t].IsEmpty() then
5:     b.types.Remove(t)
6: end if
7: return n
```



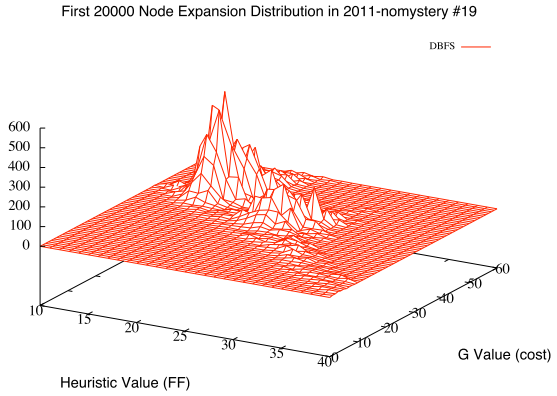
(a) (h, g)-distribution in 2011-nomystery #19 for LAMA



(b) ϵ-GBFS($\epsilon = 0.5$)



(c) Type-GBFS



(d) DBFS

Figure 2: (a): distribution of types in the regular h^{FF} open list of LAMA-2011 after 100,000 nodes in 2011-nomystery #19. (b)(c)(d): distribution of types over the first 20,000 nodes expanded by the exploring phase (ϵ-exploration or type buckets) of ϵ-GBFS($\epsilon = 0.5$), Type-GBFS and DBFS.

Exploration in Type-GBFS, ϵ-GBFS and DBFS

Type-GBFS and ϵ-GBFS with $\epsilon = 0.5$ both spend half their search effort on exploration. However, the distribution of types of the explored nodes is very different. In Nomystery-2011 #19, GBFS in LAMA-2011 grows the single peak shown in Figure 2(a). Figure 2(b-d) show the frequency of explored node types for ϵ-GBFS with $\epsilon = 0.5$, Type-GBFS[2] and DBFS after 20,000 nodes in the same format. Note that while Figure 1 shows the distribution of all nodes in the open list, Figure 2 shows the types of only those nodes that were chosen in the exploration step[3].

ϵ-GBFS mainly explores nodes close to the GBFS peak types, while Type-GBFS explores much more uniformly over the space of types. DBFS explores more types than ϵ-GBFS. Unlike type buckets in Type-GBFS, which sample types uniformly, DBFS is biased towards low h and high g values.

Note that the z-axis scales are different for the three plots. The single most explored type contains around 800 nodes for ϵ-GBFS and 600 for DBFS, but only 40 for Type-GBFS. The presence or absence of exploration helps explain the relative performance in 2011-Nomystery. The coverage for the 20 instances of this domain for one typical run with 4 GB memory and 30 minutes per instance is 9 for GBFS, 11 for ϵ-GBFS with $\epsilon = 0.5$, 17 for Type-GBFS and 18 for DBFS. Table 1 shows detailed search time results. While ϵ-GBFS slightly improves over GBFS, Type-GBFS outperforms both other algorithms. DBFS's exploration strategy also performs very well in 2011-Nomystery.

Experiments

The experiments use a set of 2112 problems (54 domains) from the seven International Planning Competitions, and were run on an 8-core 2.8 GHz machine with 4 GB memory and 30 minutes per instance. Results for planners which use randomization are averaged over five runs. All algorithms

[2]Some explored types are outside the (h, g) range shown in Figure 2(b).

[3]Unlike ϵ-GBFS and Type-GBFS, there is no clear exploration step in DBFS. All visited nodes are shown in the figure.

problem#	GBFS	ϵ-GBFS, $\epsilon = 0.5$	Type-GBFS	DBFS
1	0.02	0.03	0.03	0.02
2	0.03	0.54	0.32	0.05
3	58.97	0.1	3.45	1.17
4	42.7	100.85	9.11	2.05
5	31.47	54.21	1.9	1.36
6	1.44	DNF	79.92	8.21
7	160.54	466.89	121.2	3.56
8	DNF	DNF	1257.13	249.84
9	DNF	18.46	32.78	33.23
10	DNF	DNF	382.24	971.4
11	0.3	0.06	0.19	0.11
12	2.24	2.78	2.9	0.28
13	DNF	19.73	1.54	7.42
14	DNF	639.35	9.41	4.64
15	DNF	DNF	11.72	1.09
16	DNF	DNF	26.25	19.65
17	DNF	DNF	DNF	DNF
18	DNF	DNF	DNF	DNF
19	DNF	DNF	460.62	396.18
20	DNF	DNF	DNF	1287.59
total	9	11	17	18

Table 1: The search time (in seconds) of GBFS, ϵ-GBFS (0.5), Type-GBFS and DBFS on IPC-2011 Nomystery. "DNF" means "did not finish" within 30 minutes with 4G of memory.

are implemented using the Fast Downward code base FD-2011 (Helmert 2006). The translation from PDDL to SAS+ was done only once, and this common preprocessing time is not counted in the 30 minutes.

Performance of Baseline Algorithms

The baseline study evaluates the two algorithms GBFS and Type-GBFS without the common planning enhancements of preferred operators, deferred evaluation and multi-heuristics. It uses three popular planning heuristics - FF (Hoffmann and Nebel 2001), causal graph (CG) (Helmert 2004) and context-enhanced additive (CEA) (Helmert and Geffner 2008). We use the distance-base versions for the three heuristics. They estimate the length of a solution path starting from the evaluated state. Table 2 shows the coverage results. Type-GBFS outperforms GBFS by a substantial margin for each tested heuristic.

Heuristic	GBFS	Type-GBFS
FF	1561	**1755.6**
CG	1513	**1691.4**
CEA	1498	**1678.8**

Table 2: Baseline GBFS vs. Type-GBFS - coverage of 2112 IPC instances.

Figure 3 compares the time performance of the baseline algorithms with h^{FF}. Every data point represents one instance, with the search times for GBFS on the x-axis plotted against Type-GBFS on the y-axis. Only problems for which both algorithms need at least 0.1 seconds are shown.

Points below the main diagonal represent instances that Type-GBFS solves faster than GBFS. For ease of comparison, additional reference lines indicate $2\times$, $10\times$ and $50\times$ relative speed. Data points within a factor of 2 are greyed out in order to highlight the instances with substantial differences. Problems that were only solved by one algorithm within the 1800 second time limit are included at $x = 10000$ or $y = 10000$.

Beyond solving almost 200 more problems, Type-GBFS shows a substantial advantage over GBFS in search time for problems solved by both planners. There are many more problems where Type-GBFS outperforms GBFS by more than a factor of 10 or 50 than vice versa. Still, while Type-GBFS outperforms GBFS overall, it does not dominate it on a per-instance basis. Sometimes the extra exploration of Type-GBFS wastes time or even leads the search astray for a while.

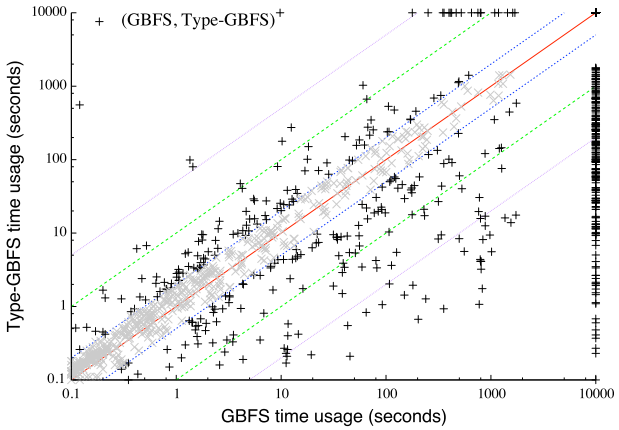


Figure 3: Search time of baseline GBFS and Type-GBFS with the h^{FF} heuristic on IPC. Results of one typical run for Type-GBFS shown.

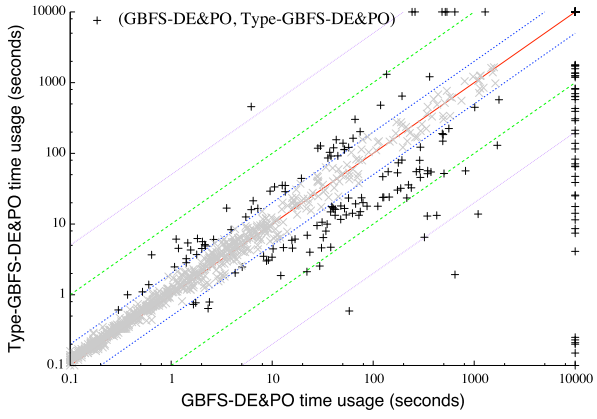


Figure 4: Comparison of search times of GBFS and Type-GBFS with FF heuristic, Deferred Evaluation and Preferred Operators. Similar to Figure 3, we use the results of a typical run for Type-GBFS with DE&PO.

Performance with Different Enhancements

How do GBFS and Type-GBFS compare when common planning enhancements are added? All combinations of Deferred Evaluation, Preferred Operators and Multiple Heuristics are tested, with h^{FF} as the primary heuristic.

- With **Deferred Evaluation**, nodes are not evaluated before adding them to open lists and type buckets. Instead, the heuristic value of their parents is used (Richter and Helmert 2009).
- With **Preferred Operators**, nodes that are reached via preferred operators, such as helpful actions in h^{FF}, are also stored in a separate open list (Richter and Helmert 2009). Boosting of preferred open lists with a parameter of 1000 is used as in LAMA-2011 (Richter and Westphal 2010). In case of deferred evaluation, preferred operators are ranked before other siblings for tie-breaking, using the so-called *pref_first* ordering (Richter and Westphal 2010). Type-GBFS uses only a single set of type buckets for all nodes. There are no separate type buckets containing preferred nodes only.
- **Multi-Heuristics** maintains multiple priority queues sorted by different heuristics. Following LAMA, the *Landmark count* heuristic h^{lm} (Richter, Helmert, and Westphal 2008) is used as the second heuristic here. Type-GBFS with Multi-Heuristics uses two open lists, one for each heuristic, plus type buckets for the (h^{FF}, g) type system.

When both Multi-Heuristic and Preferred Operators are applied, GBFS uses four queues, two regular and two preferred ones. Type-GBFS uses the same queues plus (h^{FF}, g) type buckets.

Enhancement	GBFS	Type-GBFS
(none)	1561	1755.6
PO	1826	1848.6
DE	1535	1834.6
MH	1851	1789.8
PO + DE	1871	1906.4
PO + MH	1850	1846.2
DE + MH	1660	1729.0
PO + DE + MH	**1913**	**1949.8**

Table 3: Number of IPC tasks solved out of 2112. PO = Preferred Operators, DE = Deferred Evaluation, MH = Multi-Heuristic.

Table 3 shows the experimental results on IPC domains for all 8 combinations of enhancements. When used as a single enhancement, Preferred Operators and Multiple Heuristic improve both algorithms. Deferred Evaluation also strongly improves Type-GBFS, but causes a slight decrease in coverage for GBFS, mainly due to plateaus caused by the less informative node evaluation (Richter and Helmert 2009). Apparently, Type-GBFS gets stuck in such plateaus less often.

When combining any two enhancements, both algorithms achieve their best performance with Preferred Operators plus Deferred Evaluation, as observed for GBFS in Richter and Helmert's work (2009). Figure 4 shows details of time usage for GBFS and Type-GBFS for this combination. Multiple Heuristics have a negative effect on Type-GBFS when combined with either Preferred Operators or Deferred Evaluation, but work very well when combined with both. Finding an explanation for this surprising behavior is left as future work. The last row in Table 3 lists coverage results when all three enhancements are applied as in LAMA.

Comparing State of the Art Planners in Terms of Coverage and Search Time

The performance comparison in this section includes the following planners:

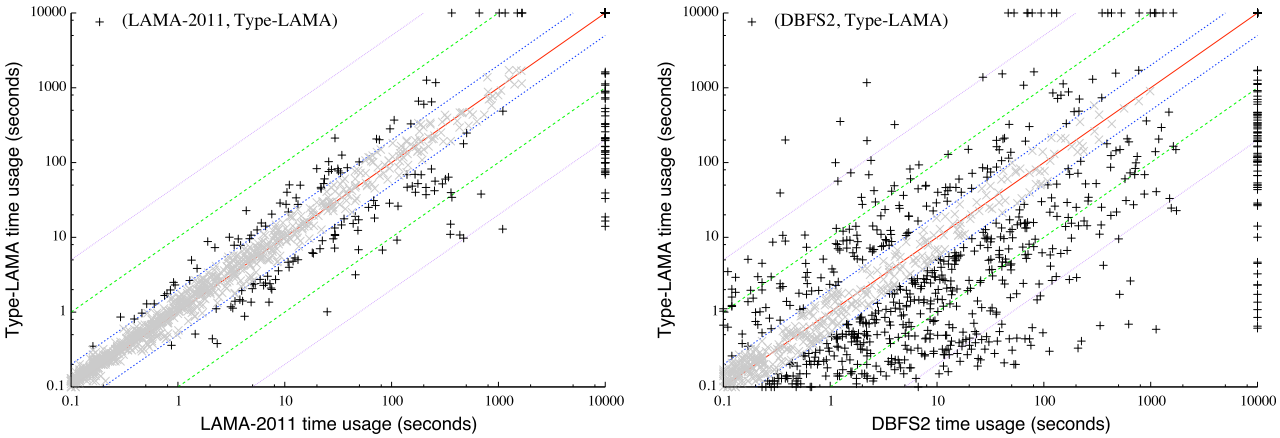
- **LAMA-2011:** only the first iteration of LAMA using GBFS is run, with deferred evaluation, preferred operators and multi-heuristics (h^{FF}, h^{lm}) (Richter and Westphal 2010).
- **Type-LAMA:** LAMA-2011 with GBFS replaced by Type-GBFS, uses the same four queues as LAMA-2011, plus (h^{FF}, g) type buckets.
- **DBFS2:** DBFS2 (Imai and Kishimoto 2011), an enhanced version of DBFS which adds a second global open list for preferred operators only, was re-implemented by Imai on the LAMA-2011 code base, which is much stronger than the LAMA-2008 code base used in (Imai and Kishimoto 2011). The parameters $P = 0.1, T = 0.5$ from the same paper are used.

LAMA-2011 and Type-LAMA correspond to PO+DE+MH in Table 3.

Table 4 shows detailed coverage differences of these three planners. Type-LAMA outperforms the other two planners, solving 36.8 more problems than LAMA-2011 and 54.4 more than DBFS2. The 5 runs for Type-LAMA had nearly identical coverage of 1952, 1950, 1950, 1948 and 1949. The percentage of unsolved problems is reduced from 9.4% for LAMA to 7.6%. This is comparable to the improvement from adding the *Landmark count* heuristic (Richter and Westphal 2010) - from 11.4% to 9.4% unsolved. Considering that only the hardest problems were left unsolved, adding the type system makes the planner substantially stronger.

Figures 5(a) and (b) compare the search time of Type-LAMA against LAMA-2011 and DBFS2 in the manner described for Figure 3. Between Type-LAMA and LAMA-2011, many results are very close, presumably for instances where exploration plays only a small role. Type-LAMA has a large time advantage of over $10\times$ more often. Results for Type-LAMA and DBFS2 are much more diverse. Besides its coverage advantage, Type-LAMA also wins the time comparison for a large number of instances by factors of over $2\times$, $10\times$ and $50\times$.

For further comparison, the coverage results of some other strong planners from IPC-2011 on the same hardware are: FDSS-2 solves 1912/2112, Probe 1706/1968 (failed on ":derive" keyword in 144 problems), Arvand 1878.4/2112, fd-auto-tune-2 1747/2112, and Lama-2008 1809/2112.



(a) Lama-2011 (x) vs. Type-LAMA (y)

(b) DBFS2 (x) vs. Type-LAMA (y)

Figure 5: Comparison of search time. Type-LAMA vs. Lama-2011 (left) and DBFS2 (right). using typical single runs of Type-GBFS, Type-LAMA and DBFS2.

domain	size	LAMA-2011	Type-LAMA	DBFS2
98-logistics	35	**35**	**35**	34
98-mystery	30	**19**	**19**	**19**
00-miconic-full	150	136	**139**	138.6
02-depot	22	20	**21.6**	18.2
02-freecell	80	78	77.8	**79.8**
04-airport-strips	50	32	34.6	**41**
04-notankage	50	**44**	**44**	43.2
04-optical-tele	48	4	**5.4**	5
04-philosopher	48	39	**48**	**48**
04-psr-large	50	**32**	31.6	15.6
04-satellite	36	**36**	35.2	27
06-pathways	30	**30**	**30**	28.4
06-storage	30	18	**23.8**	23.4
06-tankage	50	41	**42**	36.8
06-trucks-strips	30	14	20.8	**24**
08-scanalyzer	30	**30**	**30**	29.6
08-sokoban	30	**28**	27	**28**
08-transport	30	29	**30**	29.8
08-woodworking	30	**30**	29.8	**30**
11-elevators	20	**20**	**20**	18.8
11-floortile	20	6	5.6	**7.6**
11-nomystery	20	10	**17.8**	**17.8**
11-openstacks	20	**20**	**20**	12.8
11-parking	20	**18**	**17.6**	12.6
11-scanalyzer	20	**20**	**20**	19.8
11-sokoban	20	**19**	18.2	18
11-tidybot	20	16	**16.4**	16.2
11-transport	20	**16**	15.6	12.4
11-visitall	20	**20**	**20**	7
Total	2112	1913	**1949.8**	1895.4
Unsolved	-	199	**162.2**	216.6

Table 4: Number of instances solved. 25 domains with 100% coverage for all three planners omitted.

Although type buckets cause some change in solution costs, the influence is not clear-cut. If we compare LAMA-2011 and one typical run Type-LAMA's results, there are 1907 problems solved by both planners. The IPC scores for LAMA-2011 and Type-LAMA are 1895.1 vs 1892.5, only 2.6 difference over the 1907 problems. For 1698 problems, both planners achieve the same solution cost.

Effect of Different Type Systems

The results above for both Type-GBFS and Type-LAMA are for the (h^{FF}, g) type system. Table 5 summarizes results for these two planners when using several other simple type systems. (1) is the trivial single-type system $T(s) = 1$.

Among single-element type systems, (g) performs better than either heuristic, and (h^{lm}) solves around 10 more problems than (h^{FF}). Since Type-GBFS only uses h^{FF}, h^{lm} is only tested for Type-LAMA.

Compared to GBFS, (g) explores much more on nodes with low g-values, typically at shallow levels of the search tree. Many such nodes will be expanded very late in GBFS. In contrast, an (h)-only type system focuses on exploring different estimated goal distances and ignores g. Interestingly, (g) is even slightly better than (h^{FF}, g) in Type-GBFS, but (h^{FF}, g) is better in Type-LAMA. Among two-element type systems, (h^{FF}, g) and (h^{lm}, g) are the top two configurations, while (h^{FF}, h^{lm}) is just slightly better than (h^{lm}). The three-element type system (h^{FF}, h^{lm}, g) is worse and might be too fine-grained for this test set. The question of the right granularity is important and needs further study.

Type	T-GBFS	T-LAMA	Type	T-LAMA
none	1561	1913	(h^{lm})	1921.6
(1)	1529.6	1916.2	(h^{lm}, g)	1942.4
(g)	**1758.2**	1935.0	(h^{FF}, h^{lm})	1925.6
(h^{FF})	1729.0	1918.6	(h^{FF}, h^{lm}, g)	1939.0
(h^{FF}, g)	1755.6	**1949.8**		

Table 5: Coverage of Type-GBFS (T-GBFS) and Type-LAMA (T-LAMA) with simple type systems.

Type-LAMA Works Better as an Integrated System than as a Simple Portfolio

This experiment compares Type-LAMA, which integrates type-based exploration directly into LAMA's search process, with a portfolio which independently runs LAMA and a simple type-based planner ST. ST selects nodes exclusively from type buckets, using a (h^{FF}, g) type system as defined above. For consistency with LAMA, Deferred Evaluation is used for type buckets as well.

By itself, ST solves 1266 out of 2112 IPC problems. Consider a portfolio planner that uses x seconds for LAMA-2011, followed by $1800 - x$ seconds for ST. The best coverage of 1926 is achieved for $x = 1279$. Type-LAMA , whose performance is shown as a horizontal line near the top of the plot, solves 23.8 problems more than this best portfolio. This shows the synergy between exploitation-based search in LAMA and exploration using a type system.

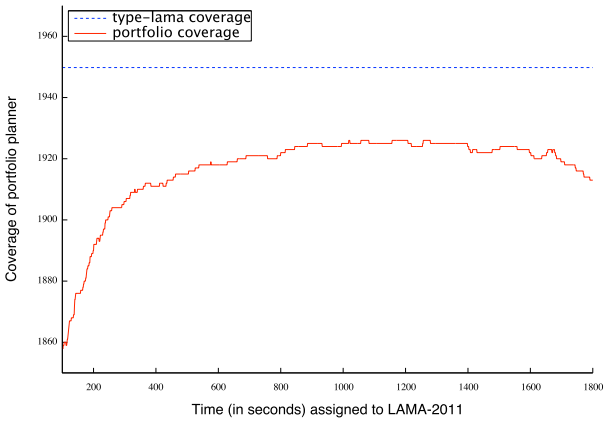


Figure 6: Coverage of LAMA+ST portfolio planner with varying time allocation.

Conclusion and Future Work

The primary contributions of this paper are an investigation of the problem of inefficient exploration in previous GBFS-type planners, and the solution of using type-based exploration. The new algorithm Type-GBFS samples nodes uniformly over a type system instead of uniformly over all nodes in an open list. By replacing GBFS with Type-GBFS, the planner Type-LAMA can solve 36.8 more problems than LAMA-2011 on average, decreasing the number of unsolved problems over all past IPC domains from 199 to 162.2.

One obvious direction for future work is to test many more type systems, including others proposed by Lelis, Zilles and Holte (2013). Different forms of boosting and non-uniform exploration of different types could also be investigated. Regarding experimental results, it is unclear why the combination of multiple heuristics with either one of preferred operators and deferred evaluation fails, and yet succeeds when combined with both.

One potential problem of Type-LAMA is that the type system might not be able to explore deeply enough when the distance from current nodes in the open list to heuristically promising nodes is large. One potential solution for this problem is to use a larger local search (Imai and Kishimoto 2011; Xie, Müller, and Holte 2014), or other forms of exploration such as random walks (Nakhost and Müller 2009).

Acknowledgements

The authors wish to thank the anonymous referees for their valuable advice. This research was supported by GRAND NCE, a Canadian federally funded Network of Centres of Excellence, NSERC, the Natural Sciences and Engineering Research Council of Canada, AITF, Alberta Innovates Technology Futures, and Grant-in-Aid for JSPS Fellows, Japan Society for the Promotion of Science.

References

Bonet, B., and Geffner, H. 2001. Heuristic search planner 2.0. *AI Magazine* 22(3):77–80.

Chen, P. C. 1992. Heuristic sampling: A method for predicting the performance of tree searching programs. *SIAM J. Comput.* 21(2):295–315.

Felner, A.; Kraus, S.; and Korf, R. E. 2003. KBFS: K-best-first search. *Ann. Math. Artif. Intell.* 39(1-2):19–39.

García-Olaya, A.; Jiménez, S.; and Linares López, C., eds. 2011. *The 2011 International Planning Competition*. Universidad Carlos III de Madrid.

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

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

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

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

Imai, T., and Kishimoto, A. 2011. A novel technique for avoiding plateaus of greedy best-first search in satisficing planning. In Burgard, W., and Roth, D., eds., *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence (AAAI-2011)*, 985–991.

Lelis, L. H. S.; Zilles, S.; and Holte, R. C. 2013. Stratified tree search: a novel suboptimal heuristic search algorithm. In Gini, M. L.; Shehory, O.; Ito, T.; and Jonker, C. M., eds., *Proceeding of the 12th International Conference on Autonomous Agents and Multi-Agent Systems*, 555–562.

Nakhost, H., and Müller, M. 2009. Monte-Carlo exploration for deterministic planning. In Walsh, T., ed., *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI-2011)*, 1766–1771.

Nakhost, H.; Hoffmann, J.; and Müller, M. 2012. Resource-constrained planning: A Monte Carlo random walk approach. In McCluskey, L.; Williams, B.; Silva, J. R.; and Bonet, B., eds., *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS-2012)*, 181–189.

Richter, S., and Helmert, M. 2009. Preferred operators and deferred evaluation in satisficing planning. In Gerevini, A.; Howe, A. E.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS-2009)*, 273–280.

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

Richter, S.; Helmert, M.; and Westphal, M. 2008. Landmarks revisited. In Rintanen, J.; Nebel, B.; Beck, J. C.; and Hansen, E. A., eds., *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS-2008)*, 975–982.

Röger, G., and Helmert, M. 2010. The more, the merrier: Combining heuristic estimators for satisficing planning. In Brafman, R. I.; Geffner, H.; Hoffmann, J.; and Kautz, H. A., eds., *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS-2010)*, 246–249.

Valenzano, R.; Schaeffer, J.; Sturtevant, N.; and Xie, F. 2014. A comparison of knowledge-based GBFS enhancements and knowledge-free exploration. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS-2014)*.

Xie, F.; Müller, M.; Holte, R.; and Imai, T. 2014. Type-based exploration with multiple search queues for satisficing planning. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI-2014)*.

Xie, F.; Müller, M.; and Holte, R. 2014. Adding local exploration to greedy best-first search in satisficing planning. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI-2014)*.

A Practical, Integer-Linear Programming Model for the Delete-Relaxation in Cost-Optimal Planning

Tatsuya Imai
Tokyo Institute of Technology
Japan

Alex Fukunaga
The University of Tokyo
Japan

Abstract

We propose a new integer-linear programming model for the delete relaxation in cost-optimal planning. While a naive formulation of the delete relaxation as IP is impractical, our model incorporates landmarks and relevance-based constraints, resulting in an IP that can be used to directly solve the delete relaxation. We show that our IP model outperforms the previous state-of-the-art solver for delete-free problems. We then use LP relaxation of the IP as a heuristics for a forward search planner, and show that our LP-based solver is competitive with the state-of-the-art for cost-optimal planning.

[This is the HSDIP workshop version of a paper that will appear in ECAI-2014 (Imai and Fukunaga 2014) (identical except for formatting). When citing this work please cite the ECAI paper.]

1 Introduction

The *delete relaxation* of a classical planning problem is a relaxation of a planning problem such that all deletions are eliminated from its operators. It is clear that h^+, the optimal value of the delete relaxation of a planning instance is an admissible, lower bound on the cost of the optimal cost plan for the instance.

In cost-optimal planning, while h^+ is known to be more accurate than commonly used heuristics such as landmark-cut (Helmert and Domshlak 2009), current planners to not directly compute h^+ because the extra search efficiency gained from using h^+ is offset by the high cost of computing h^+. In fact, computing h^+ is known to be NP-complete (Bylander 1994). As far as we are aware, the first use of h^+ inside a cost-optimal planner was by Betz and Helmert (Betz and Helmert 2009), who implemented domain-specific implementations of h^+ for several domains. Haslum evaluated the use of a domain-independent algorithm for h^+ (Haslum, Slaney, and Thiébaux 2012) as the heuristic function for cost-optimal planning, and found that the performance was relatively poor (Haslum 2012). In recent years, there have been several advances in the computation of h^+ (Gefen and Brafman 2012; Pommerening and Helmert 2012; Haslum, Slaney, and Thiébaux 2012).

A somewhat separate line of research is the increasing use of integer/linear programming (ILP) in domain-independent planning. The earliest use of linear programming (LP) in domain-independent planning that we are aware of was by Bylander, who used an LP encoding of planning as a heuristic function for a partial order planner (Bylander 1997). Briel and Kambhampati formulated and solved planning as an integer program (IP) (van den Briel and Kambhampati 2005). Recently, instead of modeling and directly solving planning as an IP, LP relaxations have been used to compute admissible heuristics in a search algorithm, including a network flow a LP heuristic for branch-and-bound (van den Briel, Vossen, and Kambhampati 2008), a heuristic for A* based on the state equations in SAS+ (Bonet 2013), and most recently, an LP encoding of a broad framework for operator-counting heuristics (Pommerening et al. 2014). IP has also been used to compute hitting sets as part of the computation of h^+ in delete-free planning (in an improved version of the algorithm described in (Haslum, Slaney, and Thiébaux 2012), (Haslum 2014).

In this paper, we propose a new, integer/linear programming approach to computing h^+. While a straightforward ILP model for h^+ is often intractable and not useful in practice, we developed an enhanced model, $\mathrm{IP}^{\mathrm{e}}(T^+)$, which incorporates landmark constraints for the delete relaxation, as well as relevance analysis to significantly decrease the number of variables. We show that $\mathrm{IP}^{\mathrm{e}}(T^+)$ allows significantly faster computation of h^+ compared to the state of the art.

Then, we consider the use of h^+ as a heuristic for A* in a cost-optimal, domain-independent planner. We further augment $\mathrm{IP}^{\mathrm{e}}(T^+)$ with constraints that consider some delete effects, as well as constraints for cycle avoidance, resulting in a new admissible heuristic which dominates h^+. Since $\mathrm{IP}^{\mathrm{e}}(T^+)$ is an IP, its LP relaxation, $\mathrm{LP}^{\mathrm{e}}(T^+)$, is also an admissible heuristic for domain-independent problem. Since even $\mathrm{LP}^{\mathrm{e}}(T^+)$ can be quite expensive, the ILP model can be further relaxed by omitting a subset of its constraints, resulting in $\mathrm{LP}^{\mathrm{e}}_{\mathrm{tr}}(T^+)$, an LP for the "relaxed" delete relaxation.

We empirically evaluate the ILP models by embedding them as heuristics in an A*-based planner. We implemented a simple method for automatically selecting which LP formulation to use as the heuristic, based on a comparison of their values at the root node. The resulting planner performs comparably to the state-of-the-art, cost-optimal planners, Fast-Downward with the landmark-cut heuristic (Helmert and Domshlak 2009) and Fast-Downward using the hy-

Copyright © 2014, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

brid bisimulation merge-and-shrink heuristic (Nissim, Hoffmann, and Helmert 2011).

The rest of the paper is organized as follows. Section 2 proposes the basic ILP model for h^+. Section 3 describes enhancements to the ILP model which significantly speeds up computation of h^+. Section 4 augments the ILP model by adding counting constraints, which results in a IP bound that dominates h^+. Section 5 summarizes the relationship among ILP models, and describes a simple method for selecting which model to apply to a given problem instance. Section 6, experimentally evaluates the proposed ILP models, as well as a portfolio approach that automatically selects one of the ILP models.

2 ILP model for h^+

A STRIPS planning task is defined by a 4-tuple $T = \langle P, A, I, G\rangle$. P is a set of *propositions*. A is a set of *actions*. A *state* is represented by a subset of P, and applying an action to a state adds some propositions and removes some propositions in the state. Each action $a \in A$ is composed of three subsets of P, $\langle \mathrm{pre}(a), \mathrm{add}(a), \mathrm{del(a)}\rangle$ which are called the *preconditions*, *add effects*, and *delete effects*. An action a is applicable to a state S iff it satisfies $\mathrm{pre}(a) \subseteq S$. By applying a to S, propositions in S change from S to $S(a) = ((S \setminus \mathrm{del}(a)) \cup \mathrm{add}(a))$. For a sequence of actions $\pi = (a_0, \cdots, a_n)$, we use $S(\pi)$ to denote $((((S \setminus \mathrm{del}(a_0)) \cup \mathrm{add}(a_0)) \setminus \mathrm{del}(a_1)) \cup \cdots) \cup \mathrm{add}(a_n)$.

Let $I \subseteq P$ be the *initial state* and $G \subseteq P$ the *goal*. The target of a planning task is to find a sequence of actions to transform I to a state S that satisfies $G \subseteq S$. Formally, a feasible solution, i.e., a *plan*, is a sequence of actions $\pi = (a_0, \cdots, a_n)$ that satisfies (i) $\forall i, \mathrm{pre}(a_i) \subseteq I((a_0, \cdots, a_{i-1}))$, and (ii) $G \subseteq I(\pi)$. The target of a cost-optimal STRIPS planning is to find a shortest plan, or to find a plan π that minimizes $\sum_{a\in\pi} c(a)$ when the non-negative *cost* $c(a)$ of each action a is defined.

The delete relaxation of a task T, denoted by T^+, is a task $\langle P, A^+, I, G\rangle$ where A^+ is a set of delete-free actions defined as $A^+ = \{\langle \mathrm{pre}(a), \mathrm{add}(a), \emptyset\rangle \mid a \in A\}$. We also use T^+ to denote a task that is delete-free from the beginning without being relaxed.

2.1 ILP formulation of a delete-free problem

We formulate a delete free task $T^+ = \langle P, A^+, I, G\rangle$ as an integer-linear program. $\mathrm{IP}(T^+)$ denotes the IP problem derived from T^+, and we use $\pi^* = (a_0^*, \cdots, a_n^*)$ to denote an optimal plan for T^+ derived from an optimal solution of $\mathrm{IP}(T^+)$. Similarly $\mathrm{LP}(T^+)$ denotes the LP relaxation of $\mathrm{IP}(T^+)$. Note that for any feasible and non-redundant (i.e., same actions appear only once) solution of $\mathrm{IP}(T^+)$ (not just the optimal solution), we can derive a corresponding, feasible plan for T^+ that has same cost as the $\mathrm{IP}(T^+)$ solution.

First, we define the variables of $\mathrm{IP}(T^+)$. In addition to being able to derive a plan from $\mathrm{IP}(T^+)$, there always exists a injective mapping from a feasible non-redundant plan to an $\mathrm{IP}(T^+)$ solution. Thus, we also show the feasible assignments of variables that can be derived from a feasible plan of T^+, as well as the meanings and roles of the variables.

proposition: $\forall p \in P, \mathcal{U}(p) \in \{0, 1\}$. $\mathcal{U}(p) = 1$ iff $p \in I(\pi^*)$.

action: $\forall a \in A, \mathcal{U}(a) \in \{0, 1\}$. $\mathcal{U}(a) = 1$ iff $a \in \pi^*$ holds.

add effect: $\forall a \in A, \forall p \in \mathrm{add}(a), \mathcal{E}(a, p) \in \{0, 1\}$. $\mathcal{E}(a, p) = 1$ iff $a \in \pi^*$ holds and a achieves p first.

time (proposition): $\forall p \in P, \mathcal{T}(p) \in \{0, \cdots, |A|\}$. $\mathcal{T}(p) = t$ when $p \in I(\pi^*)$ and p is added by a_{t-1}^* first. $\mathcal{T}(p) = 0$ for $p \notin I(\pi^*)$.

time (action): $\forall a \in A, \mathcal{T}(a) \in \{0, \cdots, |A|-1\}$. $\mathcal{T}(a) = t$ when $a = a_t^*$. $\mathcal{T}(a) = |A| - 1$ when $a \notin \pi^*$.

initial proposition: $\forall p \in P, \mathcal{I}(p) \in \{0, 1\}$. $\mathcal{I}(p) = 1$ iff $p \in I$.

If $p \in P$ appears more than once, use first indices for $\mathcal{T}(p)$. Variables $\mathcal{I}(p)$ are auxiliary variables for computing h^+. Although they are redundant when solving a delete-free task only one time, they are useful to avoid reconstructing constraints for each state when $\mathrm{IP}(T^+)$ or $\mathrm{LP}(T^+)$ are embedded as a heuristic function in a forward-search planner and called for each state.

The objective function seeks to minimize $\sum_{a\in A} c(a)\mathcal{U}(a)$.

Because of this objective function, the cost of an IP solution is equal to the cost of the corresponding (delete-free) plan.

Finally we define following six constraints.

1. $\forall p \in G$, $\mathcal{U}(p) = 1$.
2. $\forall a \in A$, $\forall p \in \mathrm{pre}(a), \mathcal{U}(p) \geq \mathcal{U}(a)$.
3. $\forall a \in A$, $\forall p \in \mathrm{add}(a), \mathcal{U}(a) \geq \mathcal{E}(a, p)$.
4. $\forall p \in P$, $\mathcal{I}(p) + \sum_{a\in A \text{ s.t.} p\in\mathrm{add}(a)} \mathcal{E}(a, p) \geq \mathcal{U}(p)$.
5. $\forall a \in A$, $\forall p \in \mathrm{pre}(a)$, $\mathcal{T}(p) \leq \mathcal{T}(a)$.
6. $\forall a \in A$, $\forall p \in \mathrm{add}(a), \mathcal{T}(a) + 1 \leq \mathcal{T}(p) + (|A| + 1)(1 - \mathcal{E}(a, p))$.

There exists a feasible plan only if $\mathrm{IP}(T^+)$ has a feasible solution. When $\mathrm{IP}(T^+)$ is solved optimally, an optimal plan for T^+ is obtained according to following lemma. For a variable $\mathcal{V}$ of $\mathrm{IP}(T^+)$, $\mathcal{V}_F$ describes the assignment of $\mathcal{V}$ on a solution F of $\mathrm{IP}(T^+)$.

Proposition 1. *Given a feasible solution F for $\mathrm{IP}(T^+)$, the action sequence obtained by ordering actions in the set $\{a \mid \mathcal{U}(a)_F = 1\}$ in ascending order of $\mathcal{T}(a)_F$ is a feasible plan for T^+.*

Proof: At first we show that π satisfies the condition (ii) of a plan (i.e., $G \subseteq I(\pi)$) by proof of contradiction. Assume that there exists a proposition $g \in G$ that satisfies $g \notin I(\pi)$. There exists no action achieving g in π according to the assumption. Since F is a solution of $\mathrm{IP}(T^+)$, $\mathcal{U}(g)_F = 1$ holds according the constraint 1. Since $g \notin I(\pi)$ deduces $g \notin I$, $\mathcal{I}(g)_F = 0$. Therefore, to satisfy the condition 4, there must exist an action $a \in A$ that satisfies $g \in \mathrm{add}(a)$ and $\mathcal{E}(a, g)_F = 1$. However, to satisfy the constraint 3, $\mathcal{U}(a)_F = 1$ has to hold. This means $a \in \pi$, and this contradicts the assumption.

Next we show that π satisfies condition (i) (i.e., $\forall i, \mathrm{pre}(a_i) \subseteq I((a_0, \cdots, a_{i-1}))$). For the base case of inductive proof, assume that there exists a proposition $p \in P$ satisfying $p \in \mathrm{pre}(a_0)$ and $p \notin I$. Since $a_0 \in \pi, \mathcal{U}(a_0)_F = 1$ has to hold, and $\mathcal{U}(p)_F = 1$ has to hold according to the constraint $\mathcal{U}(p)_F \geq \mathcal{U}(a_0)_F$. Then, similar to the proof of condition (ii), there must exist an action $a \in A$ that satisfies $p \in \mathrm{add}(a), \mathcal{U}(a)_F = 1$, and $\mathcal{E}(a, p)_F = 1$. However, to satisfy constraint 5, $\mathcal{T}(p) \leq \mathcal{T}(a_0)$ has to be true, and $\mathcal{T}(a) + 1 \leq \mathcal{T}(p)$ has to hold to satisfy condition 6. Therefore we have $\mathcal{U}(a)_F = 1$ and $\mathcal{T}(a) < \mathcal{T}(a_0)$, but a_0 is the first action of π, a contradiction.

Similar to the case of $i = 0$, when $i > 0$, if $\mathrm{pre}(a_i) \subseteq I((a_0, \cdots, a_{i-1}))$ is not true, there must exist an action $a \notin (a_0, \cdots, a_{i-1})$ that satisfies $\mathcal{U}(a)_F = 1$ and $\mathcal{T}(a) < \mathcal{T}(a_i)$, contradicting the fact that a_i is the i-th action of the sequence π. □

3 Enhancements for ILP model

In this section, we introduce some variable elimination techniques and some modifications of constraints. As we will show in the experimental results, these enhancements significantly reduce the time to solve $\mathrm{IP}(T^+)$ and $\mathrm{LP}(T^+)$. Some of the enhancements are adopted into our IP framework from previous work in planning research. In particular, landmarks, which have been extensively studied in recent years, play very important role.

Note that while some of the enhancements introduce cuts that render some solutions of $\mathrm{IP}(T^+)$ mapped from feasible plans infeasible, *at least one optimal plan will always remain*.

3.1 Landmark Extraction and Substitution

A *landmark* is an element which needs to be used in every feasible solution. We use two kinds of landmarks, called *fact landmarks* and *action landmarks* as in (Gefen and Brafman 2012). A fact landmark of a planning task T is a proposition that becomes true on some state of every feasible plan, and an action landmark of a planning task T is an action that is included in every feasible plan. We also say that a fact or action landmark l is a *landmark of a proposition* p if l is a landmark of the task $\langle P, A, I, \{p\}\rangle$. Similarly we also say that a landmark l is a *landmark of an action* a if l is a landmark of the task $\langle P, A, I, \mathrm{pre}(a)\rangle$. In the IP model of a delete-free task T^+, if a proposition p is a fact landmark, then we can substitute $\mathcal{U}(p) = 1$. Similarly, if an action a is an action landmark, then we can substitute $\mathcal{U}(a) = 1$.

In this work, we extract some kinds of action landmarks and fact landmarks according to following facts. The contrapositions of these propositions are clearly true.

Proposition 2. *Given a feasible delete-free task T^+, an action $a \in A$ is an action landmark of T^+ if the task $\langle P, A \setminus \{a\}, I, G\rangle$ is infeasible.*

Proposition 3. *Given a feasible delete-free task T^+, a proposition $p \in P$ is a fact landmark of T^+ if the task $\langle P, A \setminus A_p^{\mathrm{add}}, I \setminus \{p\}, G\rangle$ is infeasible, where A_p^{add} is defined as $A_p^{\mathrm{add}} = \{a \mid p \in \mathrm{add}(a)\}$.*

Zhu et al. defined a kind of fact landmark called *causal landmark* (Zhu and Givan 2003). A proposition p is a causal landmark if $\langle P, A \setminus A_p^{\mathrm{pre}}, I \setminus \{p\}, G\rangle$ is infeasible, where $A_p^{\mathrm{pre}} = \{a \mid p \in \mathrm{pre}(a)\}$. If $\langle P, A \setminus A_p^{\mathrm{pre}}, I \setminus \{p\}, G\rangle$ does not have any solution, then $\langle P, A \setminus A_p^{\mathrm{add}}, I \setminus \{p\}, G\rangle$ is also infeasible, therefore using A_p^{add} instead of A_p^{pre} can extract larger set of fact landmarks. Keyder et al. proposed AND-OR graph based landmark extracting method generalized from a causal landmark extracting algorithm proposed Zhu et al. (Keyder, Richter, and Helmert 2010). We use similar algorithm to extract both of our fact landmarks and action landmarks.

3.2 Relevance Analysis

Backchaining relevance analysis is widely used to eliminate irrelevant propositions and actions of a task. An action a is relevant if (i) $\mathrm{add}(a) \cap G \neq \emptyset$, or (ii) there exists a relevant action a' satisfying $\mathrm{add}(a) \cap \mathrm{pre}(a') \neq \emptyset$. A proposition p is relevant if (i) $p \in G$, or (ii) there exists a relevant action a and $p \in \mathrm{pre}(a)$ holds. In addition to this, as Haslum et al. noted, it is sufficient to consider relevance with respect to only a subset of first achievers of add effect (Haslum, Slaney, and Thiébaux 2012). Although they defined a first achiever by achievability of a proposition, it is completely equivalent to the following definition: an action a is a first achiever of a proposition p if $p \in \mathrm{add}(a)$ and p is not a fact landmark of a. When we use $\mathrm{fadd}(a)$ to denote $\{p \in \mathrm{add}(a) \mid a \text{ is a first achiever of } p\}$, it is sufficient to use fadd instead of add on the above definition of relevance.

If $a \in A$ or $p \in P$ is not relevant, we can eliminate a variable as $\mathcal{U}(a) = 0$ or $\mathcal{U}(p) = 0$. In addition to this, if $p \in \mathrm{add}(a)$ but a is not a first achiever of p, we can eliminate a variable as $\mathcal{E}(a, p) = 0$.

3.3 Dominated Action Elimination

On a delete-free task, if two actions have same add effect, then it is clearly sufficient to use at most one of two actions. Here we introduce a technique that eliminates an useless action (*dominated action*) extending this idea. If there exists a dominated action a, we can eliminate a variable as $\mathcal{U}(a) = 0$. We omit the proof due to space.

Proposition 4. *Given a feasible delete-free task T^+, there exists an optimal plan that does not contains $a \in A$ if there exists an action $a' \in A$ satisfying following: (i) $\mathrm{fadd}(a) \subseteq \mathrm{fadd}(a')$, (ii) for any $p \in \mathrm{pre}(a')$, p is a fact landmark of a or $p \in I$, and (iii) $c(a) \geq c(a')$.*

Robinson proposed similar constraints for a MaxSAT-based planner, but his condition is stricter than condition (ii) (Robinson 2012).

3.4 Immediate Action Application

On a delete-free task T^+, applying some types of actions to the initial state do not hurt optimality. We adopt to use an action with cost zero as (Gefen and Brafman 2011) and an action landmark as (Gefen and Brafman 2012) to this enhancement. For a delete-free task T^+, if an action $a \in A$ satisfies

$c(a) = 0$ and $\mathrm{pre}(a) \subseteq I$, then a sequence made by connecting a before an optimal plan of $\langle P, A \setminus \{a\}, I \cup \mathrm{add}(a), G\rangle$ is an optimal plan of T^+. Similarly, if an action a is an action landmark of T^+ and a is applicable to I, you can apply a to I immediately.

For $\mathrm{IP}(T^+)$, variables $\mathcal{T}(p)$ for $p \in I$ can be eliminated by substituting zero. Given a sequence of immediate applicable actions $(a_0, \cdots, a_k)$ (it must be a correct applicable sequence), we can eliminate some variables as follows: (i) $\mathcal{U}(a_i) = 1$, (ii) $\mathcal{T}(a_i) = i$, (iii) $\forall p \in \mathrm{pre}(a_i), \mathcal{U}(p) = 1$, and (iv) $\forall p \in \mathrm{add}(a_i) \setminus I((a_0, \cdots, a_{i-1})), \mathcal{U}(p) = 1, \mathcal{T}(p) = i$ and $\mathcal{E}(a_i, p) = 1$.

3.5 Iterative Application of Variable Eliminations

The variable elimination techniques described above can interact synergistically with each other resulting in a cascade of eliminations. For example, landmarks increase non relevant add effects, which increases dominated actions, which can result in new landmarks. Therefore, we used a iterative variable eliminating algorithm which applies eliminations until quiescence.

A full landmark extraction pass after each variable elimination would be extremely expensive, but landmark extraction can be implemented incrementally. Hence we perform a complete landmark extraction once for each state, and after that, the incremental extraction is executed after each variable reduction.

3.6 Inverse action constraints

We define the following inverse relationship between a pair of actions for a delete-free task T^+. For two actions $a_1, a_2 \in A$, a_1 is an *inverse action* of a_2 if it satisfies following: (i) $\mathrm{add}(a_1) \subseteq \mathrm{pre}(a_2)$, and (ii) $\mathrm{add}(a_2) \subseteq \mathrm{pre}(a_1)$. By the definition, it is clear that if a_1 is an inverse action of a_2, then a_2 is an inverse action of a_1. Inverse actions satisfy following fact (proof omitted due to space).

Proposition 5. *For a delete-free task T^+, a feasible solution $\pi = (a_0, \cdots, a_n)$ is not optimal if $a_i \in \pi$ is an inverse action of $a_j \in \pi$ and both of a_i and a_j have non-zero cost.*

Let $\mathrm{inv}(a, p)$ denote the set of inverse actions of an action a which have p as add effect. There are several possible ways to use above proposition (e.g., $\mathcal{U}(a) + \mathcal{U}(a') \leq 1$, for all $a' \in \mathrm{inv}(a)$). On $\mathrm{IP}(T^+)$, due to avoid adding a huge number of constraints, we modify constraint 2 as follows:

2. $\forall a \in A,\ \forall p \in \mathrm{pre}(a), \mathcal{U}(p) - \sum_{a' \in \mathrm{inv}(a,p)} \mathcal{E}(a', p) \geq \mathcal{U}(a)$.

We use e (e.g. $\mathrm{LP}^{\mathrm{e}}(T^+)$) to denotes the ILP after all of the reductions in Sections 3.1-3.6 have been applied.

3.7 Constraint Relaxation

So far in this section, we have presented enhancements which seek to speed up the computation of h^+. As we show experimentally in Section 6, computing $\mathrm{IP}(T^+)$ or $\mathrm{LP}(T^+)$ remains relatively expensive, even if we use all of the enhancements described above.

Thus, we introduce a relaxation for $\mathrm{IP}(T^+)$. We call $\mathrm{IP}(T^+)$ without constraints 5 and 6 *time-relaxed* $\mathrm{IP}(T^+)$, denoted $\mathrm{IP}_{\mathrm{tr}}(T^+)$. Similarly we call $\mathrm{LP}(T^+)$ without same constraints *time-relaxed* $\mathrm{LP}(T^+)$, denoted $\mathrm{LP}_{\mathrm{tr}}(T^+)$. It can be seen that if the relevance of propositions and actions has an ordering (i.e. it does not have a cycle) on T^+, then the optimal costs of $\mathrm{IP}(T^+)$ and $\mathrm{LP}(T^+)$ are the same as the optimal costs of $\mathrm{IP}_{\mathrm{tr}}(T^+)$ and $\mathrm{LP}_{\mathrm{tr}}(T^+)$ respectively. We shall show experimentally in Section 6.1 that the relaxation is quite tight (i.e., $\mathrm{IP}(T^+)$ and $\mathrm{IP}_{\mathrm{tr}}(T^+)$ often have the same cost), and that $\mathrm{IP}_{\mathrm{tr}}(T^+)$ can be computed significantly faster than $\mathrm{IP}(T^+)$. $\mathrm{LP}(T^+), \mathrm{LP}^{\mathrm{e}}(T^+), \mathrm{IP}^{\mathrm{e}}(T^+)$ have same behavior.

4 Counting Constraints

So far, we have concentrated on efficient computation of h^+, and all of our relaxations are bounded by h^+. However, our IP model can be extended with constraints regarding delete effects. By adding variables and constraints related to delete effects of actions, our model can also calculate lower bounds on the number of times each action must be applied. New variables are defined as follows:

- $\forall a \in A, \mathcal{N}(a) \in \{0, 1, \cdots\} : \mathcal{N}(a) = n$ iff a is used n times.
- $\forall p \in P, \mathcal{G}(p) \in \{0, 1\} : \mathcal{G}(p) = 1$ iff $p \in G$.

$\mathcal{G}(p)$ is also an auxiliary variable as $\mathcal{I}(p)$. New constraints are defined as follows:

7. $\forall a \in A, \mathcal{N}(a) \geq \mathcal{U}(a)$.
8. $\forall p \in P, \mathcal{G}(p) + \sum_{p \in \mathrm{predel}(a)} \mathcal{N}(a) \leq \mathcal{I}(p) + \sum_{p \in \mathrm{add}(a)} \mathcal{N}(a)$,

where $\mathrm{predel}(a) = \mathrm{pre}(a) \cap \mathrm{del}(a)$. Finally, the objective function is modified so as to minimize $\sum_{a \in A} c(a)\mathcal{N}(a)$.

These constraints correspond to the *net change* constraints that were recently proposed in (Pommerening et al. 2014), as well as the action order relaxation in (van den Briel et al. 2007), (both are based on SAS$^+$ formulations). Intuitively, the final constraint states that the number of times actions adding p are used must be equal to or larger than the number of times actions requiring and deleting p same time are used. Given a non delete-free task T, we use $\mathrm{IP}(T)$ to denote an IP problem composed of $\mathrm{IP}(T^+)$ and above new variables and constraints. We also use LP and tr as same as corresponding relaxations for $\mathrm{IP}(T^+)$. For any T and any feasible plan π for T, there exists a feasible solution of $\mathrm{IP}(T)$ with same cost as π, since the delete relaxation of π is a feasible plan of T^+. Hence the optimal cost of naive $\mathrm{IP}(T)$ is an admissible heuristic for T.

Unfortunately these new constraints conflict with dominated action elimination and zero cost immediate action application. When counting constraint is used, it is necessary to disable zero cost immediate action applying and to modify the condition of dominated action: an action a is a dominated action of action a' if (i) $\mathrm{add}(a) \subseteq \mathrm{add}(a')$, (ii) for any $p \in \mathrm{pre}(a')$, p is a fact landmark of a or $p \in I$, (iii) $c(a) \geq c(a')$, and (iv) $\mathrm{pre}(a') \cap \mathrm{del}(a') \subseteq \mathrm{pre}(a) \cap \mathrm{del}(a)$. On the other hand, following fact ensures that other enhancements do not hurt admissibility of $\mathrm{IP}(T)$. We omit detailed discussion due to space. We also use e (e.g. $\mathrm{LP}^{\mathrm{e}}(T)$) to

denotes the ILP after all of the valid reductions have been applied.

Proposition 6. *Given a task T, let $\mathrm{IP}^{\mathrm{e}}(T^+)$ be a variable-reduced IP for T^+, and $\mathrm{IP}^{\mathrm{e}}(T)$ be an IP made from $\mathrm{IP}^{\mathrm{e}}(T^+)$ with counting constraints. For any feasible solution π of T, if there exists a solution of $\mathrm{IP}^{\mathrm{e}}(T^+)$ derived from a subsequence of π^+, then there exists a feasible solution of $\mathrm{IP}^{\mathrm{e}}(T)$ that has same cost as π.*

5 Relationship among the ILP bounds

Based on the definitions, we know that: $\mathrm{IP}_{\mathrm{tr}}(T^+) \leq \mathrm{IP}^{\mathrm{e}}_{\mathrm{tr}}(T^+) \leq \mathrm{IP}(T^+) = \mathrm{IP}^{\mathrm{e}}(T^+) \leq \mathrm{IP}(T) = \mathrm{IP}^{\mathrm{e}}(T)$. As for the LP relaxations, we know that $\mathrm{LP}(T^+) \leq \mathrm{LP}^{\mathrm{e}}(T^+)$, $\mathrm{LP}^{\mathrm{e}}_{\mathrm{tr}}(T^+) \leq \mathrm{LP}^{\mathrm{e}}(T^+)$, $\mathrm{LP}^{\mathrm{e}}_{\mathrm{tr}}(T) \leq \mathrm{LP}^{\mathrm{e}}(T)$, and $\mathrm{LP}^{\mathrm{e}}_{\mathrm{tr}}(T) \leq \mathrm{LP}^{\mathrm{e}}(T)$. However, $\mathrm{LP}^{\mathrm{e}}(T)$ does not always dominate $\mathrm{LP}^{\mathrm{e}}(T^+)$ since sets of eliminated variables are different because of dominated action elimination and zero-cost immediate action application. Figure 1 illustrates the dominance relationships among the bounds.

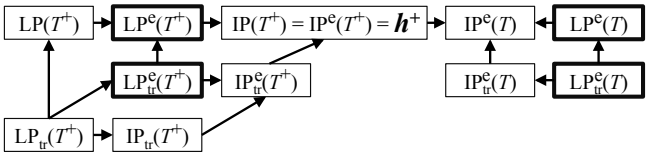


Figure 1: Dominance relationships. Edge $L_i \rightarrow L_j$ indicates "$L_i \leq L_j$". The 4 highlighted LP's are used in the A*/autoconf in Tables 2-3.

5.1 Automatic bound selection for each problem

While $\mathrm{LP}^{\mathrm{e}}_{\mathrm{tr}}(T^+)$ and $\mathrm{LP}^{\mathrm{e}}_{\mathrm{tr}}(T)$ are dominated by $\mathrm{LP}^{\mathrm{e}}(T^+)$ and $\mathrm{LP}^{\mathrm{e}}(T)$, respectively, the time-relaxed LPs are significantly cheaper to compute than their non-relaxed counterparts. Similarly, although $\mathrm{IP}^{\mathrm{e}}(T)$ dominates $\mathrm{IP}^{\mathrm{e}}(T^+)$, it is possible for $\mathrm{LP}^{\mathrm{e}}(T^+)$ to be larger than $\mathrm{LP}^{\mathrm{e}}(T)$. Thus, we have a set of 4 viable LP heuristics, none of which dominate the others when considering both accuracy and time. The "best" choice to optimize this tradeoff between heuristic accuracy and node expansion rate depends on the problem instance.

We implemented a simple mechanism for automatically selecting the LP to be used for each problem. First, we compute $\mathrm{LP}^{\mathrm{e}}(T^+)$, $\mathrm{LP}^{\mathrm{e}}(T)$, $\mathrm{LP}^{\mathrm{e}}_{\mathrm{tr}}(T^+)$, $\mathrm{LP}^{\mathrm{e}}_{\mathrm{tr}}(T)$ for the problem instance (i.e., at the root node of the A* search). We then select one based on the following rule: Choose the heuristic with the highest value. Break ties by choosing the heuristic that is cheapest to compute. Although the "cheapest" heuristic could be identified according to the cpu time to compute each heuristic, for many problems, the computations are too fast for robust timing measurements, so we simply break ties in order of $\mathrm{LP}^{\mathrm{e}}_{\mathrm{tr}}(T^+)$, $\mathrm{LP}^{\mathrm{e}}_{\mathrm{tr}}(T)$, $\mathrm{LP}^{\mathrm{e}}(T^+)$, $\mathrm{LP}^{\mathrm{e}}(T)$ (because this ordering usually accurately reflects the timing order). A more sophisticated method for heuristic selection may result in better performance (c.f. (Domshlak, Karpas, and Markovitch 2012)), and is an avenue for future work.

6 Experimental Evaluation

Below, all experiments used the CPLEX 12.6 solver to solve integer linear programs. All experiments were single-threaded and executed on a Xeon E5-2650, 2.6GHz. We used a set of 1,366 IPC benchmark problems (from 1998 to 2011) distributed with Fast Downward. Our planner can currently handle the subset of PDDL which includes STRIPS, types, and action-costs. The full list of domains and # of instances per domain is shown in Table 3.

6.1 Comparison of ILP Bounds

We evaluate the quality of the integer/linear programming bounds by evaluating the optimal costs computed for these bounds.

First, we compute the ratio between the optimal cost of the LP relaxation and the IP (Figure 2). We take the ceiling of the LP cost, because the IPC benchmarks have integer costs. As shown in Table 2, the gap between the LP and IP relaxation are quite small. In fact, for the majority of problems, the gap between the rounded-up LP value and IP value is 0 for $\mathrm{IP}^{\mathrm{e}}(T^+)$, $\mathrm{IP}^{\mathrm{e}}(T)$, $\mathrm{IP}^{\mathrm{e}}_{\mathrm{tr}}(T^+)$, $\mathrm{IP}^{\mathrm{e}}_{\mathrm{tr}}(T)$, so the LP relaxation is frequently a perfect approximation of h^+.

Next, to understand the impact of various sets of constraints in the ILP formulations, Table 1 compares pairs of IP and LP formulations. The IP ratio for $\mathrm{IP}(T^+)$ vs $\mathrm{IP}^{\mathrm{e}}(T^+)$ is always 1 because they both compute h^+. However, on almost every single domain, the LP value of the extended formulation $\mathrm{LP}^{\mathrm{e}}(T^+)$ is significantly better (higher) than the basic formulation $\mathrm{LP}(T^+)$, indicating that variable elimination and the additional constraints serve to tighten the LP bound. Thus, the enhancements to the basic model described in Section 3 provide a significant benefit. $\mathrm{LP}^{\mathrm{e}}(T)$ tends to be higher than $\mathrm{LP}^{\mathrm{e}}(T^+)$, indicating that that counting constraints enhances accuracy; note that in some cases $\mathrm{LP}^{\mathrm{e}}(T^+)$ is higher than $\mathrm{LP}^{\mathrm{e}}(T)$. The time-relaxations $\mathrm{LP}^{\mathrm{e}}_{\mathrm{tr}}(T^+)$ and $\mathrm{LP}^{\mathrm{e}}_{\mathrm{tr}}(T)$ are usually very close to $\mathrm{LP}^{\mathrm{e}}(T^+)$ and $\mathrm{LP}^{\mathrm{e}}(T)$, indicating that the time relaxation achieves a good tradeoff between computation cost and accuracy.

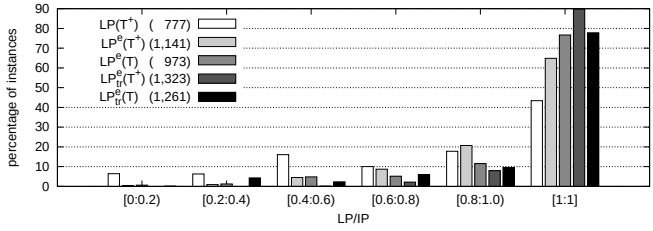


Figure 2: Ratio between the optimal costs of the IP's and their LP relaxations, categorized into buckets. [x:y) = "% of instances where the LP/IP ratio is in the range [x:y).

6.2 Evaluating ILP for Delete-free planning

To evaluate the speed of our ILP approach, we compared $\mathrm{IP}^{\mathrm{e}}(T^+)$ with Haslum et al.'s h^+ algorithm (Haslum, Slaney, and Thiébaux 2012) ("HST"), which is one of the state-of-the art solvers for the delete relaxation, of a set of 1,346 IPC benchmarks from the Fast Downward benchmark

Table 1: Comparison of bounds: $il^{+} = \mathrm{ILP}(T^{+})$, $il^{e+} = \mathrm{ILP}^{\mathrm{e}}(T^{+})$, $il^{e} = \mathrm{ILP}^{\mathrm{e}}(T)$, $il^{e+}_{tr} = \mathrm{ILP}^{\mathrm{e}}_{\mathrm{tr}}(T^{+})$, $il^{e}_{tr} = \mathrm{ILP}^{\mathrm{e}}_{\mathrm{tr}}(T)$.

	il^{+} / il^{e+}		il^{e} / il^{e+}		il^{e+}_{tr} / il^{e+}		il^{e}_{tr} / il^{e}	
	LP	IP	LP	IP	LP	IP	LP	IP
airport	.53	1.00	.99	1.00	.99	.99	1.00	.99
blocks	.92	1.00	.92	.92	1.00	1.00	1.00	1.00
depot	.54	1.00	.93	.99	.99	.92	1.00	.99
driverlog	.97	1.00	.91	.95	.96	.84	1.00	.96
elevators-opt08	.39	1.00	1.16	.96	.97	.64	1.00	.70
elevators-opt11	.36	1.00	1.17	.96	.96	.62	1.00	.73
floortile-opt11	.99	1.00	.93	.94	1.00	.97	1.00	.98
freecell	.48	1.00	1.01	1.00	.97	.92	1.00	.98
grid	-	-	.79	.85	.98	.79	1.00	.88
gripper	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
logistics98	.54	1.00	.89	1.00	.98	.88	1.00	1.00
logistics00	.47	1.00	.99	1.00	.99	.99	1.00	1.00
miconic	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
movie	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
no-mprime	.58	1.00	1.10	.97	.88	.66	1.00	.94
no-mystery	.58	1.00	1.03	.98	.92	.72	1.00	.96
nomystery-opt11	.97	1.00	.97	.97	1.00	1.00	1.00	1.00
openstacks	.38	1.00	1.00	1.00	1.00	1.00	1.00	1.00
openstacks-opt08	0	1.00	1.00	1.00	1.00	1.00	1.00	1.00
openstacks-opt11	-	-	1.00	1.00	1.00	1.00	1.00	1.00
parcprinter-08	.99	1.00	.92	.92	1.00	1.00	1.00	1.00
parcprinter-opt11	.99	1.00	.94	.94	1.00	1.00	1.00	1.00
parking-opt11	.90	1.00	.97	.97	.94	.87	.94	.86
pegsol-08	0	1.00	.81	.72	1.00	.68	1.00	.86
pegsol-opt11	0	1.00	.88	.73	1.00	.67	1.00	.86
pipes-notankage	.62	1.00	.94	.95	.92	.83	.97	.90
pipes-tankage	.62	1.00	.95	.96	.98	.87	1.00	.96
psr-small	.87	1.00	.38	.38	1.00	1.00	1.00	1.00
rovers	.63	1.00	.86	.77	1.00	1.00	1.00	1.00
satellite	.99	1.00	.99	.99	1.00	1.00	1.00	1.00
scanalyzer-08	1.00	1.00	1.00	1.00	1.00	.96	1.00	1.00
scanalyzer-opt11	1.00	1.00	1.00	1.00	1.00	.96	1.00	1.00
sokoban-opt08	.37	1.00	.88	.87	.99	.95	.99	.94
sokoban-opt11	.34	1.00	.90	.88	.99	.97	1.00	.96
storage	.55	1.00	.95	.91	1.00	1.00	1.00	1.00
transport-opt08	.26	1.00	3.42	1.00	.99	.36	1.00	.58
transport-opt11	-	-	-	-	.99	.43	-	-
visitall-opt11	1.00	1.00	.95	.93	.99	.97	.99	.95
woodworking08	.81	1.00	.94	.94	1.00	1.00	1.00	1.00
woodworking11	.80	1.00	.94	.94	1.00	1.00	1.00	1.00
zenotravel	.99	1.00	.92	.98	.96	.90	1.00	.99

suite. Both solvers were run with a 15-minute time limit on each instance. The most recent version of HST was configured to use CPLEX to solve the hitting set subproblem, as suggested by Haslum (Haslum 2014).

The number of delete-free, relaxed instances that are solved by both planner is 905. HST solved 1,117 instances, and $\mathrm{IP}^{\mathrm{e}}(T^+)$ solved 1,186 instances. $\mathrm{IP}^{\mathrm{e}}(T^+)$ was faster than HST on 575 instances, and HST was faster than $\mathrm{IP}^{\mathrm{e}}(T^+)$ on 330 instances. Figure 3 shows the ratio of runtimes of HST to our solver, sorted in increasing order of the ratio, time(HST's h^+)/time($\mathrm{IP}^{\mathrm{e}}(T^+)$). The horizontal axis is the cumulative number of instances. Overall, $\mathrm{IP}^{\mathrm{e}}(T^+)$ outperform the state-of-the-art delete-free solver and indicates that direct computation of h^+ using integer programming is a viable approach (at least for computing h^+ once for each problem).

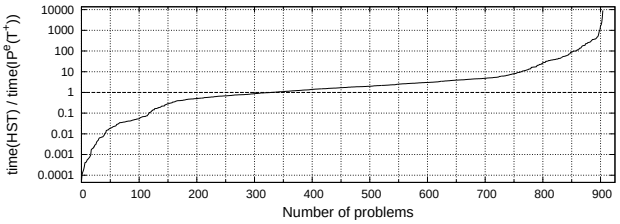


Figure 3: Computation of h^+: Comparison of $\mathrm{IP}^{\mathrm{e}}(T^+)$ and HST on delete-free, relaxed problems

6.3 Evaluating h^+-based heuristics in a cost-optimal planner

We embedded the ILP model into a A*-based, cost-optimal forward search planner. We first compared various configurations of our planner, as well as several configurations of Fast Downward (FD), given 5 minutes per problem instance and a 2GB memory limit. For the FD bisimulation merge-and-shrink heuristic, we use the IPC2011 hybrid bisimulation m&s configuration (`seq-opt-merge-and-shrink`).[1] The # of problems solved by each configuration is shown in Table 2.

As shown in Table 2, the basic IP model performs the worst, and is comparable to A*/h^+. As noted in (Haslum 2012), straightforward use of h^+ as a heuristic is unsuccessful (significantly worse than FD using $h^{\max}$). However, the addition of landmark constraints is sufficient to significantly increase the number of solved problems compared to A*/h^+, and A*/$\mathrm{IP}^{\mathrm{e}}(T^+)$, outperforms $h^{\max}$ and can be considered a somewhat useful heuristic. The time-relaxation results in significantly increases performance compared to A*/$\mathrm{IP}^{\mathrm{e}}(T^+)$ and A*/$\mathrm{IP}^{\mathrm{e}}(T)$. In addition, for all IP models, A* search using their corresponding LP relaxations as the heuristic function performs significantly better than directly using the IP as the A* heuristic. A*/$\mathrm{LP}^{\mathrm{e}}(T^+)$, A*/$\mathrm{LP}^{\mathrm{e}}_{\mathrm{tr}}(T^+)$, and A*/$\mathrm{LP}^{\mathrm{e}}_{\mathrm{tr}}(T)$, are all competitive with the bisimulation merge-and-shrink heuristic. While A*/$\mathrm{LP}^{\mathrm{e}}(T)$, does not perform quite as well, there are some problems where A*/$\mathrm{LP}^{\mathrm{e}}(T)$ performs best. Finally, A*/autoconf, which uses LP heuristic selection (Section 5.1) performs quite well, significantly better than its 4 components ($\mathrm{LP}^{\mathrm{e}}(T^+)$, $\mathrm{LP}^{\mathrm{e}}_{\mathrm{tr}}(T^+)$, $\mathrm{LP}^{\mathrm{e}}_{\mathrm{tr}}(T)$, $\mathrm{LP}^{\mathrm{e}}(T)$).

Table 3 compares the coverage following algorithms on the IPC benchmark suite with 30 minute CPU time limit and 2GB memory limit: (1) A*/autoconf, which uses the LP heuristic selection mechanism described in Section 5.1 to choose among $\mathrm{LP}^{\mathrm{e}}(T^+)$, $\mathrm{LP}^{\mathrm{e}}(T)$, $\mathrm{LP}^{\mathrm{e}}_{\mathrm{tr}}(T^+)$, $\mathrm{LP}^{\mathrm{e}}_{\mathrm{tr}}(T)$, (2) FD using the Landmark Cut heuristic (Helmert and Domshlak 2009), and (3) FD using the IPC2011 bisimulation merge-and-shrink configuration (`seq-opt-merge-and-shrink`)(Nissim, Hoffmann, and Helmert 2011).

Our results indicate that A*/autoconf is competitive with both Fast Downward using Landmark Cut, as well as the IPC2011 Merge-and-shrink portfolio configuration. None of these planners dominate the others, and each planner performs the best on some subset of domains. Compared to the two other methods, A*/autoconf seems to perform particularly well on the freecell, parcprinter, rovers, trucks, and woodworking domains. A*/h^+(Haslum, Slaney, and Thiébaux 2012) solved 443 problems with a 30-minute time limit, which is significantly less coverage than than our LP-based planners with a 5-minute time limit (Table 2).

As described in Section 5.1, A*/autoconf selects the LP heuristic to use for each problem based on a comparison of LP values at the root node. $\mathrm{LP}^{\mathrm{e}}_{\mathrm{tr}}(T^+)$ was selected on 755 problems, $\mathrm{LP}^{\mathrm{e}}_{\mathrm{tr}}(T)$ on 447 problems, $\mathrm{LP}^{\mathrm{e}}(T^+)$ on 119 problems, and $\mathrm{LP}^{\mathrm{e}}(T)$ on 25 problems. On the remaining 20 problems, A*/autoconf timed out during LP computations for the bound selection process at the root node, indicating that for some difficult problems, the LP computation can be prohibitively expensive.

7 Conclusion

This paper proposed a new, integer-linear programming formulation of the delete relaxation h^+ for cost-optimal, domain-independent planning. The major contribution of this paper are: (1) We propose an enhanced IP model for h^+ using landmarks, relevance analysis, and action elimination, which is outperforms one of the previous state-of-the-art techniques for computing h^+ (Haslum, Slaney, and Thiébaux 2012); (2) We showed that the LP relaxations of the IP models are quite tight; and (3) We embedded our relaxed LPs in a A*-based forward search planner, A*/autoconf. We showed that A* search using $\mathrm{LP}^{\mathrm{e}}(T^+)$, $\mathrm{LP}^{\mathrm{e}}_{\mathrm{tr}}(T^+)$, or $\mathrm{LP}^{\mathrm{e}}_{\mathrm{tr}}(T)$ as its heuristic is competitive with the hybrid bisimulation merge-and-shrink heuristic (Nissim, Hoffmann, and Helmert 2011). Using a simple rule to select from among $\mathrm{LP}^{\mathrm{e}}(T^+)$, $\mathrm{LP}^{\mathrm{e}}(T)$, and $\mathrm{LP}^{\mathrm{e}}_{\mathrm{tr}}(T^+)$, $\mathrm{LP}^{\mathrm{e}}_{\mathrm{tr}}(T)$, A*/autoconf is competitive with the landmark cut heuristic. A*/autoconf performs well in some domains where other planners perform poorly, so our ILP-based methods are complementary to previous heuristics.

While it has long been believed that h^+ is too expensive to be useful as a heuristic for forward-search based planning, our work demonstrates that an LP relaxation of h^+

[1] While this is tuned for 30 minutes and suboptimal for 5 minutes, we wanted to use the same configuration as in the 30-minute experiments below.

Table 2: IPC benchmark problems: # solved with 5 minute time limit.

Configuration	# solved	Description
FD/LM-cut	746	Landmark Cut (`seq-opt-lmcut`)
FD/M&S IPC2011	687	IPC 2011 Merge-and-Shrink (Nissim, Hoffmann, and Helmert 2011)
FD/$h^{\max}$	551	$h^{\max}$
A*/h^+	342	hsp_f planner using A* and h^+ heuristic (Haslum, Slaney, and Thiébaux 2012; Haslum 2012)
A*/IP(T^+)	358	basic IP formulation for h^+
A*/LP(T^+)	477	LP relaxation of IP(T^+)
A*/IP(T^+)+land	425	IP(T^+) + Landmarks
A*/LP(T^+)+land	564	LP relaxation of IP(T^+)
A*/IP$^{\mathrm{e}}$(T^+)	582	IP(T^+) with all enhancements in Sections 3.1-3.6
A*/LP$^{\mathrm{e}}$(T^+)	652	LP relaxation of IP$^{\mathrm{e}}$(T^+)
A*/IP$^{\mathrm{e}}$(T)	463	IP$^{\mathrm{e}}$(T^+) with counting constraints (Section 4)
A*/LP$^{\mathrm{e}}$(T)	608	LP relaxation of IP$^{\mathrm{e}}$(T)
A*/IP$^{\mathrm{e}}_{\mathrm{tr}}$($T^+$)	606	time-relaxation (Section 3.7) of IP$^{\mathrm{e}}$(T^+)
A*/LP$^{\mathrm{e}}_{\mathrm{tr}}$($T^+$)	674	LP relaxation of IP$^{\mathrm{e}}_{\mathrm{tr}}$($T^+$)
A*/IP$^{\mathrm{e}}_{\mathrm{tr}}$($T$)	554	time-relaxation of IP$^{\mathrm{e}}$(T)
A*/LP$^{\mathrm{e}}_{\mathrm{tr}}$($T$)	661	LP relaxation of IP$^{\mathrm{e}}_{\mathrm{tr}}$($T$)
A*/autoconf	722	Automated selection of LP at root node(Section 5.1)

can achieve the right tradeoff of speed and accuracy to be the basis of a new class of heuristics for domain-independent planning. Integrating additional constraints to derive heuristics more accurate than h^+ (e.g., the inclusion of net change constraints (Pommerening et al. 2014) in Section 4) offers many directions for future work.

Acknowledgments

Thanks to Patrik Haslum for assistance with his code for computing h^+ and his hsp_f planner. This research was supported by a JSPS Grant-in-Aid for JSPS Fellows and a JSPS KAKENHI grant.

References

Betz, C., and Helmert, M. 2009. Planning with h^+ in theory and practice. In *KI 2009*. Springer. 9–16.

Bonet, B. 2013. An admissible heuristic for SAS+ planning obtained from the state equation. In *Proc. IJCAI*, 2268–2274.

Bylander, T. 1994. The Computational Complexity of Propositional STRIPS Planning. *Artificial Intelligence* 69(1–2):165–204.

Bylander, T. 1997. A linear programming heuristic for optimal planning. In *AAAI/IAAI*, 694–699. Citeseer.

Domshlak, C.; Karpas, E.; and Markovitch, S. 2012. Online speedup learning for optimal planning. *JAIR* 44:709–755.

Gefen, A., and Brafman, R. 2011. The minimal seed set problem. In *ICAPS*, 319–322.

Gefen, A., and Brafman, R. 2012. Pruning methods for optimal delete-free planning. In *ICAPS*, 56–64.

Haslum, P.; Slaney, J.; and Thiébaux, S. 2012. Minimal landmarks for optimal delete-free planning. In *ICAPS*, 353–357.

Haslum, P. 2012. Incremental lower bounds for additive cost planning problems. In *ICAPS*, 74–82.

Haslum, P. 2014. Personal communication.

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

Imai, T., and Fukunaga, A. 2014. A practical, integer-linear programming model for the delete-relaxation in cost-optimal planning. In *Proceedings of European Conference on Artificial Intelligence (ECAI)*.

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

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 *IJCAI*, 1983–1990.

Pommerening, F., and Helmert, M. 2012. Optimal planning for delete-free tasks with incremental LM-cut. In *ICAPS*, 363–367.

Pommerening, F.; Röger, G.; Helmert, M.; and Bonet, B. 2014. LP-based heuristics for cost-optimal planning. In *ICAPS*.

Robinson, N. 2012. *Advancing Planning-as-Satisfiability*. Ph.D. Dissertation, Griffith University.

van den Briel, M., and Kambhampati, S. 2005. Optiplan: A planner based on integer programming. *JAIR* 24:919–931.

van den Briel, M.; Benton, J.; Kambhampati, S.; and Vossen, T. 2007. An LP-based heuristic for optimal planning. In *Proc. CP-2007*.

van den Briel, M.; Vossen, T.; and Kambhampati, S. 2008. Loosely coupled formulation for automated planning: An integer programming perspective. *JAIR* 31:217–257.

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

Table 3: 30 minutes, 2GB RAM: "evals"=# of calls to heuristic function

	Fast Downward LM-Cut		Fast Downward M&S		A*/autoconf	
Domain (# problems)	solved	evals	solved	evals	solved	evals
airport(50)	**28**	13403	23	461855	25	4640
barman-opt11(20)	**4**	1614605	**4**	5944586	3	473561
blocks(35)	28	95630	28	880799	**29**	51523
depot(22)	**7**	261573	**7**	1746549	**7**	34046
driverlog(20)	**14**	245920	13	4355507	13	56933
elevators-opt08(30)	**22**	1189951	14	10132421	13	66011
elevators-opt11(20)	**18**	1196979	12	11811143	10	65695
floortile-opt11(20)	**7**	2354266	**7**	10771362	**7**	152836
freecell(80)	15	180560	19	6291413	**45**	2177
grid(5)	2	94701	**3**	11667600	**3**	14197
gripper(20)	7	1788827	**20**	3131130	6	404857
logistics98(35)	6	169645	5	6825245	**7**	143897
logistics00(28)	**20**	212998	**20**	3007288	**20**	212985
miconic(150)	**141**	16635	77	3872365	**141**	15087
movie(30)	**30**	29	**30**	29	**30**	31
no-mprime(35)	**24**	55549	22	1490714	18	7260
no-mystery(30)	16	880031	**17**	3725239	12	1105
nomystery-opt11(20)	14	20744	**19**	9951860	14	754
openstacks(30)	**7**	157100	**7**	202732	**7**	4973
openstacks-opt08(30)	19	3254361	**21**	6347048	11	165070
openstacks-opt11(20)	14	4412937	**16**	8326670	6	294006
parcprinter-08(30)	19	699592	17	3129238	**29**	668
parcprinter-opt11(20)	14	949416	13	4091925	**20**	854
parking-opt11(20)	3	435359	**7**	8044843	1	2991
pegsol-08(30)	27	224149	**29**	705639	26	85760
pegsol-opt11(20)	17	370401	**19**	1092529	16	151110
pipes-notankage(50)	**17**	234717	**17**	1777823	13	6021
pipes-tankage(50)	12	361767	**16**	2447552	7	1926
psr-small(50)	49	178328	**50**	221152	**50**	4056
rovers(40)	7	77783	8	3395947	**11**	209551
satellite(36)	7	155990	7	1890912	**10**	26897
scanalyzer-08(30)	**15**	259961	14	6785907	8	4374
scanalyzer-opt11(20)	**12**	324943	11	8636568	5	6975
sokoban-opt08(30)	**30**	669669	24	3938226	23	75743
sokoban-opt11(20)	**20**	173004	19	3338708	19	77681
storage(20)	**15**	86439	**15**	1006600	**15**	21598
transport-opt08(30)	**11**	16807	**11**	1158282	10	58616
transport-opt11(20)	6	30550	**7**	4473292	5	116375
trucks(30)	10	462320	8	8478357	**15**	61067
visitall-opt11(20)	11	1255455	16	129229	**17**	20378
woodworking08(30)	17	759825	14	876479	**28**	767
woodworking11(20)	12	1076372	9	1357935	**18**	699
zenotravel(20)	**13**	318142	12	6727643	12	16571
Total (1366)	787		727		785	

Optimal Planning in the Presence of Conditional Effects: Extending LM-Cut with Context Splitting

Gabriele Röger and **Florian Pommerening** and **Malte Helmert**
University of Basel, Switzerland

Abstract

The LM-Cut heuristic is currently the most successful heuristic in optimal STRIPS planning but it cannot be applied in the presence of conditional effects. Keyder, Hoffmann and Haslum recently showed that the obvious extensions to such effects ruin the nice theoretical properties of LM-Cut. We propose a new method based on context splitting that preserves these properties.

A revised version of this paper has been accepted at ECAI (Röger, Pommerening, and Helmert 2014).

Introduction

The aim of classical planning is to find a sequence of actions that leads from the current state of the world to some desired state. Conditional effects enable situation-dependent behavior of actions. For example, there can be a single action *stop-*f in an elevator domain that boards waiting passengers at floor f and disembarks all boarded passengers with destination f. To describe such a behavior without conditional effects, one would instead need specific actions for all different situations of waiting and boarded passengers related to this floor, or use some other formulation that applies several actions to cause the same world change.

Conditional effects can be compiled away (Nebel 2000) but such transformations have severe disadvantages: any plan-preserving transformation leads to an exponential blow-up of the size of the task description. An alternative compact compilation does not preserve the delete relaxation, which many heuristics such as the LM-Cut heuristic (Helmert and Domshlak 2009) are based on. As a result, these heuristics do not give good guidance on such compiled tasks.

Haslum (2013) uses an incremental compilation approach for solving delete-relaxed tasks optimally: starting from the compact compilation (which can cause further relaxation), it successively introduces the exponential transformation until an optimal solution for the compiled task can be transformed into a plan for the original task. In the worst case, this can lead to the full exponential compilation.

We take the different approach of supporting conditional effects natively in the heuristic computation. This is not unusual for inadmissible heuristics but among current *admissible* heuristics (which are required for cost-optimal planning) the support is rather weak and a suitable extension to conditional effects is not always obvious.

For the state-of-the-art LM-Cut heuristic (Helmert and Domshlak 2009), Keyder et al. (2012) recently pointed out that obvious extensions either render the heuristic inadmissible or lose the dominance over the maximum heuristic (Bonet and Geffner 2001).

We present an extension of the LM-Cut heuristic that preserves both admissibility and dominance over the maximum heuristic. For this purpose we introduce *context splitting* as a new general technique which allows us to split up actions in a tasks to distinguish different scenarios of their application. We show how context splitting can be made useful for the extension of the LM-Cut heuristic. After proving the desired theoretical properties of the heuristic, we also evaluate its performance empirically.

Background

We consider propositional STRIPS planning with action costs, extended with conditional effects. In this formalism, which we denote as STRIPS^c, a task is given as a tuple $\Pi = \langle F, A, I, G, \mathit{cost}\rangle$ where F is a set of propositional variables (or *facts*), A is a set of actions, $I \subseteq F$ is the initial state, $G \subseteq F$ describes the goal, and the cost function $\mathit{cost} : A \to \mathbb{N}_0$ defines the cost of each action. A *state* $s \subseteq F$ of a task is given by the variables that are true in this state. Every action $a \in A$ is given as a pair $a = \langle \mathit{pre}(a), \mathit{eff}(a)\rangle$. The precondition $\mathit{pre}(a) \subseteq F$ defines when the action is applicable. The set of effects $\mathit{eff}(a)$ consists of conditional effects e, each given by a triple $\langle \mathit{cond}(e), \mathit{add}(e), \mathit{del}(e)\rangle$ where all components are (possibly empty) subsets of F. If all facts in the *effect condition* $\mathit{cond}(e)$ are true in the current state, the successor state is determined by removing all facts in the *delete effect* $\mathit{del}(e)$ and adding the facts in the *add effect* $\mathit{add}(e)$. Given an effect $e \in \mathit{eff}(a)$, we use the notation $\mathit{act}(e)$ to refer to the action a.

Action a is applicable in state s if $\mathit{pre}(a) \subseteq s$. The resulting successor state is

$$s[a] = \Big(s \setminus \bigcup_{\substack{e \in \mathit{eff}(a) \text{ with} \\ \mathit{cond}(e) \subseteq s}} \mathit{del}(e)\Big) \cup \bigcup_{\substack{e \in \mathit{eff}(a) \text{ with} \\ \mathit{cond}(e) \subseteq s}} \mathit{add}(e)$$

A plan for a state s is a sequence of actions whose sequen-

tial application leads from s to a state s^* such that $G \subseteq s^*$. A plan for the task is a plan for I. The cost of a plan is the sum of the action costs as given by *cost*, and an *optimal* plan is one with minimal cost. We denote the cost of an optimal plan for s in task Π with $h^*_\Pi(s)$.

A task where all effect conditions are empty is a standard STRIPS task (with action costs). In this case, we can combine all add (and delete) effects of an action a to a single set *add*(a) (and *del*(a)).

When introducing context splitting, we will briefly consider the more general ADL formalism, where action preconditions and effect conditions are arbitrary propositional formulas over the task variables F. For a formal semantics, we need to regard a state $s \subseteq F$ as a truth assignment $T(s)$ that assigns 1 to the variables in s and 0 to all other variables. An action a is then applicable in a state s if $T(s) \models$ *pre*(a) and an effect e triggers if $T(s) \models$ *cond*(e). If not explicitly mentioned otherwise, we are talking about STRIPS^c tasks.

The delete relaxation Π^+ of a planning task Π is equivalent to Π except that all delete effects are replaced with the empty set. We call such a task *delete-free*. The cost of an optimal plan for a state s in Π^+ is denoted with $h^+(s)$ and is an admissible estimate for $h^*_\Pi(s)$ in Π. To simplify the notation throughout this paper, we avoid making the state s explicit in all definitions. Instead, we compute heuristic estimates for a state s from a modified task Π_s where we replace the initial state with s. The heuristic estimate $h(s)$ then only depends on the task Π_s and we can write $h(\Pi_s)$ instead.

Since computing h^+ is NP-complete (Bylander 1994), it is often approximated by polynomial-time computable heuristics. One such heuristic, which is dominated by h^+ and therefore also admissible, is the *maximum heuristic* $h^{\max}$ (Bonet and Geffner 2001). It assigns a value $V^{\max}$ to variables and sets of variables. The value $V^{\max}(P)$ of a non-empty set of variables $P \subseteq F$ is the maximal value of any of its elements: $V^{\max}(P) = \max_{p \in P} V^{\max}(p)$. For the empty set, $V^{\max}(\emptyset) = 0$. The value $V^{\max}(p)$ of a variable p is 0 if p is true in the initial state. Otherwise, it is the lowest estimated cost $C^{\max}(e)$ of any effect e that achieves (adds) it: $V^{\max}(p) = \min_{\{e \mid p \in \mathit{add}(e)\}} C^{\max}(e)$.[1]

The cost $C^{\max}(e)$ of an effect e is the action cost plus the value $V^{\max}$ of all propositions that must be true for the effect to trigger: $C^{\max}(e) = \mathit{cost}(\mathit{act}(e)) + V^{\max}(\mathit{cond}(e) \cup \mathit{pre}(\mathit{act}(e)))$

The estimate of the maximum heuristic for the initial state is the value $V^{\max}$ of the goal: $h^{\max}(\Pi) = V^{\max}(G)$.

Another admissible heuristic which is also based on delete relaxation and dominates $h^{\max}$ is the *LM-Cut heuristic* $h^{\text{LM-Cut}}$ (Helmert and Domshlak 2009). It relies on *disjunctive action landmarks* which are sets of actions of which at least one must occur in every plan. The LM-Cut heuristic is only defined for STRIPS tasks (without conditional effects).

To simplify the presentation, we assume in the following that the initial state consists of a single variable i and the goal of a single variable g. If the task does not have this form, we would introduce i and g as new variables and add a goal action (having the original goal as precondition and adding g) and an init action (requiring i, deleting i, and adding all variables from the original initial state), both with cost 0. We also require that every action has a precondition (if it is originally empty, we can add an artificial precondition).

[1] Strictly speaking, $V^{\max}$ is not well-defined in the presence of 0-cost actions. In this case, $V^{\max}$ is the pointwise maximal function that satisfies the given properties. A unique maximum always exists.

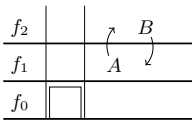


Figure 1: Running example.

The $h^{\text{LM-Cut}}$ computation works in rounds: based on the values $V^{\max}$, each round computes a disjunctive action landmark, accounts for its cost and adapts the task so that the result will be admissible:

Definition 1 (Round of LM-Cut for STRIPS) *Each round of the LM-Cut algorithm for STRIPS works as follows:*

1. *Compute $V^{\max}$ for all variables. If $V^{\max}(g) = 0$ then terminate.*
2. *Define a* precondition choice function *pcf that maps each action to one of its precondition variables with a maximal $V^{\max}$ value.*
3. *Create the weighted, directed graph $G = (V, E)$, where $V = F$ and E contains labeled edges for all actions a from the selected precondition to each add effect: $E = \{(\mathit{pcf}(a), a, v) \mid a \in A, v \in \mathit{add}(a)\}$. Each edge has weight cost(a).*
   The goal zone *$V_g \subseteq V$ consists of all nodes from which one can reach the goal variable g via edges with weight 0.*
   The cut *C contains all edges (v, a, v') such that $v \notin V_g$, $v' \in V_g$ and v can be reached from i without traversing a node in V_g.*
   The landmark L consists of all actions that occur as a label in C.
4. *Add the cost c_{min} of the cheapest action in L to the heuristic value (which starts as 0).*
5. *Reduce the action costs of all actions in L by c_{min}.*

Helmert and Domshlak (2009) call the graph G a *justification graph* of the current task because by the definition of the precondition choice function and its construction, the $h^{\max}$ value of a fact p is the cost of a cheapest (with respect to the edge weights) path from i to p. This is relevant for the proof that $h^{\text{LM-Cut}}$ dominates $h^{\max}$, so we will retain this property in our adaption to conditional effects.

Running Example

Throughout the paper we use a running example (Figure 1), borrowed from Haslum (2013, Example 1). It is based on a delete-free variant of the Miconic domain,[2] where pas-

[2] Compared to the domain reported in our experiments, there are no *move* actions and the stop action is delete-free, to get a simpler example.

sengers are transported between floors by an elevator. In this small example there are three floors (f_0, f_1, f_2) and two passengers (A and B). Passenger A wants to go from f_1 to f_2 and passenger B from f_2 to f_1. The elevator starts at f_0. The possible actions are to stop at any floor f which causes all passengers who start at f to board and all boarded passengers with target f to disembark. This is implemented by conditional effects: Each action *stop-*f has a conditional effect $\mathit{board}(p) = \langle \emptyset, \{\mathit{boarded}(p)\}, \emptyset \rangle$ for each person p originated at f. The effect condition can stay empty because in the delete-relaxed variant it is irrelevant whether we "re-board" a passenger who has already been served. For each person who has f as destination floor, the *stop-*f action has a conditional effect $\mathit{disembark}(p) = \langle \{\mathit{boarded}(p)\}, \{\mathit{served}(p)\}, \emptyset \rangle$ that marks p as served if she was in the cabin. Both such actions, *stop-*f_1 and *stop-*f_2, have no preconditions and a cost of 1.

An optimal plan for the example is $\langle \mathit{stop}\text{-}f_1, \mathit{stop}\text{-}f_2, \mathit{stop}\text{-}f_1 \rangle$. At least one *stop* action must be used twice because the first application of such an action can only trigger the effect causing the passenger to board and not the one causing the other passenger to disembark.

LM-Cut for Conditional Effects

We will now introduce a generic adaption of the LM-Cut algorithm to STRIPS^c tasks. As above, we assume that the input task has a single initial variable i and a single goal atom g. Moreover, we require without loss of generality that every conditional effect in the task only adds a single variable. If this is not the case, we can simply break up the conditional effect accordingly.

Since we still want to compute a justification graph in every round of the computation, we need to consider the effect conditions in the (pre-) condition choice function. It is also necessary that the cut in the graph distinguishes different conditional effects of an action.

Definition 2 (Generic Round of LM-Cut for STRIPS^c) *Each round of the LM-Cut algorithm for* STRIPS^c *works as follows:*

1. *Compute the $V^{\max}$ values for all variables. If $V^{\max}(g) = 0$ then terminate.*
2. *Define a* condition choice function *ccf that maps each effect to a fact from the effect condition or its action's precondition that has a maximal $V^{\max}$ value.*
3. *Create the justification graph $G = (V, E)$, where $V = F$ and E contains edges for all conditional effects e from the selected condition to the single add effect of e (labeled with e). Each edge has weight $\mathit{cost}(\mathit{act}(e))$. The goal zone V_g and the cut C are defined as in the standard STRIPS case. The landmark L consist of all actions of which an effect occurs as a label in C.*
4. *Add the cost c_{min} of the cheapest action in L to the heuristic value (which starts as 0).*
5. *Adapt the task.*

In our example, the generic LM-Cut algorithm would calculate a $V^{\max}$ value of 1 for each $\mathit{boarded}(p)$ fact and a $V^{\max}$ value of 2 for each $\mathit{served}(p)$ fact and the artificial goal fact g. The condition choice function would select one $\mathit{served}(p)$ fact arbitrarily. Let us assume it selects $\mathit{served}(A)$. The resulting justification graph is show in Figure 2a (the continuation in Figures 2b and 2c belongs to a later example). The only effect achieving $\mathit{served}(A)$ is $\mathit{disembark}(A)$, which will be the only effect in the cut. It belongs to the action *stop-*f_2, so we have $L = \{\mathit{stop}\text{-}f_2\}$ and $c_{min} = 1$.

The open question here is how to adapt the task. The most obvious way would be to apply the same strategy as in the STRIPS case and to reduce the costs of all actions in L. We denote this instantiation of the algorithm by $h^{\text{LM-Cut}}_{\text{basic}}$.

With this strategy *stop-*f_2 is free of cost after the first round in our example. In the second round the $V^{\max}$ value of both $\mathit{served}(p)$ facts is 1 and one is selected arbitrarily by the condition choice function. The discovered landmark is either $\{\mathit{board}(A)\}$ or $\{\mathit{disembark}(B)\}$ depending on this choice, but in both cases the cost of *stop-*f_1 is reduced next. After this round both *stop* actions are free of cost, the $V^{\max}$ value of the goal becomes 0, and the LM-Cut algorithm terminates with a heuristic value of 2. In this example, the $h^{\text{LM-Cut}}_{\text{basic}}$ estimate is still as high as $V^{\max}(g)$ but this is not guaranteed in general. Keyder et al. (2012) showed that $h^{\text{LM-Cut}}_{\text{basic}}$ does not dominate $h^{\max}$ with an example task Π for which $h^{\text{LM-Cut}}_{\text{basic}}(\Pi) < h^{\max}(\Pi)$.

They also considered a strategy where each conditional effect is treated separately and showed that this leads to an inadmissible heuristic. With this strategy LM-Cut would run for 4 rounds in our example. It discovers the landmarks $\{\mathit{disembark}(A)\}$, $\{\mathit{disembark}(B)\}$, $\{\mathit{board}(A)\}$, and $\{\mathit{board}(B)\}$ in an order that depends on the condition choice function. The heuristic value of 4 is inadmissible because increasing the heuristic value by 1 for each of these landmarks ignores the fact that two effects can be achieved with one action application. For example, $\mathit{board}(B)$ and $\mathit{disembark}(A)$ can be achieved by *stop-*f_2 if *stop-*f_1 was executed before.

In the following sections, we will show how one can adapt the task without sacrificing either admissibility or dominance over $h^{\max}$.

Context Splitting

Before we present the adaption specifically for the LM-Cut heuristic, we would like to introduce *context splitting* as a new general concept. For this, we briefly consider the more general ADL formalism.

Actions behave differently if they are applied in different scenarios (e. g., a conditional effect triggers only if the effect condition is true). The core idea of context splitting is that we can include such scenarios in the action preconditions, splitting up an action into several ones with disjoint scenarios. An extreme case of this general idea is the compilation from STRIPS^c to STRIPS by Nebel (2000). For each action, it introduces new actions for each possible subset of effects and adds a corresponding condition to the action precondition.

However, such scenario information can also be useful for heuristic computations: if we account for an action application in a heuristic computation, we often know that some desired effects only trigger in a certain scenario. If the action has other required effects that do not trigger at the same time, we could account for its cost again for a later application of the action.

In general, a context split is defined by the description of a scenario. Such a description is given as a propositional formula over the task variables, which we call the *context*. If we split an action with a context, we introduce two new actions, one requiring the context to be true, the other one requiring it to be false.

Definition 3 (Context splitting) *A* context *is a propositional formula.* Context-splitting *an action a with context φ means replacing a with two new actions of the same cost: $a_\varphi = \langle \mathit{pre}(a) \wedge \varphi, \mathit{eff}(a) \rangle$ and $a_{\neg\varphi} = \langle \mathit{pre}(a) \wedge \neg\varphi, \mathit{eff}(a) \rangle$.*

Context splitting is a task transformation that does not affect the optimal goal distance of any state:

Theorem 0.1 *Let Π be an* ADL *planning task with action set A. For action $a \in A$ and context φ, let a_φ and $a_{\neg\varphi}$ be the two new actions resulting from context-splitting a with φ. Let Π' denote the task that only differs from Π in its action set $A' = (A \setminus \{a\}) \cup \{a_\varphi, a_{\neg\varphi}\}$.*

For all states s of Π (and Π') it holds that $h^*_\Pi(s) = h^*_{\Pi'}(s)$.

Proof: We can associate every plan π for s in Π with a plan π' for s in Π' of the same cost and vice versa.

From π' to π, we simply replace every occurrence of an action a_φ or $a_{\neg\varphi}$ with the original action a. This is possible because these actions only differ in the precondition and $\mathit{pre}(a_\varphi) \models \mathit{pre}(a)$ and $\mathit{pre}(a_{\neg\varphi}) \models \mathit{pre}(a)$.

From π to π' we check for every occurrence of a if φ is true in the state s' in which action a is applied. If yes, we replace a with a_φ, otherwise we replace it with $a_{\neg\varphi}$. These actions will be applicable and have the same effect and cost as the original action a. ∎

The theorem ensures that an admissible heuristic estimate for the transformed task is also an admissible estimate for the original task.

Relaxed Context Splitting

The key idea of our adaption of the LM-Cut heuristic is to reduce action costs only where necessary. After discovering the landmark $\{\mathit{disembark}(A)\}$ in our example we would like to reduce the cost of *stop*-f_2 whenever it is used in a way that this effect triggers. If we stick to the original actions, however, we can only reduce the cost of the whole action, i. e., also in situations where the effect does not trigger because A has not boarded yet. Another way of looking at this is that we can reduce the cost of the original actions at most twice before all actions are free of cost, so the heuristic value can be at most 2 when no actions are modified. This is where context splitting comes into play.

The context for each action should capture all situations in which the LM-Cut heuristic accounts for its cost. This is the case whenever one of its effects occurs as a label in the cut C. So we need to formulate a context that covers all situations in which one of the effects in the cut triggers. This leads to the natural definition of the context as

$$\varphi_a = \bigvee_{(v,e,v') \in C \text{ with } \mathit{act}(e) = a} \mathit{cond}(e).$$

If we split all actions in the LM-Cut landmark L with their respective context, the set of actions $\{a_{\varphi_a} \mid a \in L\}$ will be a landmark of the modified task. So we can admissibly count the landmark cost, reduce the cost of all a_{φ_a}, leave the cost of all $a_{\neg\varphi_a}$ unchanged, and proceed.

However, this idea cannot be implemented directly because we leave the STRIPS^c formalism with the context splitting. To see this, consider a context split of action a.

The precondition of the first new action a_{φ_a} is of the form $\mathit{pre}(a) \wedge (\mathit{cond}(e_1) \vee \cdots \vee \mathit{cond}(e_n))$ for some conditional effects $e_1, \ldots, e_n \in \mathit{eff}(a)$. Since the precondition $\mathit{pre}(a)$ and the effect conditions are all conjunctions of atoms, we can break up the action into n new STRIPS^c actions $a^e_{\varphi_a} = \langle \mathit{pre}(a) \wedge \mathit{cond}(e), \mathit{eff}(a) \rangle$ for $e \in \{e_1, \ldots, e_n\}$. Whenever a plan contains an action a_{φ_a}, there would also be a plan using an action $a^e_{\varphi_a}$ instead and vice versa.

The problem arises from the second new action $a_{\neg\varphi_a}$ whose precondition in general cannot be expressed as a negation-free formula. So we cannot easily reformulate these actions in STRIPS^c as we did with the actions a_{φ_a}.

As a solution, we propose *relaxed context splitting* which ignores the condition $\neg\varphi_a$ and simply preserves the original action:

Definition 4 (Relaxed Context Splitting) Relaxed context splitting *of an action a with context φ adds a new action $a_\varphi = \langle \mathit{pre}(a) \wedge \varphi, \mathit{eff}(a) \rangle$ with cost $c(a)$ to the task.*

Like unrelaxed context splitting, relaxed context splitting preserves the goal distance of states. It also preserves the value $V^{\max}$ of all variables: in general, adding actions to a task can only lead to a decrease of $V^{\max}$. However, in this case a decrease cannot happen: the new actions have the same effects and costs as the original ones but their precondition is a superset of the original precondition. Therefore the cost $C^{\max}$ of the effects of the new action cannot be lower than the one of the original effects, so no variable can be achieved more cheaply.

Unfortunately, with *relaxed* context splitting the set of actions $\{a_{\varphi_a} \mid a \in L\}$ is not a landmark of the modified task because a plan could contain action $a \in L$ instead of a_{φ_a}. So we cannot obviously apply the cost adaption as proposed at the beginning of this section. In the next section we will show that we still can define an extension to LM-Cut based on relaxed context splitting that preserves the desired properties of the heuristic.

LM-Cut with Relaxed Context Splitting

The key insight of our proposed heuristic is that we can safely leave the cost of all actions unchanged in each round of the LM-Cut computation as long as we add new reduced-cost actions that "fit" the context of the cut.

Definition 5 (LM-Cut with relaxed context splitting) *The* LM-Cut heuristic with relaxed context splitting *($h^{\text{LM-Cut}}_{\text{context}}$) instantiates the generic heuristic from Definition 2. In the task adaption step, for every edge $(v, e, v') \in C$ it extends the task with an action $a_e = \langle pre(a) \cup cond(e), eff(a) \rangle$ with $cost(a_e) = cost(a) - c_{min}$, where $a = act(e)$.*

In our example, we discover the landmark $\{disembark(A)\}$ in the first round (Figure 2a). Since there is only one effect in the cut, the disjunction in the context collapses to a single condition $\varphi_{stop\text{-}f_2} = cond(disembark(A)) = boarded(A)$. With relaxed context splitting we create the new action $stop\text{-}f_2{}' = stop\text{-}f_{2_{disembark(A)}}$ with the additional precondition $boarded(A)$ and the reduced cost 0.

In the next round (Figure 2b) we discover the landmark $\{disembark(B)\}$, which is handled just like in the first round and we add the action $stop\text{-}f_1{}' = stop\text{-}f_{1_{disembark(B)}}$ with the additional precondition $boarded(B)$ and the reduced cost 0.

In the final round (Figure 2c) the values $V^{\max}$ of all $boarded(p)$ and $served(p)$ facts and g are 1. The discovered landmark consists of a single *board*-effect. Which of the two is chosen depends on the condition choice function, but we assume that $board(A)$ is selected. Since this effect has no condition, the context is $\varphi_{stop\text{-}f_1} = cond(board(A)) = \top$ and the newly added action $stop\text{-}f_1{}''$ is identical to $stop\text{-}f_1$, except that it is free of cost.

With this new action, the $V^{\max}$ value of all facts now is 0. In particular, $boarded(B)$ can be reached from $boarded(A)$ with action $stop\text{-}f_2{}'$ without additional cost. The LM-Cut algorithm stops with a perfect heuristic value of 3.

In the following, we will show that $h^{\text{LM-Cut}}_{\text{context}}$ is admissible and dominates $h^{\max}$.

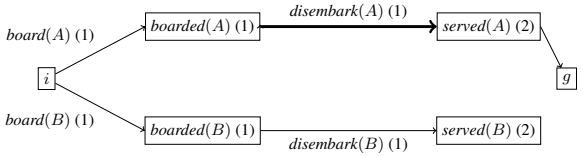
Theorem 0.2 *The LM-Cut heuristic with relaxed context splitting ($h^{\text{LM-Cut}}_{\text{context}}$) is* admissible.

Proof: We will show that the optimal delete-relaxation heuristic h^+ dominates $h^{\text{LM-Cut}}_{\text{context}}$. Since h^+ is admissible, we can conclude that $h^{\text{LM-Cut}}_{\text{context}}$ is also admissible.

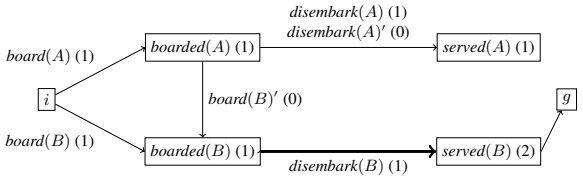
If $h^{\max}(\Pi) = 0$, the LM-Cut algorithm directly terminates with $h^{\text{LM-Cut}}(\Pi) = 0$, so there is nothing to show in this case. Otherwise, let Π and Π' be the (relaxed) tasks before and after a round of $h^{\text{LM-Cut}}_{\text{context}}$, respectively. We will show that $h^+(\Pi) \geq c_{min} + h^+(\Pi')$. The dominance of h^+ then follows from an inductive application of this argument.

Every atom (except i) of the task Π can only be made true by an effect of an incoming edge in the justification graph and this effect only triggers if the source of the edge has been true. So any plan of Π must use all action effects of some path from i to g in the justification graph and therefore also at least one effect from the cut.

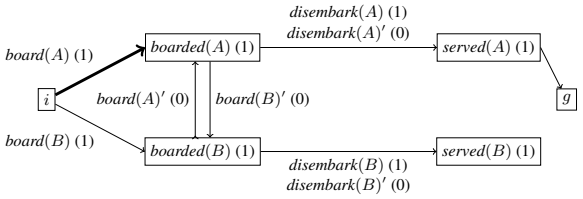
Let $\pi = \langle a_1, \ldots, a_n \rangle$ be an optimal plan for Π and let a_i be the first action in this plan whose application triggers an effect e from the cut. Π' has an action $a'_i = \langle pre(a_i) \cup cond(e), eff(a_i) \rangle$ with cost $c(a_i) - c_{min}$. Since e triggers in π, $pre(a_i) \cup cond(e)$ must be true after the application of $\langle a_1, \ldots, a_{i-1} \rangle$. As a_i and a'_i have the same effect, $\pi' = \langle a_1, \ldots, a_{i-1}, a'_i, a_{i+1}, \ldots, a_n \rangle$ is a plan for Π' that costs c_{min} less than π and therefore $h^+(\Pi') \leq h^+(\Pi) - c_{min}$. ∎



(a) Round 1.



(b) Round 2.



(c) Round 3.

Figure 2: Justification graphs in the LM-Cut rounds for $h^{\text{LM-Cut}}_{\text{context}}$ on the example task. Action costs for effects and $V^{\max}$ values for facts are given in parentheses, edges in the cut are bold.

The new heuristic is more informed than the maximum heuristic:

Theorem 0.3 *The LM-Cut heuristic with relaxed context splitting ($h^{\text{LM-Cut}}_{\text{context}}$) dominates $h^{\max}$.*

Proof: To increase clarity, in the following we denote the $V^{\max}$ value of a variable v in a task Π by $V^{\max}_{\Pi}(v)$.

If $h^{\max}(\Pi) = 0$ there is nothing to show. If $h^{\max}(\Pi) > 0$, we again denote the original (relaxed) task by Π and the transformed one after one LM-Cut round by Π'. We show that $h^{\max}(\Pi) \leq c_{min} + h^{\max}(\Pi')$. An inductive application of this argument proves the theorem.

Let A and A' denote the action sets of Π and Π', respectively. Consider the standard algorithm for computing $V^{\max}$: it uses a priority queue, initially containing the initial facts with a priority 0. The algorithm successively pops a fact with minimal priority from the queue and assigns it the priority as value $V^{\max}$ if the fact has not already been popped before. Whenever all relevant conditions of an effect e have been popped, the algorithm enqueues its added fact f with priority $C^{\max}(e)$.

Let $f' \in F$ be the first fact which is popped during the $V^{\max}_{\Pi'}$ computation that gets assigned a value $V^{\max}_{\Pi'}(f') < V^{\max}_{\Pi}(f')$, if such a fact exists. If g is popped before f' or no such fact f' exists, then $h^{\max}(\Pi) = V^{\max}_{\Pi}(g) = V^{\max}_{\Pi'}(g) = h^{\max}(\Pi')$ and there is nothing to show. In the following, we assume that g is popped after f' and hence $h^{\max}(\Pi') = V^{\max}_{\Pi'}(g) \geq V^{\max}_{\Pi'}(f')$.

Let e' be the effect due to which f' had been enqueued. Then e' must be an effect of some newly added action $a' \in A' \setminus A$: since f' is the first value with a differing $V^{\max}$, the change cannot be due to "cheaper" condition costs.

The action a' must have been added because an effect e (of an action a) occurred in the cut. Therefore, $a' = \langle \mathit{pre}(a) \cup \mathit{cond}(e), \mathit{eff}(a) \rangle$ with cost $\mathit{cost}(a') = \mathit{cost}(a) - c_{\mathit{min}}$ for some action a of Π and effect e of a. Let f be the fact added by e.

Since e was in the cut, f must have been in the goal zone and therefore it holds that $V^{\max}_{\Pi}(\mathit{pre}(a) \cup \mathit{cond}(e)) + \mathit{cost}(a) \geq V^{\max}_{\Pi}(f) \geq V^{\max}_{\Pi}(g) = h^{\max}(\Pi)$ (*).

We can bound $h^{\max}(\Pi)$ as follows:

$$h^{\max}(\Pi) \leq V^{\max}_{\Pi}(\mathit{pre}(a) \cup \mathit{cond}(e)) + \mathit{cost}(a) \quad (1)$$
$$= V^{\max}_{\Pi'}(\mathit{pre}(a) \cup \mathit{cond}(e)) + \mathit{cost}(a) \quad (2)$$
$$\leq V^{\max}_{\Pi'}(\mathit{pre}(a) \cup \mathit{cond}(e) \cup \mathit{cond}(e')) + \mathit{cost}(a) \quad (3)$$
$$= V^{\max}_{\Pi'}(f') + c_{\mathit{min}} \quad (4)$$

Statement (1) uses the previously derived bound (*). Equation (2) holds as $\mathit{pre}(a) \cup \mathit{cond}(e)$ is the precondition of a' and hence all facts in this set must have been popped before f' was enqueued by effect e'. Since f' is the first popped fact for which $V^{\max}_{\Pi} \neq V^{\max}_{\Pi'}$ it follows for all $p \in \mathit{pre}(a) \cup \mathit{cond}(e)$ that $V^{\max}_{\Pi}(p) = V^{\max}_{\Pi'}(p)$. Inequality (3) is due to $V^{\max}(P) \leq V^{\max}(P')$ if $P \subseteq P'$. The last line exploits that effect e' of action a' establishes the value $V^{\max}_{\Pi'}(f')$ and that $\mathit{cost}(a') = \mathit{cost}(a) - c_{\mathit{min}}$.

Overall we have shown that $h^{\max}(\Pi) \leq V^{\max}_{\Pi'}(f') + c_{\mathit{min}}$. Since we know from above that $h^{\max}(\Pi') \geq V^{\max}_{\Pi'}(f')$, it holds that $h^{\max}(\Pi) \leq h^{\max}(\Pi') + c_{\mathit{min}}$. ∎

We have seen that $h^{\text{LM-Cut}}_{\text{context}}$ preserves the desired properties of the LM-Cut heuristic for STRIPS. In the next section we will evaluate whether it also preserves its good performance.

Experimental Evaluation

For the evaluation we use the same sets of domains T0 and FSC as Haslum (2013). The T0 domains are generated by a compilation from conformant to classical planning by Palacios and Geffner (2009); the set FSC has been generated by the finite-state controller synthesis compilation by Bonet et al. (2009). In addition, we include tasks from the briefcase world from the IPP benchmark collection (Köhler 1999). We also use the Miconic simpleadl version from the benchmark set of the International Planning Competition (IPC-2000) because it has conditional effects but no derived predicates after grounding with Fast Downward.

We compare $h^{\max}$ and three variants of the LM-Cut heuristic:

- our version $h^{\text{LM-Cut}}_{\text{context}}$ using relaxed context splitting,
- the version $h^{\text{LM-Cut}}_{\text{basic}}$ mentioned by Keyder et al. (2012) that reduces the action cost of every action with an effect in the cut and does not dominate $h^{\max}$, and

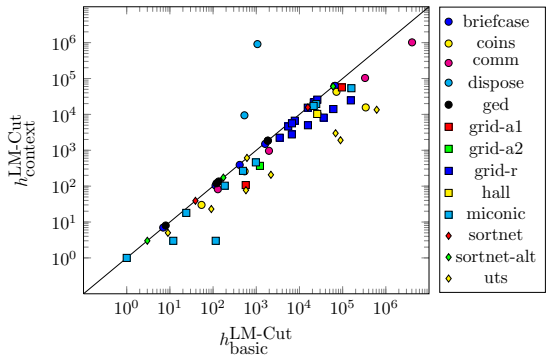


Figure 3: Expansions (excluding the ones on the last f-layer) of $h^{\text{LM-Cut}}_{\text{basic}}$ and $h^{\text{LM-Cut}}_{\text{context}}$ for the commonly solved tasks.

- the standard LM-Cut version $h^{\text{LM-Cut}}_{\text{standard}}$ (Helmert and Domshlak 2009), which does not support conditional effects. For this variant, we transform the tasks with the exponential compilation by Nebel (2000).

All heuristics were implemented in the Fast Downward planning system (Helmert 2006), which separates the preprocessing phase from the actual search phase. For each phase, we set a time limit of 30 minutes and a memory limit of 2 GB per task. The experiments were conducted on Intel Xeon E5-2660 processors (2.2 GHz).

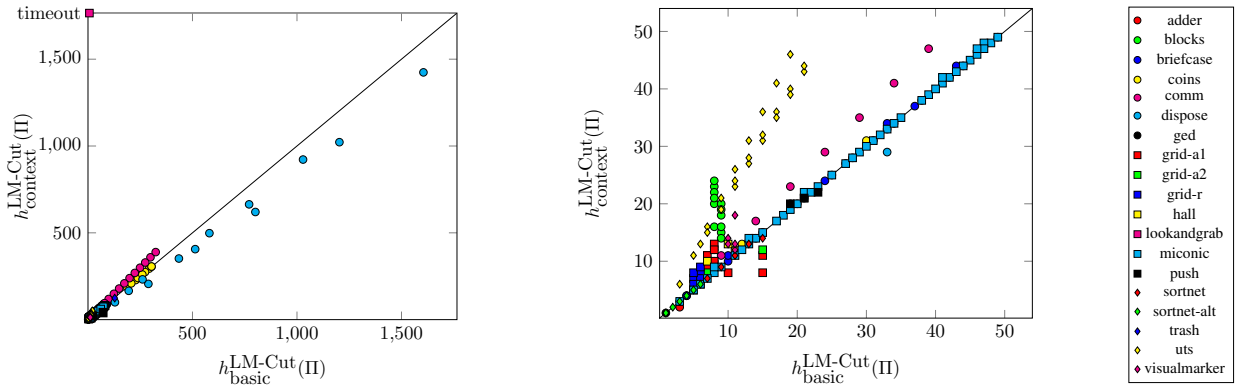
We first compare the two LM-Cut versions that support conditional effects directly.

Figure 3 plots the number of A* expansions of $h^{\text{LM-Cut}}_{\text{basic}}$ vs. those of $h^{\text{LM-Cut}}_{\text{context}}$ for each task. As expected, context splitting almost always gives equal or better guidance than the basic approach. The only exception is the t0-grid-dispose domain in which $h^{\text{LM-Cut}}_{\text{basic}}$ is superior.

To get a clearer idea of the difference of the heuristic estimates, we compare the heuristic values of the initial states in Figure 4. The very high estimates in the t0-grid-dispose domain render the results of the other tasks almost indistinguishable. For this reason, Figure 4b shows the same results but only includes tasks where both heuristic estimates are below 50. Overall, we note that the estimates of $h^{\text{LM-Cut}}_{\text{context}}$ are much better than those of $h^{\text{LM-Cut}}_{\text{basic}}$ and in the t0-uts domain they are always at least twice as high.

Since the results of the t0-grid-dispose domain stick out negatively, we had a closer look at this domain to understand the different performance. A deeper analysis of one task reveals that the variant with relaxed context splitting makes unfavorable decisions when selecting one of several candidates with maximal $V^{\max}$ for the condition choice function. As a result, effects that achieve different sub-goals end up in one cut, and they all become cheaper in the next round. A similar effect can also be observed with $h^{\text{LM-Cut}}_{\text{standard}}$ in the STRIPS freecell domain.

Table 1 shows the number of solved instances for all heuristics (omitting domains where no task was solved by any heuristic). Note that $h^{\text{LM-Cut}}_{\text{standard}}$ cannot be directly compared to the other heuristics based on these numbers because it requires a compilation of the task that removes conditional effects. The small numbers behind the domain names state



(a) All values.

(b) Heuristic values below 50.

Figure 4: Heuristic values of the initial state for $h^{\text{LM-Cut}}_{\text{basic}}$ and $h^{\text{LM-Cut}}_{\text{context}}$.

	$h^{\text{LM-Cut}}_{\text{standard}}$	$h^{\max}$	$h^{\text{LM-Cut}}_{\text{basic}}$	$h^{\text{LM-Cut}}_{\text{context}}$
briefcaseworld (9,50)	6	7	**9**	8
fsc-grid-a1 (0,16)	-	**2**	**2**	**2**
fsc-grid-a2 (0,2)	-	**1**	**1**	**1**
fsc-grid-r (0,16)	-	**15**	**15**	13
fsc-hall (0,2)	-	**1**	**1**	**1**
gedp-ds2ndp (0,24)	-	**18**	12	12
miconic(149,150)	78	70	**141**	**141**
t0-coins (20,30)	**14**	10	**14**	**14**
t0-comm (25,25)	**5**	4	**5**	**5**
t0-grid-dispose (0,15)	-	0	**3**	2
t0-grid-lookandgrab (0,1)	-	**1**	**1**	0
t0-sortnet (0,5)	-	**2**	**2**	**2**
t0-sortnet-alt (1,6)	1	**4**	**4**	**4**
t0-uts (6,29)	5	6	8	**10**
Sum (210,371)	109	141	**218**	215

Table 1: Coverage results.

for how many tasks the Fast Downward preprocessing phase completed with and without the compilation. It is apparent that – at least with the exponential transformation – compiling away conditional effects and using a standard heuristic is not competitive.

Except for the Miconic domain, which dominates the summary results with its large number of tasks, the three remaining heuristics are surprisingly close to each other and each one is better than the others in some domain. While $h^{\max}$ performs worst as expected, the better guidance of $h^{\text{LM-Cut}}_{\text{context}}$ does not translate to higher coverage than $h^{\text{LM-Cut}}_{\text{basic}}$ because it does not offset the additional time for the heuristic evaluations. However, in the conclusion we will explain how this might be resolvable in future work.

Conclusions and Future Work

We presented an extension of the LM-Cut heuristic to conditional effects that is admissible and dominates the maximum heuristic. For this purpose we introduced context splitting as a new general concept of which we belief that it will prove useful also for other applications.

One obstacle for the new heuristic is that it adds many new actions in every round of its computation, which causes computational overhead in the following rounds. However, we hope that we can resolve this to some extent in future work: in certain respects, the computation of $h^{\text{LM-Cut}}_{\text{context}}$ is based on the individual conditional effects plus their action precondition. From this perspective, the context split adds many "equivalent" effects in every round. If it is possible to represent them only once (similar to the way it is done in an efficient $h^{\max}$ implementation), we expect a significant speed-up of the computation.

To avoid unfavorable selections of the condition choice function, it might be beneficial to deploy additional strategies, such as preferring conditions that were not added by a context split. As this paper focuses on the theoretical properties of the heuristics, we leave this topic for future work.

Acknowledgments

This work was supported by the German Research Foundation (DFG) as part of the project "Kontrollwissen für domänenunabhängige Planungssysteme" (KontWiss) by DFG grant HE 5919/2-1.

References

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

Bonet, B.; Palacios, H.; and Geffner, H. 2009. Automatic derivation of memoryless policies and finite-state controllers using classical planners. In Gerevini, A.; Howe, A.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 34–41. AAAI Press.

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

Haslum, P. 2013. Optimal delete-relaxed (and semi-relaxed) planning with conditional effects. In Rossi, F., ed., *Proceed-*

ings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013), 2291–2297.

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 Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 162–169. AAAI Press.

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

Keyder, E.; Hoffmann, J.; and Haslum, P. 2012. Semi-relaxed plan heuristics. In McCluskey, L.; Williams, B.; Silva, J. R.; and Bonet, B., eds., *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS 2012)*, 128–136. AAAI Press.

Köhler, J. 1999. Handling of conditional effects and negative goals in IPP. Technical Report 128, Institute for Computer Science, Albert-Ludwigs-Universität, Freiburg, Germany.

Nebel, B. 2000. On the compilability and expressive power of propositional planning formalisms. *Journal of Artificial Intelligence Research* 12:271–315.

Palacios, H., and Geffner, H. 2009. Compiling uncertainty away in conformant planning problems with bounded width. *Journal of Artificial Intelligence Research* 35:623–675.

Röger, G.; Pommerening, F.; and Helmert, M. 2014. Optimal planning in the presence of conditional effects: Extending LM-Cut with context splitting. In *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI 2014)*.

Width-based Algorithms for Classical Planning: New Results

Nir Lipovetzky
The University of Melbourne
Melbourne, Australia
nir.lipovetzky@unimelb.edu.au

Hector Geffner
ICREA & Universitat Pompeu Fabra
Barcelona, Spain
hector.geffner@upf.edu

Abstract

We have recently shown that classical planning problems can be characterized in terms of a width measure that is bounded and small for most planning benchmark domains when goals are restricted to single atoms. Two simple algorithms have been devised for exploiting this structure: Iterated Width (IW) for achieving atomic goals, that runs in time exponential in the problem width by performing a sequence of pruned breadth first searches, and Serialized IW (SIW) that uses IW in a greedy search for achieving conjunctive goals one goal at a time. While SIW does not use heuristic estimators of any sort, it manages to solve more problems than a Greedy BFS using a heuristic like h_{add}. Yet, it does not approach the performance of more recent planners like LAMA. In this short paper, we introduce two simple extension to IW and SIW that narrow the performance gap with state-of-the-art planners. The first involves changing the *greedy search* for achieving the goals one at a time, by a *depth-first search* that is able to backtrack. The second involves computing a relaxed plan once before going to the next subgoal for making the pruning in the *breadth-first procedure* less agressive, while keeping IW exponential in the width parameter. The empirical results are interesting as they follow from ideas that are very different from those used in current planners.

A version of this short paper has been accepted at ECAI (Lipovetzky and Geffner 2014)

Introduction

The main approach for domain independent planning is based on heuristic search with heuristics derived automatically from problems (McDermott 1996; Bonet and Geffner 2001). To this, recent planners add other ideas like helpful actions, landmarks, and multiqueue best-first search for combining different heuristics (Hoffmann and Nebel 2001; Helmert 2006; Richter and Westphal 2010). From a different angle, we have recently shown that most of the benchmark domains are easy when the goals contain single atoms, and that otherwise, goals can be easily serialized (Lipovetzky and Geffner 2012). More precisely, we showed that the former problems have a low width, and developed an algorithm, Iterative Width (IW) that runs in time that is exponential in the problem width by performing a sequence of pruned breadth first searches. This algorithm is used in the context of another algorithm, Serialized IW (SIW), that achieves conjunctive goals by using IW greedily for achieving one goal at a time. Surprisingly, the blind-search algorithm SIW which has no heuristic guidance of any sort, performs better than a greedy best-first search guided by delete-relaxation heuristics. SIW, however, does not perform as well as the most recent planners that incorporate other ideas as well.

In this short paper, we introduce two simple extensions to IW and SIW that narrow the performance gap between width-based algorithms and state-of-the-art planners. The first extension involves changing the *greedy search* for achieving the goals, one at a time, by a *depth-first search* able to backtrack. The second involves computing a relaxed plan once before going to the next subgoal for making the pruning in the *breadth-first procedure* less aggressive, while keeping IW exponential in the width parameter.

Preliminaries

We assume a STRIPS problem $P = \langle F, I, O, G\rangle$, where F is the set of atoms, I is the set of atoms characterizing the initial state, O is the set of actions, and G is the set of goal atoms.

The algorithm *Iterated Width* or *IW* consists of a sequence of calls *IW*(i) for $i = 0, 1, \ldots, |F|$ until the problem is solved. Each iteration *IW*(i) is a breadth-first search that prunes states that do not pass a *novelty* test; namely, for a state s in *IW*(i) *not* to be pruned there must be a tuple t of at most i atoms such that s is the first state generated in the search that makes t true. The time complexities of *IW*(i) and *IW* are $O(n^i)$ and $O(n^w)$ respectively where n is $|F|$ and w is the problem width. The width of existing domains is low for atomic goals, and indeed, 89% of the benchmarks can be solved by *IW*(2) when the goal is set to any of the atoms in the goal (Lipovetzky and Geffner 2012). The width of the benchmark domains with conjunctive goals, however, is not low in general, yet such problems can be serialized.

The algorithm *Serialized Iterative width* or *SIW* uses *IW* for serializing a problem into subproblems and for solving the subproblems. Basically, SIW uses IW to achieve one atomic goal at a time, greedily, until all atomic goals are achieved jointly. In between, atomic goals may be un-

Copyright © 2014, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

	GBFS	SIW	SIW$^+$	FF	DFS$^+$	BFS(f)	PROBE	LAMA'11
Barman	0	0	17	0	**20**	**20**	**20**	**20**
Elevators	2	17	19	**20**	**20**	17	**20**	**20**
Floortile	3	0	0	2	0	**6**	5	5
NoMystery	6	0	0	5	4	**15**	6	10
Openstacks	14	5	**20**	**20**	**20**	16	14	**20**
Parcprinter	19	**20**	**20**	**20**	**20**	17	13	19
Parking	5	**20**	**20**	19	**20**	**20**	19	19
Pegsol	**20**	0	1	**20**	19	**20**	**20**	**20**
Scanalyzer	**18**	16	16	17	**18**	17	**18**	**18**
Sokoban	15	0	0	15	1	13	15	**17**
Tidybot	17	5	17	**19**	**19**	18	**19**	15
Transport	4	13	14	8	16	16	16	**17**
VisitAll	3	**20**	**20**	3	**20**	**20**	18	**20**
Woodworking	2	19	**20**	19	**20**	**20**	**20**	19
All	128	135	184	187	217	235	223	**239**

Table 1: Number of instances solved. Bold shows best performer.

done, but after each invocation of IW, each of the previously achieved goals must hold. SIW will thus never call IW more than $|G|$ times where $|G|$ is the number of atomic goals. SIW compares surprisingly well to a baseline heuristic search planner based on greedy best-first search and the h_{add} heuristic (Bonet and Geffner 2001), but doesn't approach the performance of the most recent planners. For this, a new planner $BFS(f)$ was introduced that integrates the *novelty* measure with helpful-actions, landmarks and delete-relaxation heuristics (Lipovetzky and Geffner 2012). Here we aim to achieve similar results but with different and simpler ideas.

Extensions: *IW*$^+$ and *DFS(i)*

The first extension, from *IW* to *IW*$^+$ involves a slight change in the pruning criterion in the breadth-first search *IW*(i) procedure. When the new procedure *IW*$^+(i)$ is called from a state s, a *relaxed plan* from s is computed, so that the states s' generated by *IW*$^+(i)$ keep a count of the number of atoms in the relaxed plan from s that have been achieved in the way to s'. These atoms are the ones made true by actions in the relaxed plan that do not hold in s. We say that state s' makes the *extended tuple* (t, m) true if s' makes the tuple t true and m is the the number of atoms in the relaxed plan from s that are made true by the sequence of actions leading to s' in *IW*$^+$. For the state s' in the breadth-first search underlying *IW*$^+(i)$ *not* to be pruned, the new condition is that there must be a tuple t with at most i atoms, such that s' is the first state in the search that makes the *extended tuple* (t, m) true. An interesting property and motivation of the *IW*$^+$ algorithms is that all *delete-free* problems will be solved efficiently, as they will be solved by *IW*$^+(1)$. The same is not true of *IW*(1).

While the algorithm *IW* calls the algorithm *IW*(i) sequentially, *IW*$^+$ calls *IW*$^+(i)$ instead. The complexity of *IW*$^+$ is $O(n^{w+1})$ where w is the problem width, as in the worst case, the number of atoms in a relaxed plan can be equal to the number of fluents.

The second extension is a depth-first search algorithm that extends the serialization of conjunctive goals made greedily by *SIW* with the ability to backtrack. For this, the depth-first search algorithm *DFS(i)* uses the positive integer parameter i to indicate that invocations of the *IW*$^+$ procedure should involve a sequence of *IW*$^+(0)$, *IW*$^+(1), \ldots,$ *IW*$^+(i)$ calls that should fail and lead to a backtrack when *IW*$^+(i)$ fails to achieve one more atomic goal. *SIW*, on the other hand, will keep trying *IW*$(i+1)$, *IW*$(i+2)$, and so on, and will never backtrack.

DFS(i) can be understood as normal depth-first search where the actions in each node are sets of "macro actions" *IW*$^+(1), \ldots,$ *IW*$^+(i)$ performed in order, such the children resulting from the "macro actions" in *IW*$^+(k)$ applied in a state s are the states s' reachable by *IW*$^+(k)$ from s that achieve all the goal atoms in s plus one.

Notice that while *DFS(i)* computes a relaxed plan once for each *IW*$^+$ call, *DFS(i)* does not use the relaxed plan for computing heuristic estimates. Rather it uses the relaxed plans to make the pruning in the breadth-first searches in *IW*$^+(i)$ less aggressive, while keeping its complexity exponential in the i parameter.

Experiments

The algorithm *SIW*$^+$ is *SIW* with the *IW* procedure replaced by *IW*$^+$. Thus, *SIW*$^+$ incorporates the first extension only, while *DFS(i)* incorporates both extensions. In order to evaluate their performance, we include results for a baseline heuristic search planner made of a greedy best-first search (GBFS) driven by h_{add} (Bonet and Geffner 2001), and state-of-the-art planners such as FF (Hoffmann and Nebel 2001), PROBE (Lipovetzky and Geffner 2011), LAMA-11 (Richter and Westphal 2010) and BFS(f). All planners are run over the set of benchmarks from the last International Planning Competition on a 2.40GHz Intel Processor, with processing time or memory out of 30min and 2GB.

		GBFS	SIW	SIW$^+$	FF	DFS$^+$	BFS(f)	PROBE	LAMA'11
Barman	T	0	0	3.59	0	4.52	15.51	**16.56**	10.18
	Q	0	0	15.3	0	17.95	**19.19**	17.39	15.19
Elevators	T	0.01	1.23	11.33	11.78	15.62	2.41	6.16	**16.33**
	Q	1.14	15.24	10.83	**19.44**	10.93	12	14.62	18.24
Floortile	T	0.06	0	0	0.59	0	**3.26**	2.2	2.8
	Q	2.96	0	0	1.83	0	**5.75**	4.61	4.67
NoMystery	T	1.59	0	0	5	0.08	**12.04**	2.52	9.48
	Q	5.74	0	0	5	3.77	**14.57**	5.72	9.72
Openstacks	T	0.64	1.73	15.72	11.62	**19.88**	1.99	1.64	13.94
	Q	13.4	5	**19.83**	18.81	**19.83**	15.38	13.79	19.24
Parcprinter	T	16.06	**20**	**20**	**20**	**20**	7.85	11.74	19
	Q	18.9	19.01	19.01	**19.79**	19.01	16.01	12.88	18.31
Parking	T	0.07	**19.19**	18.82	2.57	18.5	2.35	2.15	5.06
	Q	2.53	**19.53**	19.36	10.94	19.42	10.72	6.27	10.68
Pegsol	T	19.01	0	1	**20**	7.3	19.5	16.86	19.23
	Q	17.89	0	1	17.93	17.1	**18.99**	18	17.92
Scanalyzer	T	5.15	**15.24**	14.32	13.27	14.97	9.91	13.75	10.71
	Q	14.81	13.48	13.4	15.82	14.88	14.04	**16.56**	14.86
Sokoban	T	2.2	0	0	9.47	0.3	6.68	**11.22**	10.08
	Q	9.66	0	0	**13.83**	0.93	10.58	10.29	13.23
Tidybot	T	1.66	3.04	6.88	**14.41**	8.44	2.32	8.02	2.81
	Q	13.04	4.86	14.77	14.43	15.81	15.31	**16.46**	12.54
Transport	T	0.09	3.11	13.67	0.18	**15.17**	4.91	5.84	9.23
	Q	2.66	11.79	12.27	6.04	13.38	14	11.14	**15.82**
VisitAll	T	0.14	17.37	12.81	1.88	12.96	**20**	5.68	6.46
	Q	0.25	19.5	19.5	1.58	19.5	**20**	14.39	14.69
Woodworking	T	1.01	7.92	6.05	**19**	7.18	2.16	2.74	3.44
	Q	2	17.47	17.88	16.97	17.81	19.86	**20**	15.51
All	T	47.69	88.84	124.2	129.78	**144.92**	110.89	107.07	139.96
	Q	104.98	125.88	163.15	162.41	190.91	**206.49**	182.12	200.62

Table 2: Time(T), and plan length (Q) of first solution as IPC scores. Bold shows best performer.

Interestingly, the results in Table 1 and 2 highlight the big gap in performance reached by just considering the first extension *IW*$^+$ in the greedy serialization *SIW*$^+$, reaching indeed the performance of FF that uses helpful actions in addition to heuristics and was the state-of-the-art planner until few years ago. When the second extension is considered, the performance compares well with current state-of-the-art planners that also incorporate multiple heuristics and landmarks. Indeed, as shown in Table 2, even by having a smaller coverage, *DFS(i)* has the *highest IPC score*[1] in terms of speed and doesn't perform bad at all in terms of the *quality IPC score*. *DFS(i)* backtracks mainly in Pegsol, NoMystery, and Sokoban: 31000, 21217, and 137 times on average.

[1]Score for each planner is the sum of T/T^* (Q/Q^*), where T (Q) is solution time (length) divided by the fastest (shortest) solution found T^* (Q^*)

Conclusion

We have set to explore further the potential of classical planning algorithms based on width considerations. We have shown how two simple extensions to the Iterated Width (IW) and Serialized IW (SIW) algorithms introduced by Lipovetzky and Geffner result in a simple planner whose performance approaches the performance of the best current planners. The planner is based on a depth-first search engine that uses pruned breadth-first procedures as macros. The ideas are thus very different than those that can be found in current planners that usually rely on heuristic estimators, helpful actions, and landmarks, suggesting that they may deserve further consideration.

References

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

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

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

Lipovetzky, N., and Geffner, H. 2011. Searching for plans with carefully designed probes. In *Proceedings of the Twenty-First International Conference on Automated Planning and Scheduling (ICAPS 2011)*, 154–161.

Lipovetzky, N., and Geffner, H. 2012. Width and serialization of classical planning problems. In *Proceedings of the Twentieth European Conference on Artificial Intelligence (ECAI 2012)*, 540–545.

Lipovetzky, N., and Geffner, H. 2014. Width-based algorithms for classical planning: New results. In *Proceedings of the Twenty-First European Conference on Artificial Intelligence (ECAI 2014)*.

McDermott, D. 1996. A heuristic estimator for means-ends analysis in planning. In *Proceedings of the Third International Conference on Artificial Intelligence Planning Systems (AIPS-96)*.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research* 39:122–177.

To reopen or not to reopen in the context of Weighted A*? Classfications of different trends

Vitali Sepetnitsky **Ariel Felner** **Roni Stern**
Information Systems Engineering
Ben-Gurion University
{sepetnit, ariel.felner1, roni.stern}@gmail.com

Abstract

This paper studies the tradeoffs between *reopening* and *not reopening* nodes in the context of Weighted A* (WA*). A straightforward intuitive scenario is that reopening nodes results in finding shorter solutions than not reopening nodes, at the cost of expanding more nodes. But, there are other scenarios where other tendencies occur. In this paper we classify these tendencies and show an example graph where different tendencies are evident only by varying the W parameter of WA* and without changing the structure of the graph. We then experimentally demonstrate the different tendencies on variants of the sliding tile puzzle and on grid based maps. Finally, we provide experimental support that intelligent reopening polices might lead to better performance. This is a work in progress and we discuss several future directions.

Introduction

Weighted A*

Since A* (Hart, Nilsson, and Raphael 1968) may run out of memory and out of time, many search algorithms settle for suboptimal solutions. The most famous, and perhaps most simple suboptimal search algorithm is *Weighted A** (WA*) (Pohl 1970) – the focus of this paper.

Similar to A*, WA* uses two data structures: *OPEN* and *CLOSED*. Initially, *OPEN* contains only the start state and *CLOSED* is empty. At every iteration, the algorithm chooses the state in *OPEN* that minimizes the cost function $f(n) = g(n) + W \cdot h(n)$, where $g(n)$ is the cost of the lowest cost path found so far from the start state to n, $h(n)$ is an admissible heuristic estimate of the cost of reaching the goal from n, and W is a parameter. The chosen state is expanded, generating its successors. The expanded state is moved to *CLOSED* and its successors are inserted to *OPEN*. WA* continues until a goal state is chosen for expansion or *OPEN* is empty. As W grows, WA* is expected to find solutions faster (but with decreasing quality) since nodes estimated to be close to the goal (with lower heuristic value) become more attractive.[1]

Reopening of nodes in WA*

Some states have multiple paths from the start state to them, of different costs. It is well-known that A* with a *consistent* heuristic (and thus the f-cost is monotonically increasing) expands a state only after the lowest cost path to it was found. This is not the case for non-monotonic f-functions such as the one used by WA*. In this case, a state n may be generated with a smaller g-value, after it has been already expanded, and inserted to *CLOSED*. At this point, it can be removed from *CLOSED* and put back in *OPEN* – an action called *reopening*. Reopening is optional; for every closed state which is seen with a smaller g-value, a decision needs to be made whether to reopen it or not. In the later case, the newly seen state is just discarded. In this paper we focus on the two extreme reopening polices: *always reopen* (AR) and *never reopen* (NR).

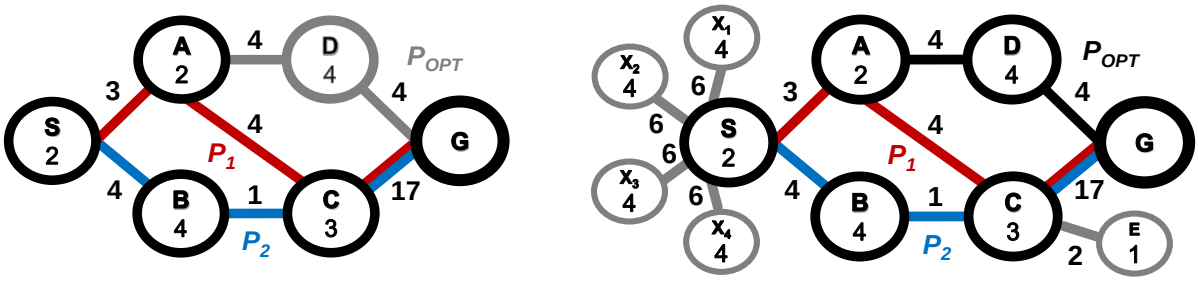
Example of reopening

Figure 1a illustrates a reopening scenario with WA* for $W = 4$ (denoted WA*(4)). h-values (inside nodes) and edge costs are shown. For now, assume that state D and its connecting edges (all colored in gray) are not in the graph. The graph has two paths from S to G, $P_1 = \{S, A, C, G\}$ of cost 24, and $P_2 = \{S, B, C, G\}$ of cost 22. The first sequence of nodes expanded by WA*(4) is: $\{S(8), A(11), C(19)\}$ (numbers are f-values). *OPEN* now contains $B(20)$ and $G(24)$. So, B is expanded and C is generated but now with a better g-value of 5. With AR, C is reopened and re-expanded. Then G is generated with $f(G) = 22$ and we update *OPEN* accordingly. Finally, G is expanded and the algorithm terminates and returns P_2. Alternatively, with NR, after expanding B, C will be seen but will be *discarded* and not re-inserted to *OPEN*. In this case, G with $f(G) = 24$ will be expanded immediately and path P_1 will be returned.

In our example, reopening produced a shorter path but expanded more states (via the alternate shorter path). This is a straightforward and intuitive scenario. Reopening improves the solution as it allows finding better paths to previously generated nodes. This comes at the cost of more nodes expansions caused by reopening as some nodes are expanded more than once. However, there are scenarios where AR finds longer solutions than NR and there are scenarios where AR expands fewer states than NR.

In this paper we classify the different possible behaviors into 9 different cases. We then show an example graph where most of these cases exist by only varying W and without changing the structure of the graph. We then experimen-

[1] (Wilt and Ruml 2012) discussed some counter examples for this general trend. See below.



(a) Different paths returned

(b) Different nodes expanded

Figure 1: Reopening examples

tally show that these cases are evident on common testbed domains such as the sliding tile puzzle and grid-based maps. Finally we provide evidence that more intelligent reopening polices can potentially improve the extreme AR and NR policies.

Related work

The notion of reopening has been first introduced by (Pohl 1970) and discussed in a variety of papers thereafter (Likhachev, Gordon, and Thrun 2003; Hansen and Zhou 2007; Thayer and Ruml 2008). For example, the policy of not allowing reopening (NR) in the context of A^*_ϵ (Pearl and Kim 1982) and DWA* (Pohl 1973) was proposed earlier (Ebendt and Drechsler 2009).

A number of authors discussed the influence of AR and of NR on the path returned and on the number of expanded nodes. For example, (Thayer and Ruml 2010) write that the NR policy (referred to as "ignoring duplicates") "can decrease solution quality, and even quality bounds for some algorithms". They also report that "ignoring duplicates (= NR) allows ... generating orders of magnitude fewer nodes". (Malima and Sabanovic 2007) write that "If reopening nodes is allowed it will produce a better and smoother path, but it will take more time". On the other hand other authors recognized that other trends are also possible. For example, (Hansen and Zhou 2007) discuss some reasons for why AR may require fewer node expansions than NR in some cases, but did not discuss the case where AR finds a worse solutions than NR. We consider the full spectrum of the relative performance of AR and NR.

The AR vs. NR dilemma is one choice that a WA* implementer needs to consider. (Wilt and Ruml 2012) discussed a complementing dilemma: what is the impact of increasing W. While it is common to assume that higher W results in a faster search, they showed that in several problem domains the number of expanded nodes (often correlated with runtime of finding the goal) is not monotonically decreasing when W increases. They attempted to characterize in which domains increasing W would not be effective by investigating the influence of various domain- and heuristic attributes. While still an open question their results point out the importance of the correlation between distance estimates (i.e., number of actions to the goal) and the used heuristic, which estimates the *cost* of the actions that reach the goal[2]. In this paper we do not investigate the impact of the W parameter, but provide a systematic classification of the different possible relations between AR and NR, demonstrating them on an example graph and experimentally on a number of domains. Thus, our work is a step towards deciding when to use which reopening policy (AR or NR), while (Wilt and Ruml 2012) made a step towards deciding how to tune W.

Ranges of W with different behavior

Any possible relation between AR and NR can occur regarding the path returned and the number of nodes expanded. We show this now on our example graph. The different trends can occur by only modifying the weight W but without changing the structure of the graph.

Let $P(AR)$ and $P(NR)$ denote the cost of the path returned by the AR and NR policies, respectively. Let $P^+, P^=, P^-$ denote a win for AR ($P(AR) < P(NR)$), a tie ($P(AR) = P(NR)$) and a win for NR ($P(NR) < P(AR)$), respectively. Similarly, let $N(AR)$ and $N(NR)$ denote the number of nodes expanded by the AR and NR policies, respectively. Finally, let $N^+, N^=, N^-$ denote a win for AR ($N(AR) < N(NR)$), a tie ($N(AR) = N(NR)$)and a win for NR ($N(NR) \leq N(AR)$), respectively. There are $3 \times 3 = 9$ combinations and we use a four character notation P^*N^* to denote these cases. For example, P^+N^- denotes the case where in the path aspect (P), AR is the winner, while in the nodes expanded aspect (N), NR is the winner.

Discussion on the returned path

We first discus the returned path and then move to also discuss the number of nodes expanded.

Consider again the graph from Figure 1a but now also including state D and the edges connected to it. This adds a new path to the goal $P_{OPT} = \{S, A, D, G\}$ with cost 11. For all possible values of $W > 1$, the first sequence of nodes expanded is: $\{S, A, C, B\}$. In the last step, when B is expanded, C is re-generated via a shorter path. If C is reopened (AR), a better path to G will be found via path P_2

[2]An admissible heuristic is an estimate that is a lower bound on the lowest cost path to the goal, which in some domains is very different from the number of actions to the goal.

(=$\{S, B, C, G\}$). If C is not reopened (NR) it will be discarded and P_2 will never be found. In addition, if the search finds a path to the goal with $f(G) < f(D)$, then node D will remain in *OPEN* and the optimal path to G, P_{OPT}, will not be found. However, if D reaches the top of *OPEN*, it will be expanded and P_{OPT} will be revealed. Therefore, the f-value of B, D and G and their respective order in *OPEN* determine the identity of the returned path.

We will show that different values of W cause different expansion order of these nodes and as a consequence different paths will be returned. There are four ranges for values of $W > 1$ that influence the identity of the returned path. We label them by *range 1* through *range 4* as shown in Table 1. These ranges and this table are discussed in depth below.

Discussion on the number of nodes expanded

We make additional changes to our graph in order to enrich the ranges of W to also have different tendencies in the node expansions. The resulting graph is shown in Figure 1b. First, we add a single state E connected to C with an edge of 2 and with $h(E) = 1$. Note that $f_{P_1}(E) = 9 + W$ (i.e., $f(E)$ via path P_1) while $f_{P_2}(E) = 7 + W$ (i.e., $f(E)$ via path P_2). Expanding E will not reveal a new path but will add 1 to the node count every time a path from C to G is explored. In addition, we add four states (X_1,X_2,X_3,X_4), connected to S with edges of cost 6, each having $h(X_i) = 4$. Thus, $f(X_i) = 6 + 4 \cdot W$. Expanding the X_i nodes does not reveal any new path to G, but adds 4 to the total expansion count. Note that for every value of $W > 1$ the following two inequalities hold:

- **(1:)** $f(D) > f(X_i)$ (as $g(D) > g(X_i)$ and $h(D) = h(X_i)$).
- **(2:)** $f(D) > f(E)$.

These two facts imply that if nodes X_i and E are expanded, it must be before the expansion of D.

The four ranges mentioned above include subranges, depicted in the right part of Table 1, where the X_i nodes are, or are not, expanded by the AR and NR policies. Next, 7 out of the 9 possibilities ($\{P^+, P^=, P^-\} \times \{N^+, N^=, N^-\}$), are demonstrated on the graph in Figure 1b, just by varying the value of the W parameter. Since E and the X_i nodes never influence the identity of the path returned, then, whenever we talk about the lengths of the path returned, it is sufficient to look at Figure 1a. When we want to show differences in node expansions we will refer to Figure 1b.

(Range 1:) $\mathbf{1 \leq W < 3.75}$**:** Here W is small enough and thus $f(D) < 7 + 3.75 \cdot 4 = 22$. In this case, when D and G (either with $f_{P_1}(G) = 24$ or with $f_{P_2}(G) = 22$) are in *OPEN*, D is chosen for expansion. Thus, for this range of W, P_{OPT} will be returned whether or not reopening is done. This is shown in the left part of Table 1. Hence we label this range as a $P^=$ range (shown in the P column).

Since states X_i will be expanded before D, both AR and NR will expand them; this is indicated in the X_i column. The *AR* and *NR* columns give the total number of nodes expanded by these policies. There are two subranges here, shown in the right part of Table 1, regarding node C.

(1a:) $\mathbf{1 \leq W < 3}$: Here W is so small that B is expanded before C. Thus, C is seen again via P_2 while it is still in *OPEN*, before it was ever expanded. The dilemma of whether to reopen C will not even occur. In this subrange, both AR and NR are identical in both aspects (path length and nodes expanded). Hence, this subrange is designated by $P^=N^=$.

(1b:) $\mathbf{3 \leq W < 3.75}$: If reopening is done, states C and E will be expanded twice, making AR lose on the nodes expanded count (see the AR and NR columns in Table 1). Hence, this subrange is designated by $P^=N^-$.

(Range 2:) $\mathbf{3.75 \leq W < 4.25}$**:** This case is very interesting as: $22 = f_{P_2}(G) \leq f(D) < f_{P_1}(G) = 24$. Therefore, if C is not reopened, $f(G)$ remains 24 and D will reach the top of *OPEN*. It will then be expanded and P_{OPT} will be returned. But, if C is reopened, $f(G)$ becomes 22 (via P_2). $f(G)$ is now $\leq f(D)$; D will not be expanded and P_2 will be returned. This range is an example where AR results in a longer path than NR.

In subrange **(2a)**, where $\mathbf{3.75 \leq W < 4}$, $f(X_i) < 22$. With both policies, states X_i are expanded before G, as even for AR $f_{P_2}(G) = 22$. However, AR expands more states since it expands E twice. Here NR is winner in both aspects!, i.e., this case is designated by P^-N^-.

In subrange **(2b)**, where $\mathbf{4 \leq W < 4.25}$, if C is not reopened P_{OPT} will be returned via D. However, the states X_i will be expanded before D. But, if C is reopened, $f(G)$ will be updated to 22 and G will be expanded before states X_i and D. Therefore, although NR returns a shorter path, it expands more states than AR. This case is designated by P^-N^+ – the exact opposite of the intuitive case we showed above.

(Range 3:) $\mathbf{4.25 \leq W < 5}$**:** In this range $f(D) \geq 24$. $f(D)$ is now larger than both $f_{P_1}(G)$ and $f_{P_2}(G)$. Thus, D will never be chosen for expansion. So, if C is not reopened, P_1 will be returned. But, if C is reopened, P_2 will be returned. This is the intuitive case reported above where reopening revealed a better path via C. This range is labeled as P^+. There are again two subranges.

(3a:) $\mathbf{4.25 \leq W < 4.5}$: Here $22 < f(X_i) < 24$. Therefore, if C is reopened, then $f(G) = 22 < f(X_i)$ and G will be expanded immediately without expanding X_i. However, if C is not reopened, states X_i will be expanded before G as $f_{P_1}(G) = 24$. This case is designated as P^+N^+. AR wins here in both aspects.

(3b:) $\mathbf{4.5 \leq W < 5}$: In this case $f(X_i) \geq 24$. Therefore, neither AR nor NR expand states X_i (since $f(G) \leq 24$ and therefore G is expanded first). This is the intuitive case shown above where reopening revealed a better path via C but expanded more states (since AR expands C, E and G twice), i.e., P^+N^-.

(Range 4:) $\mathbf{W \geq 5}$**:** Here $f(D) \geq 7 + 5 \cdot 4 = 27$. Thus, D will never be chosen for expansion. Furthermore, here, $f(B) \geq 24$. Thus, B will never be chosen for expansion and C will not even be generated again. In this case, again, we will never even have the dilemma of whether to reopen C or not. Hence, both AR and NR will return P_1 and expand

		Path returned			Nodes Expansions					
Range	W	AR	NR	P	Subrange	Case	W	X_i	AR	NR
1	$1 \leq W < 3.75$	P_{OPT}	P_{OPT}	$P^{=}$	1a	$P^{=}N^{=}$	$1 \leq W < 3$	both	11	11
					1b	$P^{=}N^{-}$	$3 \leq W < 3.75$	both	13	11
2	$3.75 \leq W < 4.25$	P_2	P_{OPT}	P^{-}	2a	$P^{-}N^{+}$	$3.75 \leq W < 4$	NR	8	11
					2b	$P^{-}N^{-}$	$4 \leq W < 4.25$	both	12	11
3	$4.25 \leq W < 5$	P_2	P_1	P^{+}	3a	$P^{+}N^{+}$	$4.25 \leq W < 4.75$	NR	8	10
					3b	$P^{+}N^{-}$	$4.75 \leq W < 5$	None	8	6
4	$5 \leq W$	P_1	P_1	$P^{=}$	4	$P^{=}N^{=}$	$5 \leq W$	None	5	5

Table 1: The path and expansions for different values of W in Figure 1

exactly the nodes in P_1 (hence $P^{=}N^{=}$).

Table 1 summarizes the different ranges (left) and subranges (right). For very large values (range 4), reopening will never happen, hence both policies tie. For very small values (range 1), reopening will make no difference as the optimal path will be returned anyway – again a tie. The interesting cases are in the middle range. For range 3, AR will result in a better path. This is the intuitive case where AR is expected to be better, or at least no worse than NR (if ranges 1 and 4 are considered too). The somewhat counter-intuitive case is range 2, where AR finds a better path (P_2) and halts while the path that NR has (P_1) is not strong enough, forcing NR to continue searching and eventually finding the optimal path (P_{OPT}). As can be seen, in our example graph 7 out of the 9 possible cases are present.

Experiments

Our example graph demonstrated that different behaviors can occur on the same graph while only varying the value of W. Next, we show experimentally that different trends also occur in two benchmark domains: the sliding tile puzzle and grid-world pathfinding.

Tile Puzzle

We experimented with the 3x3 8-puzzle, and the 4x4 15-puzzle with the Manhattan distance heuristic. For each puzzle we generated 1000 random instances and used W values 1.5, 2, 3, 5, 10, 20, 30, 50, 100, 200, 500, and 1,000.

Figure 2 shows the percentage of 15-puzzle instances, partitioned into five disjoint groups according to the cost of the found path. The figure highlights the "anomalies", or the *counter intuitive* cases – when NR finds a *strictly* better path than AR (P^{-}) and when AR expands *strictly* fewer nodes than NR (N^{+}).

The two topmost slices show the case where no anomaly was seen (includes the range $P^{+=}N^{-=}$). This zone is split into the topmost slice (gray), P^{+} and the second slice (light blue) where there was a tie in the path returned ($P^{=}$). It is interesting to note that around 50% of the instances fall into the $P^{=}$ slice for all values of W.

The lower three slices correspond to cases with "anomalies", when we had either P^{-} or N^{+} (or both). The $P^{+=}N^{+}$ slice (green) shows the case where AR strictly dominates NR, expanding fewer nodes and finding solutions that are at least as good as NR. Similarly, the $P^{-}N^{-=}$ slice (red) shows the other extreme case where NR strictly dominates AR, finding better solutions without expanding more nodes. Our results show that for some values of W these full-dominance cases occurs in a non-negligible percentage of the instances. This calls for a method for predicting which reopening policy to use. Finally, the $P^{-}N^{+}$ slice (purple) represents the counter intuitive case where NR returned a better path at the tradeoff of more nodes expanded.

As can be seen, the "anomalies" are less frequent for very low or very high W values. However, for mid-range W values they occur very often. For example, for $W = 3$, almost 40% of the random instances we generated had either P^{-} or N^{+}; 10% had both ($P^{-}N^{+}$). When omitting the light blue slice ($P^{=}$), one can see that for many W values, anomalies occur more often than the case that strictly match the intuitive case $P^{+}N^{-=}$.

We obtained similar results for the 8-puzzle, but the "anomaly" cases were less frequent. Our conjuncture is that this is because the 8-puzzle is a smaller domain, having shorter solution lengths and as a result less prone to the generation of these "anomalies".

Grid-based maps

Our next domain is pathfinding in a 4-connected grid. We used the Manhattan distance heuristic, and experimented on three grids from the video game *Dragon Age: Origins* from the publicly available repository of (Sturtevant 2012). The three grids, `brc202d`, `ost003d` and `den520d` differ in their density and number of obstacles. `brc202d` has narrow corridors and no open spaces. `ost003d` has more open spaces bounded by 'rooms' and finally, `den520d` has no corridors or rooms but have many open spaces. For each map we generated 100 different instances by choosing a ran-

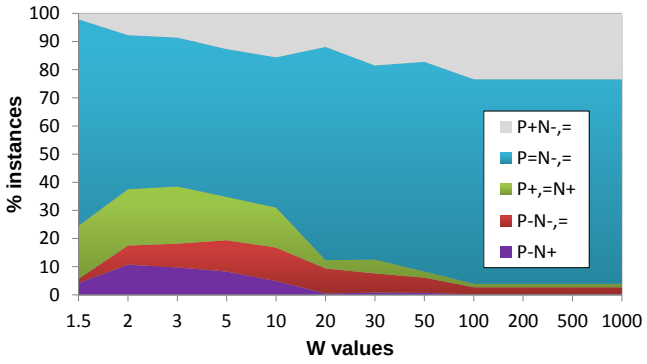


Figure 2: % instances per class, 15-puzzle

P	N	1.1	1.2	2	3	5	Image
brc202d							
P^+	$N^{-=}$	76	81	95	94	90	
$P^=$	$N^{-=}$	12	11	4	5	10	
$P^+ =$	N^+	10	0	0	0	0	
P^-	$N^{-=}$	2	8	1	1	0	
P^-	N^+	0	0	0	0	0	
ost003d							
P^+	$N^{-=}$	37	40	80	83	58	
$P^=$	$N^{-=}$	30	37	8	12	38	
$P^+ =$	N^+	26	15	5	3	3	
P^-	$N^{-=}$	5	5	7	2	1	
P^-	N^+	2	3	0	0	0	
den520d							
P^+	$N^{-=}$	19	23	46	40	42	
$P^=$	$N^{-=}$	57	48	39	50	54	
$P^+ =$	N^+	14	14	9	7	4	
P^-	$N^{-=}$	7	15	6	3	0	
P^-	N^+	3	0	0	0	0	

Table 2: % of instances per class for grid experiments

dom start and goal locations on the map and assuring the shortest path between them is at least $\frac{2}{3}$ of the grid side size. We used the same range of W values as in the tile puzzle and added $W = 1.1$ and $W = 1.2$.

Table 2 shows a partition of the map instances based on the same P^*N^* combinations as Figure 2. Results are only given for $W \leq 5$, as beyond that point all instances were part of the $P^{+=}N^{-=}$ case. We observe two trends that are similar to the tile puzzle experiments above. First, for these maps as well, we see that the "anomalies" P^- and N^+ (bottom three rows) make up to $\frac{1}{3}$ of the cases for some values of W (e.g., for $W = 1.1$ for `ost003d`). Second, the percentage of both P^- and N^+ initially grows with W until reaching some "maximum" point after which the P^- and N^+ percentage drops eventually to zero.

We note that the different maps exhibit different behavior. The more maze-like map (`brc202d`) had only a few instances from either P^- or N^+. In the more open-spaced maps, (`ost003d` and `den520d`) more cases with "anomalies" were seen. In some cases, e.g., in `den520d` for $W = 1.1$ and $W = 1.2$, "anomalies" occur even more often than the top row which strictly matches the intuitive case. We conjecture that this is because maze-like maps have lower frequency of multiple paths to the goal, and therefore less reopening occurs in general. To support this claim, we also ran experiments on mazes with varying corridor width (namely, 1, 2, 4, 8 and 16), also taken from the same benchmark repository (Sturtevant 2012). In these mazes, we did not observe any P^- instances, and very few N^+ instances.

Better polices

AR and NR are extreme policies – either *always* or *never* reopen nodes. One can implement a hybrid policy that chooses to reopen some nodes while bypassing reopening in other nodes. In the future, we intend to develop such policies. However, to demonstrate the potential of such hybrid policies we implemented a *reopening oracle*, as follows. Given an instance to be solved by WA*, for every closed state which is generated again with a smaller g-value, a decision needs to be made whether to reopen it or not. By marking positive decision (to reopen) by 1 and negative decision (not to reopen) by 0, a sequence of reopenings choices can be described as a binary vector. There are two types of reopening oracles: *min-cost oracle* and *min-expansions oracle*. A min-cost oracle simulates the decisions imposed by *all* the possible binary vectors and then returns the sequence of reopening choices that results in finding a solution of lowest cost. A min-expansions oracle chooses the sequence of reopening choices that results in finding a solution while expanding a minimal number of nodes.

If the number of decision points is p an oracle needs to simulate 2^p sequences. For computational complexity reasons, we implemented *limited* versions of the min-cost and min-expansions oracles, such that they only consider the first r reopening choices, where r is a parameter, after which they degrade back to either AR or to NR (which ever minimizes the cost/expansions better). We implemented these limited min-cost and min-expansions oracles with $r = 9$ and run them on 100 random instances of the 8-puzzle with weights 1.5, 2, 3, 5, 10, 20 and 30. For each instance and weight we compared the results of AR, NR to our limited min-cost oracle and to our limited min-expansions oracle. Our results showed that the limited min-cost oracle found shorter solutions of length which was up to 17% shorter (i.e., a factor of 0.83) than the best solution found by best extreme policy. Similarly, we have found instances where the sequence returned by the limited min-expansions oracle expanded only 0.6 of the nodes expanded by the best extreme policies. This experiment was performed on a limited series of $r = 9$ and on a rather small domain (8-puzzle). Nevertheless, it demonstrates the potential benefit of further pursuing intelligent, hybrid reopening policies.

Discussion and future work

The intuitive case is that avoiding reopening nodes in WA* will decrease runtime at the expense of finding better solutions. We classified all the other cases and demonstrated that their occurrence is non negligible. We illustrated a small graph where, depending on the value of W, NR may find better solutions than AR, and on the other hand NR might expand more nodes than AR. In practice, we demonstrated experimentally on two standard benchmarks that these anomalies occur for many W values. In addition, we conjunctured about the relation between certain domain properties and the frequency of these "anomalies" occurring.

This is a work in progress. Based on our oracle experiments our research now focuses on finding reopening policies smarter than the extreme AR and NR. Such policies probably should use the current state of the search (e.g. the g and h value of the state being reopened, the current number of expansions etc.) and try to predict at each decision point, whether reopening should be done.

In addition, we plan to inspect other scenarios where reopening occurs such as enhanced versions of WA*, more advanced bounded-suboptimal search algorithms (Thayer and Ruml 2008; 2009) and A* with inconsistent heuristics (Felner et al. 2011).

References

Ebendt, R., and Drechsler, R. 2009. Weighted A* search – unifying view and application. *Artif. Intell.* 173(14):1310–1342.

Felner, A.; Zahavi, U.; Holte, R.; Schaeffer, J.; Sturtevant, N. R.; and Zhang, Z. 2011. Inconsistent heuristics in theory and practice. *Artif. Intell.* 175(9–10):1570–1603.

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

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

Likhachev, M.; Gordon, G. J.; and Thrun, S. 2003. ARA*: Anytime A* with provable bounds on sub-optimality. In *NIPS*. MIT Press.

Malima, A., and Sabanovic, A. 2007. Motion planning and assembly for microassembly workstation. In *Proceedings of the 10th IASTED International Conference on Intelligent Systems and Control*, 467–474.

Pearl, J., and Kim, J. H. 1982. Studies in semi-admissible heuristics. *IEEE Trans. Pattern Anal. Mach. Intell.* 4(4):392–399.

Pohl, I. 1970. Heuristic search viewed as path finding in a graph. *Artificial Intelligence* 1(3-4):193–204.

Pohl, I. 1973. The avoidance of (relative) catastrophe, heuristic competence, genuine dynamic weighting and computational issues in heuristic problem solving. In *Proceedings of the 3rd international joint conference on Artificial intelligence*, 12–17.

Sturtevant, N. 2012. Benchmarks for grid-based pathfinding. *Transactions on Computational Intelligence and AI in Games*.

Thayer, J. T., and Ruml, W. 2008. Faster than weighted A*: An optimistic approach to bounded suboptimal search. In *ICAPS*.

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

Thayer, J. T., and Ruml, W. 2010. Anytime heuristic search: Frameworks and algorithms. In *SOCS*.

Wilt, C. M., and Ruml, W. 2012. When does weighted A* fail? In *SOCS*.

Delete Relaxation and Traps in General Two-Player Zero-Sum Games

Thorsten Rauber and **Denis Müller** and **Peter Kissmann** and **Jörg Hoffmann**
Saarland University, Saarbrücken, Germany
{s9thraub, s9demue2}@stud.uni-saarland.de, {kissmann, hoffmann}@cs.uni-saarland.de

Abstract

General game playing (GGP) is concerned with constructing players that can handle any game describable in a pre-defined language reasonably well. Nowadays, the most common approach is to make use of simulation based players using UCT. In this paper we consider the alternative, i.e., an Alpha-Beta based player. In planning, delete relaxation heuristics have been very successful for guiding the search toward the goal state. Here we propose evaluation functions based on delete relaxation for two-player zero-sum games.

In recent years it has been noted that UCT cannot easily cope with shallow traps, while an Alpha-Beta search should be able to detect them. Thus, a question that arises is how common such traps are in typical GGP benchmarks. An empirical analysis suggests that both cases, relatively few traps and a high density of traps, can occur. In a second set of experiments we tackle how well the Alpha-Beta based player using the proposed evaluation function fares against a UCT based player in these benchmarks. The results suggest that (a) in most games with many traps Alpha-Beta outperforms UCT, (b) in games with few traps both players can be on par, (c) the evaluation functions provide an advantage over a blind heuristic in a number of the evaluated games.

Introduction

Game playing has always been an important topic in artificial intelligence. The most well-known achievements are likely the successes of specialized game players such as DeepBlue (Campbell, Hoane, and Hsu 2002) in Chess or Chinook (Schaeffer et al. 1992) in Checkers, defeating the human world-champions in the respective games. However, these specialized players have deviated far from the original idea of a general problem solver (Newell and Simon 1963).

In 2005 this idea was picked up again, by introducing a new competition for promoting research in general game playing (Genesereth, Love, and Pell 2005). Here the players are not supposed to play only a single game on world-class level, but rather to be able to handle any game that can be described in a given language and play it reasonably well. Most research in this area has been invested in deterministic games of full information.

After early successes of players based on Alpha-Beta search with automatically generated evaluation functions (e.g., (Clune 2007; Schiffel and Thielscher 2007)), a new trend dominates the field: the use of UCT (Kocsis and Szepesvári 2006). This is a simulation-based approach, i.e., lots of games are simulated and the achieved rewards propagated toward the root of a partial game-tree, in order to decide on the best move to take. Since 2007 all winners of the international competition have made use of this technique (e.g., (Björnsson and Finnsson 2009; Méhat and Cazenave 2011)). However, for certain games it has been shown that UCT is not always the best choice. One property that is difficult to handle by that approach is the presence of shallow traps (Ramanujan, Sabharwal, and Selman 2010), i.e., states from which the opponent has a winning strategy of short length. While Alpha-Beta can identify such traps, UCT typically can not, at least if the branching factor is high enough or the possible playouts within the trap are long enough.

The basic setting of general game playing is comparable to that of action planning. There the aim also is to implement solvers that can handle any planning task describable in the given language. The current trend in planning is in heuristic search, where the heuristics are automatically generated at run-time. One successful approach is based on delete relaxation, e.g., the FF heuristic (Hoffmann and Nebel 2001). In the delete relaxed setting, anything that once was true remains true. The length of a plan (i.e., a solution) for a delete relaxed planning task can then be used as an estimate for the length of a plan in the original setting.

In this paper we propose new evaluation functions for general game playing based on the length estimates derived by delete relaxation heuristics and apply these evaluation functions in an Alpha-Beta implementation. Furthermore, we empirically evaluate a number of games to get an idea of their trap density. In the experimental results we will see that our Alpha-Beta based player indeed outperforms a UCT based player in most tasks that contain a large amount of shallow traps and is on-par in several of the games with fewer traps. Additionally, the use of the evaluation function brings a real advantage over a blind heuristic in a number of the evaluated games.

Background

In this section we provide the necessary background on general game playing, UCT search, traps in games and delete relaxation heuristics as they are used in planning. We assume the reader to be familiar with the basics of Alpha-Beta search, so that we skip an introduction.

General Game Playing

The main idea of general game playing (GGP) is to implement players that play any game that can be described by the given language reasonably well. The current setting as it was introduced for the first international competition in 2005 (Genesereth, Love, and Pell 2005) allows for a wide range of games: single-player puzzles or two- and multi-player games, which can be, among others, turn-taking or with simultaneous moves, zero-sum or more general rewards, cooperative, etc. In all settings the goal for each player is to maximize its own reward. Furthermore, all these games are finite, discrete, deterministic, and all players have full information.

In this paper we consider only the case of strictly turn-taking two-player zero-sum games. The two players are denoted Max (the starting player) and Min. As possible outcomes we allow only win (here denoted 1), loss (denoted -1), and draw (denoted 0) from the Max player's point of view. Basically, our definition of a game is an extension of the multi-agent STRIPS setting (Brafman and Domshlak 2008) to adversarial agents:

Definition 1. *A game is a tuple* $\Pi_G = \langle V, A_{Max}, A_{Min}, I, G, R\rangle$*, where* V *is the set of state variables or facts,* A_{Max} *and* A_{Min} *are the actions of the Max and Min player, respectively,* I *is the initial state in form of a complete assignment to* V*,* G *is the termination criterion in form of a partial assignment to* V*, and* R *is a function assigning a reward in* $\{-1, 0, 1\}$ *to each terminal state. Similar to planning, an action* a *is of the form* $\langle pre_a, add_a, del_a\rangle$*, where* pre_a *is the precondition,* add_a *the list of add-effects, and* del_a *the list of delete-effects.*

Note that this definition deviates from the commonly used game description language GDL (Love, Hinrichs, and Genesereth 2008), where the effects of an action specify all facts that are true in the successor state, which together with the closed world assumption results in a full state specification.

Definition 2. *For a game* $\Pi_G = \langle V, A_{Max}, A_{Min}, I, G, R\rangle$*, the semantics are defined by means of a transition system* $\Theta_G = \langle S, L, T, I, S_g^{-1}, S_g^0, S_g^1\rangle$*, where* $S = S_{Max} \cup S_{Min}$ *is the finite set of all states,* S_{Max} *the set of states where Max has to move and* S_{Min} *the set of states where Min has to move.* $L = L_{Max} \cup L_{Min}$ *is a finite set of labels with* $L_{Max} = A_{Max}$ *and* $L_{Min} = A_{Min}$*.* $T = T_{Max} \cup T_{Min}$ *is a set of transitions consisting of* $T_{Max} \subseteq S_{Max} \times L_{Max} \times S_{Min}$ *and* $T_{Min} \subseteq S_{Min} \times L_{Min} \times S_{Max}$*. Precisely,* $(s, l, s') \in T_{Max}$ *if* l *is applicable in* s *(i.e., if* $s \in S_{Max}$*,* $l \in L_{Max}$*,* $pre_l \subseteq s$*), and* $s' \in S_{Min}$ *is the resulting successor state (i.e., if* $s' = (s \setminus del_a) \cup add_a$*); similar for* T_{Min}*.* $I \in S_{Max}$ *is the initial state.* $S_g^{-1} \subseteq S$ *is the set of terminal states lost for Max,* $S_g^0 \subseteq S$ *the set of terminal draw states, and* $S_g^1 \subseteq S$ *the set of terminal states won by Max, i.e.,* $s \in S_g^r$ *if* $G \subseteq s$ *and* $R(s) = r$*. The Max player tries to maximize the reward while Min tries to minimize it.*

Upper Confidence Bounds Applied to Trees

The upper confidence bounds (UCB1) algorithm (Auer, Cesa-Bianchi, and Fischer 2002) is used in the area of multi-armed bandits and aims at maximizing the expected reward. The upper confidence bounds applied to trees (UCT) algorithm (Kocsis and Szepesvári 2006) is an extension of this to tree based searches. It treats every internal node as a multi-armed bandit (where the different arms correspond to the possible actions to take) and tries to learn which actions are preferable. UCT consists of four phases: selection, expansion, simulation, and backpropagation.

In the selection phase nodes stored in the UCT tree are evaluated and a path is followed until a leaf of that tree is reached. The evaluation works as follows. Let s be the state represented by a node, $a_1, \ldots, a_n$ the actions applicable in state s, $s_1, \ldots, s_n$ the corresponding successor states, $n(s)$ the number of times state s was reached, $n(s, a)$ the number of times that action a was chosen in state s, and $\delta(s)$ the average reward achieved when starting in state s. The UCT value for the different actions is defined as

$$UCT(s, a_i) = \delta(s_i) + C\sqrt{\frac{\log n(s)}{n(s, a_i)}}\,. \qquad (1)$$

In our two-player setting, if $s \in S_{Max}$, we can use this directly and select the action achieving the highest UCT value; if $s \in S_{Min}$, we use the negation of $\delta(s_i)$ in equation (1) and still select the action achieving the highest UCT value. The constant C is used to set the amount of exploration or exploitation: With a small value the algorithm will tend to exploit the knowledge already generated by mainly following the most promising actions, while with a high value the algorithm will tend to explore different areas, often selecting actions that lead to less promising successors.

If a state contains some unexplored successors, instead of evaluating the UCT formula one of the unexplored successors will be selected randomly. This assures that the formula is evaluated only if initial values for all successors are set.

The expansion phase starts when a leaf node of the UCT tree has been reached. In that case the leaf node will be expanded, the successor added to the tree, and the simulation phase starts.

In the simulation phase a Monte-Carlo run is started, which chooses among the applicable actions of the current state at random until a terminal state is reached.

When this happens, the backpropagation starts. This updates the average rewards and counters of all states visited during the selection phase. As soon as all nodes are updated the selection phase starts over at the root of the UCT tree.

When an actual action is to be performed and we are the Max (Min) player, the action leading to the successor with highest (smallest) average will be selected and the corresponding successor node will become the new root of the UCT tree. If it is not our turn to move in the current state, we wait for the action chosen by the opponent and take the corresponding successor as the new root. Afterwards the search starts over at the new root node.

Traps in Games

In some games such as Go traditional Alpha-Beta based players are clearly inferior to UCT based players like MoGo (Gelly and Silver 2008). In others like Chess, however, the Alpha-Beta based players are on a level beyond any human

world-champion and clearly outperform UCT based players. One explanation for this behavior was provided by Ramanujan, Sabharwal, and Selman (2010), who noticed that Chess contains shallow traps while Go does not.

A state s is called at risk if for the current player p there is a move m so that the corresponding successor s' is a state in which the opponent of p has a winning strategy. If the winning strategy has a maximum of k moves, we call state s' a level-k search trap for p.

A trap is considered to be shallow if it can be identified by Alpha-Beta search. Due to its exhaustive nature, this is the case if its depth-limit is at least $k + 1$. Such a player can avoid falling into the trap by using a move different from m in state s. Typically, traps of level 3–7 are considered to be shallow. UCT often cannot identify such traps as it spends a lot of time exploring states much deeper than the level of the trap, in areas it considers promising.

In a subsequent study Ramanujan, Sabharwal, and Selman (2011) created a synthetic game in which they could manually set the density of traps. With this they found that without any traps, UCT was much better than Minimax. With only few traps in the state space UCT was still better than Minimax. However, the higher the density of traps the worse UCT performed in comparison to Minimax.

Delete Relaxation Heuristics

The setting in planning is similar to ours, with the exception that planning allows only for a single agent. Additionally, the aim of this agent is to reach a terminal state in as few steps as possible. Other than that, especially the handling of actions is the same in both formalisms.

The delete relaxation corresponds to the idea of ignoring the delete effects. That means, everything that once was true will remain true forever. This allows the calculation of a fixpoint of those facts that can become true at any point in the future and of those actions that may be applicable at some point in the future.

One way to do so is by means of the relaxed planning graph (RPG). The RPG consists of alternating layers of facts and actions. The first layer contains all those facts currently true. Then follows a layer with all the actions that are applicable based on those facts. The next layer contains all facts true in the previous layers and the ones added by the actions of the previous layer. This continues until a fixpoint is reached or all facts of a specified goal state are present in the last generated layer. Instead of generating the full RPG, it often suffices to store, for each action and each fact, the first layer it appeared in.

The optimal relaxation heuristic h^+ gives the minimal number of actions needed to reach a given goal state from the current state in the relaxed setting, which is an admissible (i.e., not overestimating) heuristic for the original non-relaxed search problem. As this is NP-hard to calculate approximations are used, e.g., the FF heuristic (Hoffmann and Nebel 2001). After generating the RPG, it marks all facts of the specified goal. Then it works in a backpropagation manner through the RPG, starting at the last generated layer. For each marked fact newly added in this layer it marks an action that adds it. Given these newly marked actions it marks all facts in their preconditions. This continues until the first layer of the RPG is reached. At that point, the marked actions correspond to a solution plan of the relaxed task, and their number is returned as an approximation of the h^+ heuristic.

Delete Relaxation in GGP

In order to evaluate non-terminal states we propose the following new approach based on delete relaxation heuristics. Similar to automated planning, we can define the delete relaxation of a game:

Definition 3. *For a game $\Pi_G = \langle V, A_{Max}, A_{Min}, I, G, R\rangle$, we denote its delete relaxation as $\Pi_G^+ = \langle V, A_{Max}^+, A_{Min}^+, I, G, R\rangle$ where $A_{Max}^+ = \{\langle pre_a, add_a, \emptyset\rangle \mid \langle pre_a, add_a, del_a\rangle \in A_{Max}\}$ (similar for A_{Min}^+).*

Given a state s, we use the FF heuristic (Hoffmann and Nebel 2001) operating on the full set of actions $A^+ = A_{Max}^+ \cup A_{Min}^+$ to estimate the number of moves needed to reach a state with reward 1, denoted as $l_{win}(s)$, and to estimate the number of moves needed to reach a state with reward -1, denoted as $l_{lose}(s)$. Each of these values is set to ∞ if no corresponding terminal state is reachable anymore. We define the evaluation function $h_1(s)$ of state s as

$$h_1(s) = \begin{cases} 1 & \text{if } l_{win}(s) \neq \infty \text{ and } l_{lose}(s) = \infty \\ -1 & \text{if } l_{win}(s) = \infty \text{ and } l_{lose}(s) \neq \infty \\ 0 & \text{if } l_{win}(s) = \infty \text{ and } l_{lose}(s) = \infty \\ \frac{l_{lose}(s) - l_{win}(s)}{\max(l_{lose}(s), l_{win}(s))} & \text{otherwise.} \end{cases}$$

If only one player's winning goal states cannot be reached anymore we treat the state as being won by the opponent. Otherwise the quotient results in a value in $[-1, 1]$. If it takes more moves to reach a lost state the Max player seems to have a higher chance to win, so that the evaluation will be greater than 0; otherwise the Min player seems to have a better chance, resulting in a value smaller than 0.

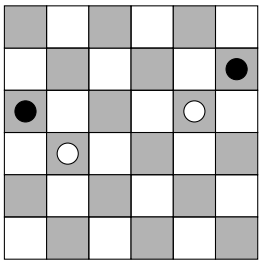


Figure 1: Example state of the game Breakthrough.

Example 1. *As an example we take the game Breakthrough. In this game, white starts with two rows of pawns at the bottom and black with two rows of pawns at the top. The pawns may only be moved forward, vertically or diagonally, and can capture the opponent's pawns diagonally. The goal of each player is to reach the other side of the board with one of their pawns. Consider the state given in Figure 1. Assume that white is the Max player and black the Min player, and in the current state it is white's turn to move.*

In the following we will evaluate the states reached by applying the two different capturing moves. The first one captures with the most advanced white pawn, resulting in state s_1, the second one captures with the least advanced white pawn, resulting in state s_2. In s_1 white needs at least one more move, while black needs at least 3 more moves, so that $h_1(s_1) = (3-1)/3 = 0.67$, indicating an advantage for the white player. In s_2 the white player will need at least two moves and the black player at least four moves to win the game, so that $h_1(s_2) = (4-2)/4 = 0.5$, which indicates a smaller advantage for the white player compared to s_1.

An alternative is to take the mobility into account. A previous approach (Clune 2007) used the mobility directly: The author compared the number of moves of both players, normalized over the maximum of possible moves of both players. While this seems to work well in several games, it bears the danger of sacrificing own pieces in games like Checkers where capturing is mandatory: In such games, bringing the mobility of the opponent to as small a value as possible typically means restricting the opponent to a capture move.

Thus, we do not inspect the mobility in the current state but rather try to identify how many moves remain relevant for achieving a goal. In order to do so we first calculate a full fixpoint of the RPG, i.e., we generate the RPG until no new facts or no new actions are added to it. Only the actions in that graph can become applicable at some time in the future. Now, starting at the facts describing a player's won states, we perform backward search in the RPG, identifying all actions of that player that may lie on a path to those facts. These actions we call relevant.

Let $n_{Max,rel}(s)$ be the number of relevant actions of the Max player in state s and $n_{Min,rel}(s)$ the number of relevant actions of the Min player in state s. Similarly, let n_{Max} be the total number of actions of the Max player and n_{Min} the total number of actions of the Min player. Then we define another evaluation function h_2 for state s as follows:

$$h_2(s) = \frac{n_{Max,rel}(s)}{n_{Max}} - \frac{n_{Min,rel}(s)}{n_{Min}}$$

This assumes that a player has a higher chance of winning if the fraction of still relevant actions is higher for this player than for the opponent.

Example 2. *Consider again the Breakthrough example from Figure 1 and the two successor states s_1 and s_2. In Breakthrough, all moves advance a pawn to the opponent's back row, so that any move that can still be performed at some point is relevant. We distinguish between the (left/right) border cells, where the players have two possible moves, and the inner cells, where the players have three possible moves. Initially, both players have 80 relevant actions (both can reach 10 border cells and 20 inner cells). In s_1, the white player can reach three border cells, and five inner cells, resulting in a total of 21 relevant moves. The black player can still reach three border cells and three inner cells, resulting in a total of 15 relevant moves. Thus, $h_2(s_1) = 21/80 - 15/80 = 6/80$, giving a slight advantage for white. In s_2, the white player can reach three border cells and four inner cells, resulting in a total of 18 relevant moves. The black player can still reach four border cells and six inner cells, resulting in a total of 26 relevant moves. Thus, $h_2(s_2) = 18/80 - 26/80 = -8/80$, indicating a slight advantage for black.*

As a third option, we combine these two evaluation functions to a new function $h_{1+2}(s) = w_1 h_1(s) + w_2 h_2(s)$. Learning weights w_1 and w_2, especially ones optimized for the game at hand, remains future work; in this paper we use a uniform distribution, i.e., $w_1 = w_2 = 0.5$.

Implementation Details

We implemented an Alpha-Beta based player and a UCT based player as well as the new evaluation functions on top of the FF planning system (Hoffmann and Nebel 2001). In this section, we provide some details on the extensions over basic Alpha-Beta and UCT players that we additionally implemented.

Alpha-Beta

In our Alpha-Beta implementation we make use of iterative deepening (Korf 1985), searching to a fixed depth in each iteration and evaluating the non-terminal leaf nodes based on our evaluation function. Between iterations we increase the depth limit by one.

In addition we implemented several extensions found in the literature, among them the use of a transposition table and approaches to order the moves based on results in the transposition table and from previous iterations, which is supposed to result in stronger pruning.

As soon as the evaluation time is up and the player must decide which action to take, the current iteration stops. If it is not this player's turn, nothing has to be done. Otherwise, in case of being the Max (Min) player the action leading to the successor with highest (smallest) value is chosen. The successor reached by the chosen action is taken as the new root of the graph and Alpha-Beta continues.

Quiescence Search On top of this we implemented quiescence search, which tries to circumvent the so called horizon effect. The basic idea is to distinguish between noisy and quiet states, where a state is considered to be noisy in case of tremendous changes in the game with respect to the previous state. As soon as our normal Alpha-Beta search reaches the depth limit we check whether the current state is noisy, and if so, we will switch into quiescence search and continue until we reach a terminal state, a quiet state, or the predefined depth limit of quiescence search.

Our idea for deciding if a state is noisy is to check if the number of moves has drastically changed. Thus, we defined and tested the following criteria:

Applicable actions In each state compute, for the current player, the number of the currently applicable actions and compare it to the value of the previous state where it was this player's turn.

Possible actions In each state compute, for the current player, the number of actions that might still be possible to take later in the game (found by building the RPG

fixpoint, similar to our evaluation function h_2) and compare it to the value of the previous state where it was this player's turn.

In preliminary tests we found that the applicable actions criterion works better than the possible actions criterion. An explanation for this is the overhead induced by computing the RPG fixpoint.

For deciding if a state is noisy, apart from the criterion to check we also need a threshold for deciding if that change corresponds to a noisy state. If the change in the corresponding criterion is greater than the given threshold we consider it to be noisy. We tested threshold values between 5% and 50% and came to the conclusion that 30% is the best value wrt. the applicable actions criterion.

While for some games other values would be better, recall that we consider domain-independent approaches here, so that we cannot choose the most appropriate value for each game in advance. It remains future work to find intelligent ways for adapting this value at run-time depending on the properties of the currently played game.

UCT

For UCT, instead of a tree we generate a graph by using a hash function, similar to the transposition table in Alpha-Beta. In the expansion phase, if we generate a successor state that is already stored in the hash table we take the corresponding existing search node as the child node. While some implementations might make use of parent pointers, thus effectively updating nodes not really visited, we propagate the reached results only along the path of actually visited nodes.

Another extension concerns the use of a Minimax-like scheme in the UCT graph. Similar to the approach proposed by Winands and Björnsson (2011), we mark a node in the UCT graph as solved if it corresponds to a terminal state. During the backpropagation phase we check for every encountered node whether all successors have already been solved. If that is the case, we can mark this node as solved as well and set its value in the Minimax fashion based on the values of its successors. Furthermore, if we are in control in a node and at least one successor is marked as solved and results in a win for us, we mark this node as solved as well and set its value to a win for us. In the selection phase we go through the UCT graph as usual, but stop at solved states and can start the backpropagation phase immediately. Overall, this approach is supposed to bring us the advantage that the values converge much faster and that the runs can become shorter and only within the UCT graph, which prevents the numerous expansions in the simulation phase.

Experimental Results

In this section we start by describing the games we considered in our experiments. Next we point out some insights on traps in those games, along with an empirical evaluation of the trap densities. Finally, we present results of running our Alpha-Beta versions against UCT on that set of games.

Benchmark Games

In the following we will outline the games we used in our experiments.

Breakthrough consists of a Chess-like board, where the two rows closest to a player are fully filled with pawns of their color. The moves of the pawns are similar to Chess, with the exception that they can always move diagonally. The goal is to bring one pawn to the opponent's side of the board or to capture all opponent's pawns.

Chomp consists of a bar of chocolate with the piece in the bottom left corner being poisoned. The moves of the players are to bite at a specified position that still holds a piece of chocolate. The result is that all pieces to the top-right of this are eaten. The player eating the poisoned piece loses the game.

Chinese Checkers is normally played on a star-like grid. In the two-player version we can omit the home bases of the other four players, so that the board becomes diamond-shaped. In each move the players may only move forward (or do nothing), and perform single or double jumps. Each player has three pieces and must move them to the other side of the board, consisting of 5×5 cells. If no player is able to do so in 40 moves the game ends in a draw.

Clobber is played on a rectangular board. The pieces are initially placed alternatingly, filling the entire board. A move consists of moving a piece of the own color to a (horizontally or vertically) adjacent cell with a piece of the opponent's color on it. That piece is captured and replaced by the moved piece of the active player. The last player able to perform a move wins.

Connect Four is a classical child's game. The players take turns putting a piece of their color in one of the columns, where it falls as far to the bottom as possible. The goal is to achieve a line of four pieces of the own color. If the board gets fully filled without one player winning, the game ends in a draw.

Gomoku is played on a square board, where the players take turns placing pieces on empty cells. The first player to achieve a line of five or more pieces of the own color wins the game; if the board is fully filled without any player winning it is a draw.

Knightthrough is very similar to Breakthrough, but here the pieces are knights instead of pawns. The moves are the same as in Chess, with the exception that they may only advance toward the opponent, never move back.

Nim consists of a number of stacks of matches. In each move, a player may remove any number of matches from one of the stacks. The player to take the last match wins the game.

Sheep and Wolf is played on a Chess-like board, where, similar to Checkers, only half the board is used. The sheep start on every second cell on one side, the wolf in the middle on the other side. The wolf moves first. The sheep may move only forward to a diagonally adjacent cell, while the wolf may move forward or backward to a diagonally adjacent cell. The goal of the sheep is to surround the wolf so that it cannot move any more; the goal of the wolf is to either block the sheep or to get behind them.

Game	without traps	trap depth 0	1	2	3	4	5	6	7
Breakthrough (8x8)	662 (119)	0	83	0	114	0	141	?	?
Chinese Checkers	904 (101)	3	5	6	7	4	17	7	47
Chomp (10x10)	14 (14)	4	0	687	0	32	0	263	0
Clobber (4x5)	121 (121)	515	0	23	0	64	0	277	0
Connect Four (7x6)	625 (85)	0	263	0	50	0	28	0	34
Nim (11,12,15,25)	469 (32)	0	40	0	102	0	44	0	345
Nim (12,12,20,20)	435 (41)	0	50	0	98	0	32	0	385
Sheep & Wolf (8x8)	882 (193)	0	11	0	11	0	22	0	74

Table 1: Trap search results for 1000 randomly sampled states, searching for traps of depth up to 7 (exception: Breakthrough only up to 5). The numbers are the depths of the deepest traps found in each state. Additionally, we give the number of states without traps, and for how many of those we can prove that they already are lost anyway (given in parantheses).

Traps in the Benchmark Games

We implemented an algorithm that evaluates games for getting an idea of the density of traps in those games. To do so, we first randomly choose the depth in which to find a state, and then perform a fully random game until this depth. The reached state will be the root of a Minimax tree, which we use to decide whether or not the state is at risk. If we can prove that the state is already lost anyway, there cannot be any trap. Otherwise, if we find some successor state that is provably lost, the root node is at risk, and the lost successor states correspond to traps. The depth of a trap is then the depth of the Minimax tree needed for proving it a lost state.

Table 1 displays the results of performing this approach for 1000 different sampled states and searching for traps of a depth of at most 7. For some games generating the full Minimax subtrees is not feasible. This is true for Gomoku and larger versions of Clobber. For Breakthrough this holds as well, but the algorithm finished when searching only for traps of depth 5 or less. In some games a state is at risk by several traps of different depths; the table gives only the depth of the deepest trap the algorithm identified.

Even though the algorithm did not work out for **Gomoku**, we assume it to contain a large number of shallow traps of at least depth 3. Whenever a player achieves a situation with a line of three pieces in the own color and the two cells on both sides of that line are empty, the opponent is in a state at risk. If the next move is not next to the line of three, a trap is reached as the player may then place a fourth piece adjacent to the existing line so that the adjacent cell on both sides is empty, which is an obvious win. Due to the large branching factor (the default board size is 15×15) and the possibility to continue playing for a long time without actually playing one of the finishing moves these traps are hard to detect by UCT, even though they are rather shallow.

In **Connect Four** the number of shallow traps is likely much smaller than in Gomoku. While it is enough to have a line of two pieces to create a state at risk it is further required that the two cells to each side of such a line must be immediately playable. As such, the surrounding board must be sufficiently filled with pieces. Additionally, a vertical line can not be seen to create a serious threat, as only one side of such a line remains open and due to the small branching factor UCT should have no trouble identifying it.

Situations of *zugzwang*, for which Connect Four is known, might also be considered as traps. However, these traps are not shallow as they typically result in filling several columns until the actual move to end the game can be played.

From the results in Table 1 we can see that most traps are of depth 1, which means that there is a line of 3 pieces of the opponent which it can finish in its next move – this can hardly be considered a serious trap, as it will be easily identified by UCT.

In **Breakthrough** we expected to be confronted with a large number of shallow traps. In a situation where an opponent's pawn is three cells from the current player's side and it is the last chance to take that pawn clearly is a state at risk. While Alpha-Beta will have no trouble identifying this as the game will be lost in five more steps, UCT again has to cope with a rather large branching factor and the fact that the game can continue for a long time if the simulations do not move the pawn that threatens to end the game. However, from the gathered results it seems that traps of depth 5 or less are not as common as we expected, at least in the 8×8 version of the game; only a third of all evaluated states contained such a trap.

For **Knightthrough** we expect that it contains a rather high density of shallow traps. Here a knight may be six cells from the opponent's side in order to need only three more own moves to reach it, so that states at risk can occur much earlier in the game.

The branching factor of Knightthrough is a bit higher than that of Breakthrough, but the length of the game typically is shorter, as the pieces can move up to two cells closer to the opponent's side. As such, the difficulty to identify traps for UCT might be similar to that in Breakthrough played on a board of the same size.

The game **Nim** is easily solved by mathematical methods (Bouton 1901). A winning strategy consists of reacting directly to the opponent's moves. The idea is to encode the stacks as binary numbers and then calculate the bitwise exclusive or of these numbers. If the result is different from $0 \ldots 0$ in the initial state the game is won for the starting player. In fact, the winning player can always counter an opponent's move in such a way that the result will be $0 \ldots 0$. This means that each state is at risk for the supposedly winning player, as a wrong move immediately means that the opponent can follow the same strategy and then ensure a win. However, this results in arbitrarily long games, so that we cannot expect to find many shallow traps easily identified by Alpha-Beta search.

From the results we can see that slightly more than half of the explored states contain traps of a depth of 7 or less. The somewhat surprisingly large number of shallow traps may be explained by the fact that in the tested cases we have only four stacks with relatively few matches, so that the endgame can be reached after only few steps in case of random play.

In **Chomp** every state is at risk: The player to move may choose to take the poisoned piece and thus immediately lose

the game, which corresponds to a trap of depth 0. Alternatively a player may decide to take the piece horizontally or vertically adjacent to the poisoned one. In such a situation the opponent can then take all remaining non-poisoned pieces. This corresponds to a trap of depth 2. However, both situations can hardly be considered as serious threats: In the second case, the branching factor and the maximal depth are rather small. From the evaluation results we see that these are the most common traps and they are the deepest ones for nearly 70% of the evaluated states.

For **Clobber** it is hard to find a general criterion for the presence of traps, so we used only our evaluation of sampled states. In our implementation of this game, a player can always give up and thus lose the game. This explains why we have so many traps of depth 0, which we can disregard as no player should fall for them. Traps of depth 2 are uncommon, starting with depth 4 they become much more common again (though in half the cases the states at risk that contain such a trap also contain one of depth 6). Finally, a quarter of all evaluated states contains a trap of depth 6. This high density of shallow traps might be due to the fact that the game is rather short (it typically ends after slightly more than ten moves); for larger boards (e.g., 5×6) we expect the situation to change and the number of shallow traps to decrease significantly in the early game (the first 8–10 moves).

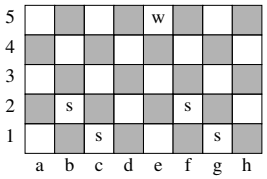


Figure 2: Relevant part of a state at risk in Sheep and Wolf.

For **Sheep and Wolf** consider the situation depicted in Figure 2. The sheep are to move next and the state is at risk. If the sheep on b2 is moved to a3 the wolf cannot be stopped from reaching cell c3. From there only one sheep is left that might stop it from going to b2 or d2, so that the wolf will win. If instead the sheep on c1 would have been moved to d2, the sheep could still win the game. Similar situations also exist with the wolf being closer to the sheep. However, our evaluation of sample states shows that such shallow traps are rather rare throughout the game. In total we found only 118 states at risk, and 74 of those had traps of depth 7.

Chinese Checkers requires to have all own pieces on the other side of the board in order to win. This means that for a trap of depth 7 or less all pieces must be placed in such a way that at most four own moves and/or jumps are required to reach the goal area. Thus, for most parts we are not confronted with any shallow traps.

Results for the Alpha-Beta Based Player

Here we provide results for running our Alpha-Beta players against the UCT player. All experiments were conducted on machines equipped with two Intel Xeon E5-2660 CPUs with 2.20 GHz and 64 GB RAM. Both processes were run on the same machine using one core each. We allowed a fixed amount of 10s for each move and performed a total of 200 runs for each game: 100 runs with UCT playing as Min player, and 100 runs with UCT as Max player.

Game	$\alpha\beta(0)$ vs. UCT	UCT vs. $\alpha\beta(0)$	$\alpha\beta(h_1)$ vs. UCT	UCT vs. $\alpha\beta(h_1)$	$\alpha\beta(h_{1+2})$ vs. UCT	UCT vs. $\alpha\beta(h_{1+2})$	$Q\alpha\beta(h_1)$ vs. UCT	UCT vs. $Q\alpha\beta(h_1)$
Breakthrough (6x6)	-0.22	-0.10	0.92	-1.00	0.80	-0.80	0.72	-0.90
Breakthrough (8x8)	0.30	-0.24	0.92	-1.00	0.94	-0.96	0.94	-0.94
Chinese Checkers	0.00	0.00	0.08	-0.18	0.03	0.00	0.32	-0.05
Chomp (10x10)	-1.00	0.64	-1.00	0.70	-1.00	0.50	-1.00	0.62
Clobber (4x5)	0.64	0.88	0.92	0.88	0.94	0.88	1.00	0.74
Clobber (5x6)	-0.92	0.94	-0.88	0.86	-0.88	0.94	-0.86	0.90
Connect Four (7x6)	-1.00	1.00	-0.86	0.75	-1.00	0.48	-0.91	0.70
Gomoku (8x8)	-0.98	0.98	0.44	0.33	0.69	0.21	0.50	0.49
Gomoku (15x15)	0.46	-0.24	1.00	-1.00	1.00	-0.98	1.00	-1.00
Knightthrough (8x8)	0.82	-0.82	0.72	-0.74	0.76	-0.82	0.70	-0.56
Nim (11,12,15,25)	0.70	-0.68	0.00	0.00	-0.04	0.06	-0.18	-0.12
Nim (12,12,20,20)	0.76	-0.78	0.04	-0.14	0.24	0.00	0.24	-0.34
Sheep & Wolf (8x8)	0.18	-1.00	0.04	-0.98	0.32	-1.00	0.12	-0.98

Table 2: Average rewards for the tested games using Alpha-Beta (six left columns) and Alpha-Beta with quiescence search (last two columns).

Table 2 shows the average rewards achieved when running Alpha-Beta with heuristic h_1 (denoted $\alpha\beta(h_1)$) and h_{1+2} (denoted $\alpha\beta(h_{1+2})$), as well as quiescence search with heuristic h_1 (denoted $Q\alpha\beta(h_1)$). As the results of quiescence search with heuristic h_{1+2} are very similar we omit those. Additionally, we used a blind heuristic (denoted $\alpha\beta(0)$), assigning each non-terminal state a value of 0, in order to show that our heuristics actually provide additional information over the basic trap detection inherent in Alpha-Beta search.

From these results we can make some observations. First of all, for the two evaluation functions and the two Alpha-Beta versions, the differences are surprisingly small. For the heuristics this might be explained by the fact that h_1 is a part of h_{1+2}. For quiescence search a possible explanation might be that the benefit of increased depth in some parts results in shallower depth in others due to the fixed time-out, so that overall both searches perform similar.

Second, the additional information of the new evaluation function provides a significant advantage in the games Breakthrough and Gomoku. While the players using the evaluation functions consistently win especially on larger boards, the player with the blind heuristic performs much worse. Obviously our heuristics are good enough for these games to prevent creating situations from which the player cannot recover, i.e., falling into traps deeper than the depth of the Alpha-Beta search tree. For the smaller version of Clobber and for Connect Four the advantage of using our evaluation functions is not as big but still noticeable. However, in the game of Nim the blind heuristic performs much better – here the evaluation functions are clearly misleading.

For Gomoku we note that while Alpha-Beta using the evaluation functions and UCT achieve similar results on the small 8×8 sized board, on the traditional 15×15 board UCT fails completely. Here we see that a likely higher number of

shallow traps together with a large branching factor and the possibility of long playouts results in an immense decrease in performance of UCT. An inverse observation can be made for Clobber: while Alpha-Beta fares reasonably well on a board of size 4×5, the density of shallow traps is likely smaller on the larger board of size 5×6, resulting in an advantage for UCT.

Considering Connect Four, we note that even though the game is closely related to Gomoku, Alpha-Beta fares much worse. As pointed out before, in Connect Four the number of shallow traps is rather small, so that the chances of UCT falling for one are decreased. Concerning Chomp, even though every state is a state at risk, we can ignore traps of depth 0 and 2. Other than these, the trap density is rather small. In the end, this results in bad performance of the Alpha-Beta players compared to the UCT player.

Not all games with few shallow traps are bad for our Alpha-Beta players with the evaluation function: In Chinese Checkers and Nim they are still on-par with UCT. Finally, Sheep and Wolf gives a rather surprising result. The number of shallow traps is not overly high, the branching factor is comparatively small and the length of the game is clearly limited by the size of the board (at worst, all sheep must be moved to the other side). It is quite easy to come up with a strategy where the Min player (the sheep) wins the game. Obviously, our UCT player cannot identify such a strategy while the Alpha-Beta player can, so that the UCT player wins less than half the games when playing as Min, while Alpha-Beta consistently wins.

Related Work on Evaluation Functions for GGP

While most state-of-the art players nowadays make use of UCT, there has been some research in the use of evaluation functions for GGP.

When the current form of GGP was introduced in 2005, the first successful players made use of Alpha-Beta with automatically generated evaluation functions. The basic idea was to identify features of the game at hand (e.g., game boards, cells, movable pieces). By taking order relations into account, it is possible to evaluate distances of pieces to their goal locations (where the order relations describe the connection of the cells of a game board) or the difference in number of pieces of the players (where the order relations describe the increase/decrease of pieces, e.g., when one is captured) (see, e.g., (Kuhlmann, Dresner, and Stone 2006; Clune 2007)).

Another way to evaluate states was used by Fluxplayer (Schiffel and Thielscher 2007): This uses fuzzy logic to evaluate how well the goal conditions are already satisfied. In a setting with a simple conjunction of facts, as we assume in this paper, this pretty much corresponds to a goal-counting heuristic. Additionally they also took identified features and order relations into account to improve this evaluation function. They do so by using different weights, e.g., taking a fact's distance to its goal value into account instead of only a satisfied/unsatisfied status.

A more recent approach (Michulke and Schiffel 2012) considers a so-called fluent graph, which captures some conditions for a fact to become true, but for each action considers only one of the preconditions as necessary for achieving one of its effects. Based on this graph an estimate on the number of moves needed for achieving a fact is calculated, which is again used for weighing the fuzzy logic formulas, similar to the previous approach in Fluxplayer. A similar graph, the so called justification graph, has been used in planning for calculating the efficient LM-Cut heuristic (Helmert and Domshlak 2009), though there the graph is used to calculate disjunctive action landmarks.

While in principle it should be possible to use our proposed heuristics (or other planning based distance estimates) in a similar way, it is not clear how useful this might be. The fuzzy logic based approach of Fluxplayer makes sense when applied in the original GDL setting, which allows for arbitrary Boolean formulas with conjunctions and disjunctions, at least when rolling out axioms. In our setting, however, we allow only conjunctions of variables in the goal descriptions. One way to emulate the Fluxplayer approach in our setting would be to calculate the required distance for each of the goal variables, and then combine those results to calculate an actual value of the evaluation function. If this improves the results is not immediately clear and remains as future work.

Conclusion

In this paper we have proposed new evaluation functions for general two-player zero-sum games inspired by successful heuristics used in automated planning, which are based on ignoring delete lists. By taking the difference in plan lengths for reaching won/lost states and the factor of still relevant actions into account, we ended up with a heuristic with which an Alpha-Beta based player is able to consistently defeat a basic UCT player on games with a large amount of traps. It also copes rather well in some of the games having only few shallow traps, where UCT typically is expected to work well.

In addition to these new evaluation functions we also provided some insight into the presence of traps in a set of GGP benchmarks. The observation here is that basically all those games contain some shallow traps, though for several games the density is rather small, which is a factor explaining the success of UCT players in the GGP setting.

In the future we will adapt further heuristics to two-player games. One approach that comes to mind is the use of abstractions. For some extensive games such an approach has yielded pathological behavior (Waugh et al. 2009), i.e., worse play when refining an abstraction, and it will be interesting to see if such behavior can also occur in our setting.

References

Auer, P.; Cesa-Bianchi, N.; and Fischer, P. 2002. Finite-time analysis of the multiarmed bandit problem. *Machine Learning* 47(2–3):235–256.

Björnsson, Y., and Finnsson, H. 2009. Cadiaplayer: A simulation-based general game player. *IEEE Transactions on Computational Intelligence and AI in Games* 1(1):4–15.

Bouton, C. L. 1901. Nim, a game with a complete mathematical theory. *Annals of Mathematics* 3(2):35–39.

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

Campbell, M.; Hoane, Jr., A. J.; and Hsu, F.-H. 2002. Deep Blue. *Artificial Intelligence* 134(1–2):57–83.

Clune, J. 2007. Heuristic evaluation functions for general game playing. In Howe, A., and Holte, R. C., eds., *Proceedings of the 22nd National Conference of the American Association for Artificial Intelligence (AAAI-07)*, 1134–1139. Vancouver, BC, Canada: AAAI Press.

Gelly, S., and Silver, D. 2008. Achieving master level play in 9 x 9 computer go. In Fox, D., and Gomes, C., eds., *Proceedings of the 23rd National Conference of the American Association for Artificial Intelligence (AAAI-08)*, 1537–1540. Chicago, Illinois, USA: AAAI Press.

Genesereth, M. R.; Love, N.; and Pell, B. 2005. General game playing: Overview of the AAAI competition. *AI Magazine* 26(2):62–72.

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

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

Kocsis, L., and Szepesvári, C. 2006. Bandit based Monte-Carlo planning. In Fürnkranz, J.; Scheffer, T.; and Spiliopoulou, M., eds., *Proceedings of the 17th European Conference on Machine Learning (ECML 2006)*, volume 4212 of *Lecture Notes in Computer Science*, 282–293. Springer-Verlag.

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

Kuhlmann, G.; Dresner, K.; and Stone, P. 2006. Automatic heuristic construction in a complete general game player. In Gil, Y., and Mooney, R. J., eds., *Proceedings of the 21st National Conference of the American Association for Artificial Intelligence (AAAI-06)*, 1457–1462. Boston, Massachusetts, USA: AAAI Press.

Love, N. C.; Hinrichs, T. L.; and Genesereth, M. R. 2008. General game playing: Game description language specification. Technical Report LG-2006-01, Stanford Logic Group.

Méhat, J., and Cazenave, T. 2011. A parallel general game player. *KI* 25(1):43–47.

Michulke, D., and Schiffel, S. 2012. Distance features for general game playing agents. In Filipe, J., and Fred, A. L. N., eds., *Proceedings of the 4th International Conference on Agents and Artificial Intelligence (ICAART'12)*, 127–136. Vilamoura, Algarve, Portugal: SciTePress.

Newell, A., and Simon, H. 1963. GPS, a program that simulates human thought. In Feigenbaum, E., and Feldman, J., eds., *Computers and Thought*. McGraw-Hill. 279–293.

Ramanujan, R.; Sabharwal, A.; and Selman, B. 2010. On adversarial search spaces and sampling-based planning. In Brafman, R. I.; Geffner, H.; Hoffmann, J.; and Kautz, H. A., eds., *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS'10)*, 242–245. AAAI Press.

Ramanujan, R.; Sabharwal, A.; and Selman, B. 2011. On the behavior of UCT in synthetic search spaces. In *Proceedings of the ICAPS Workshop on Monte-Carlo Tree Search: Theory and Applications (MCTS'11)*.

Schaeffer, J.; Culberson, J.; Treloar, N.; Knight, B.; Lu, P.; and Szafron, D. 1992. A world championship caliber checkers program. *Artificial Intelligence* 53(2–3):273–289.

Schiffel, S., and Thielscher, M. 2007. Fluxplayer: A successful general game player. In Howe, A., and Holte, R. C., eds., *Proceedings of the 22nd National Conference of the American Association for Artificial Intelligence (AAAI-07)*, 1191–1196. Vancouver, BC, Canada: AAAI Press.

Waugh, K.; Schnizlein, D.; Bowling, M. H.; and Szafron, D. 2009. Abstraction pathologies in extensive games. In Sierra, C.; Castelfranchi, C.; Decker, K. S.; and Sichman, J. S., eds., *Proceedings of the 8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'09)*, 781–788. Budapest, Hungary: IFAAMAS.

Winands, M. H. M., and Björnsson, Y. 2011. $\alpha\beta$-based play-outs in monte-carlo tree search. In Cho, S.-B.; Lucas, S. M.; and Hingston, P., eds., *Proceedings of the 2011 IEEE Conference on Computational Intelligence and Games (CIG 2011)*, 110–117. Seoul, South Korea: IEEE.

Generalized Label Reduction for Merge-and-Shrink Heuristics

Silvan Sievers and **Martin Wehrle** and **Malte Helmert**
Universität Basel
Basel, Switzerland
{silvan.sievers,martin.wehrle,malte.helmert}@unibas.ch

Abstract

Label reduction is a technique for simplifying families of labeled transition systems by dropping distinctions between certain transition labels. While label reduction is critical to the efficient computation of merge-and-shrink heuristics, current theory only permits reducing labels in a limited number of cases. We generalize this theory so that labels can be reduced in *every* intermediate abstraction of a merge-and-shrink tree. This is particularly important for efficiently computing merge-and-shrink abstractions based on *non-linear* merge strategies. As a case study, we implement a non-linear merge strategy based on the original work on merge-and-shrink heuristics in model checking by Dräger et al.

Introduction

State-space search is a fundamental problem in artificial intelligence. Many state spaces of interest, including those that arise in classical planning and in the verification of safety properties in model checking, can be compactly specified as a family of labeled transition systems (e. g., Helmert, Haslum, and Hoffmann 2008; Dräger, Finkbeiner, and Podelski 2009).

Label reduction identifies and eliminates semantically equivalent labels in such transition systems. It was originally introduced as an efficiency improvement for *merge-and-shrink abstractions* (Helmert, Haslum, and Hoffmann 2007). Later, Nissim, Hoffmann, and Helmert (2011a) showed that label reduction can (in some cases) exponentially reduce the representation size of abstractions based on *bisimulation*.

All implementations of merge-and-shrink abstractions described in the planning literature apply label reduction whenever possible: it has no negative impact on abstraction quality, is very fast to compute, and significantly reduces time and memory required to compute an abstraction. However, the current theory of merge-and-shrink abstractions only allows reducing labels in limited cases.

Broadly speaking, the merge-and-shrink approach consists in constructing a set of *atomic* transition systems, each corresponding to a single state variable of the problem, and then iteratively *merging* two transition systems into a larger one until only one transition system remains, which then induces a heuristic for an overall state-space search algorithm. Intermediate results can be *shrunk* to trade off computation effort against heuristic accuracy. A so-called *merge strategy* decides which transition systems to merge in each step of the algorithm. The merge strategy defines a binary tree over the atomic transition systems, the so-called *merge tree*.

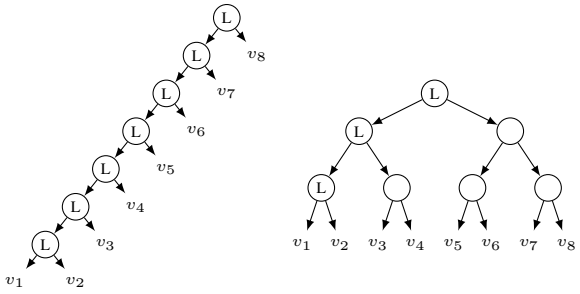


Figure 1: Two merge trees for a problem with 8 state variables. Previous theory allows reducing labels in the intermediate abstractions marked with an "L" when v_1 is the pivot.

Figure 1 shows two possible merge trees for a state space with 8 atomic transition systems, defined by state variables $v_1, \ldots, v_8$. The left part of the figure shows a merge tree which degenerates to a list; such merge trees correspond to so-called *linear merge strategies* (Helmert, Haslum, and Hoffmann 2007). The right part shows a complete merge tree, corresponding to a *non-linear merge strategy*. According to current theory (Nissim, Hoffmann, and Helmert 2011a), when defining a merge strategy, one must select a single leaf of the merge tree, called a *pivot*, and may only reduce labels after merge operations which correspond to ancestors of the pivot in the merge tree. In general, this means that with a complete tree over n atomic transition systems, only $O(\log n)$ of the merged transition systems can have their labels reduced.

We introduce a generalized concept of label reduction to overcome this limitation. The generalization is introduced in a declarative way, independently of the merge-and-shrink framework. It is conceptually much easier to understand than the previous theory, yet more powerful in the sense that it allows reducing to a smaller set of labels than previous

Copyright © 2014, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.
This work has been published at AAAI 2014.

techniques and in the sense that it can be applied safely to *every* intermediate abstraction of a merge tree.

Generalized label reduction is particularly beneficial for the efficient computation of merge-and-shrink abstractions with non-linear merge strategies. As a case study, we have implemented such a merge strategy, based on the original work on merge-and-shrink heuristics in model checking by Dräger, Finkbeiner, and Podelski (2006), which did not make use of label reduction. We show experimental results that highlight the usefulness of generalized label reduction in general and non-linear merge strategies in particular.

Planning Tasks

We present our techniques with the terminology of automated planning, but note that they are applicable to factored transition systems in general. We consider planning tasks in the SAS$^+$ formalism (Bäckström and Nebel 1995) augmented with action costs. A *planning task* is a 4-tuple $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$, where $\mathcal{V}$ is a finite set of *state variables*, $\mathcal{O}$ is a finite set of *operators*, s_0 is the *initial state* and $s_\star$ is the *goal*.

Each variable $v \in \mathcal{V}$ has a finite domain $\mathcal{D}(v)$. A *partial state* s is a variable assignment on a subset of $\mathcal{V}$, denoted by $\mathit{vars}(s)$. We write $s[v]$ for the value assigned to $v \in \mathit{vars}(s)$, which must satisfy $s[v] \in \mathcal{D}(v)$. We say that s *complies* with partial state s' if $s[v] = s'[v]$ for all $v \in \mathit{vars}(s) \cap \mathit{vars}(s')$. A partial state s is a *state* if $\mathit{vars}(s) = \mathcal{V}$.

Each operator $o \in \mathcal{O}$ has a *precondition* $pre(o)$ and *effect* $\mathit{eff}(o)$, which are partial states, and a *cost* $c(o) \in \mathbb{R}_0^+$. An operator o is *applicable* in a state s if s complies with $pre(o)$, in which case o can be *applied*, resulting in the *successor state* s' that complies with $\mathit{eff}(o)$ and satisfies $s'[v] = s[v]$ for all $v \notin \mathit{vars}(\mathit{eff}(o))$.

The initial state s_0 is a state; the goal $s_\star$ is a partial state.

A *plan* is a sequence $o_1, \ldots, o_n \in \mathcal{O}$ of operators which are applicable, in order, to the initial state, resulting in a state that complies with the goal. Such a plan is *optimal* if $\sum_{i=1}^n c(o_i)$ is minimal among all plans. The objective of optimal planning is to find an optimal plan for a planning task or to prove that no plan exists.

Transition Systems and Merge-and-Shrink

We briefly recap the key ideas behind merge-and-shrink abstractions (e. g., Helmert, Haslum, and Hoffmann 2007). The central notion in this context is the explicit manipulation of *transition systems*. We define a transition system as a 4-tuple $\Theta = \langle S, L, T, S_\star \rangle$ where S is a finite set of *states*, L is a finite set of *labels*, $T \subseteq S \times L \times S$ is a set of (labeled) *transitions*, and $S_\star \subseteq S$ is the set of *goal states*. Each label $l \in L$ has a *cost* $c(l) \in \mathbb{R}_0^+$. Where it simplifies notation, we write $s \xrightarrow{l} s'$ to denote a transition $\langle s, l, s' \rangle$ from s to s' with label l, and we may write $s \xrightarrow{l} s' \in \Theta$ for $s \xrightarrow{l} s' \in T$.

A planning task naturally induces a transition system, which is usually too large to be represented explicitly. Instead, the merge-and-shrink approach works with a set X of smaller transition systems, which it iteratively transforms until only one transition system remains. This final transition system is then used to define a heuristic for solving the planning task.

The process starts by setting X to the set of *atomic* transition systems, which capture the behaviour of a single state variable. Then X is transformed by repeatedly applying one of the following two operations:

- *Merge:* Remove two transition systems $\Theta = \langle S, L, T, S_\star \rangle$ and $\Theta' = \langle S', L, T', S'_\star \rangle$ from X and replace them with their *synchronized product* $\Theta \otimes \Theta' = \langle S \times S', L, T^\otimes, S_\star \times S'_\star \rangle$, where a *synchronized transition* $\langle s, s' \rangle \xrightarrow{l} \langle t, t' \rangle \in T^\otimes$ exists iff $s \xrightarrow{l} t \in T$ and $s' \xrightarrow{l} t' \in T'$.
- *Shrink:* Remove a transition system $\Theta = \langle S, L, T, S_\star \rangle$ from X and replace it with the *abstract transition system* $\alpha(\Theta) := \langle \alpha(S), L, \{\langle \alpha(s), l, \alpha(t) \rangle \mid \langle s, l, t \rangle \in T\}, \alpha(S_\star) \rangle$, where α is an arbitrary function on S.

We remark that it is critical for merge operations (and hence for the correctness of the overall approach) that all transition systems work on a common set of labels. In the "basic" merge-and-shrink approach described in the paper by Helmert et al. (2007), this is always the set of operators of the underlying planning task. This changes when we make use of *label reduction*, described in the following section.

Before we move to label reduction, it is useful to introduce one more concept: the *global transition system* represented by X is the synchronized product (merge) of all elements in X, which we denote by $\bigotimes X$. (The product operator is associative and commutative modulo names of states, which we do not care about, so this is well-defined without having to specify an order on the individual merges.) At every stage of the merge-and-shrink algorithm, the current set X can be seen as a compact representation of $\bigotimes X$. In planning, initially $\bigotimes X$ equals the global transition system of the planning task (shown by Helmert et al., 2007). Merge steps do not change the represented global system, and shrink steps apply an abstraction to it.

Label Reduction: State of the Art

Label reduction adds a third class of transformations to the merge-and-shrink approach. It was first implemented, but not described, in the original application of merge-and-shrink abstractions to planning (Helmert, Haslum, and Hoffmann 2007). Nissim et al. (2011a) gave the first description; Helmert et al. (2014) discuss it more thoroughly. The key idea is to identify transition labels that can be combined into a single label without losing relevant information. Among other benefits, this can significantly reduce the representation size of the transition system because parallel transitions with different labels can collapse into a single transition.

The existing theory of label reduction is very complicated. We do not describe it in detail here: this would require much space, and a full description is not necessary for this paper. Details can be found in Section 5 of Nissim et al. (2011a) and Section 5 of Helmert et al. (2014). Here, it suffices to discuss three weaknesses of the current theory.

Firstly, the current theory largely attempts to define label reduction as a *local* concept considering individual transition systems: the central notion is that of a *label-reduced transition system*. This is fundamentally at odds with the

purpose of labels in the merge-and-shrink framework to coordinate the joint behaviour of *all* transition systems in the set X. If we change the labels in some, but not all transition systems in X, synchronization cannot work correctly.

The earlier papers address this difficulty by performing a kind of "just-in-time label reduction" that makes the labels of two transition systems correspond just before they are merged (which is the only point at which labels matter). This works, but the resulting theory is complex to understand and reason about, as different parts of the merge tree work with different labels. Consequently, current theory only permits reducing labels in certain cases, with other cases deemed to be unsafe and hence forbidden. Complications mainly arise in the case of *non-linear merge strategies*, and consequently, these were never correctly implemented.

Secondly, the current theory of label reduction is in a certain sense *syntax-based* while the rest of the merge-and-shrink framework is *semantic*. *Merge* operations and *shrink* operations are purely semantic: once a planning task (or other problem) is translated into atomic transition systems, the task description is not needed any more. Labels are opaque tokens that do not need to "stand for" anything. This greatly simplifies the theory of merge-and-shrink abstractions and makes them very flexible: they work for everything representable as transition systems.

Unfortunately, the current theory of label reduction needs to "look inside" the labels in order to decide which labels can be combined into one. For planning tasks, label reduction must treat labels as structured pairs of preconditions and effects, reintroducing and critically depending on the syntactic descriptions we would prefer not to have to reason about.

Thirdly, current theory cannot exploit label reductions that are enabled by shrinking. The decision how to reduce labels is completely independent of the shrink steps of the algorithm and hence needs to be correct for *all possible* shrink strategies. This severely limits simplification possibilities.

All these issues are addressed in the new theory of label reduction developed in the following section.

Label Reduction: New Theory

In this section, we introduce the new theory of label reduction and discuss its properties. Like the *merge* and *shrink* operations described earlier, we define label reduction as a transformation of the set X of transition systems:

- *Reduce labels:* Let τ be a *label mapping*, i. e., a function defined on the labels L of X, which satisfies $c(\tau(l)) \leq c(l)$ for all $l \in L$. Replace each transition system $\Theta = \langle S, L, T, S_\star \rangle \in X$ with the *label-reduced* system $\tau(\Theta) := \langle S, \tau(L), \{\langle s, \tau(l), t\rangle \mid \langle s, l, t\rangle \in T\}, S_\star\rangle$.

In words, label reduction means replacing all occurrences of each label l in all transition systems by the new label $\tau(l)$. (Of course, $\tau(l) = l$ is permitted.) When we choose to introduce a new label (i. e., $\tau(l) \notin L$), its cost can be set arbitrarily as long as it does not exceed $c(l)$. The operation is called label *reduction* because it is generally used to reduce the number of labels by choosing a non-injective function τ. (Using an injective function τ is possible, but pointless.)

It is worth emphasizing that, unlike previous definitions, label reduction always affects *all* transition systems simultaneously. As we will see in the following, this is sufficient to guarantee that label reduction is always "safe" to be applied. Unlike the previous theory, there is no need for pivot variables or to restrict label reduction to certain stages of the merge-and-shrink computation. Also, labels in the new theory always remain completely opaque objects (without associated "preconditions" and "effects").

However, there is a complication: the previous theory of label reduction reasoned about preconditions and effects to decide which labels can be combined to obtain *exact* label reductions, i. e., ones that do not introduce spurious transitions in $\bigotimes X$. With opaque labels, the question of exact label reduction must be addressed on the *semantic* level. Fortunately, we will see later that this is quite easy to do and more powerful than the previous syntax-based methods.

Properties of Label Reduction

To be able to use merge-and-shrink abstractions for admissible heuristics, we must guarantee that whenever a path from a given state s to some goal state exists in the actual problem, a corresponding path of at most the same cost exists in the final transition system computed.

Consider a transformation of a set of transition systems X with labels L into a new set X' with labels L' (e. g., merging, shrinking or reducing labels). We call such a transformation *transition-safe* if all transitions in $\bigotimes X$ have a corresponding transition in $\bigotimes X'$ (possibly with a different label) and goal states are preserved. Formally, the transformation is transition-safe if there exist functions α and τ mapping the states and labels of $\bigotimes X$ to the states and labels of $\bigotimes X'$ such that $\tau(L) = L'$, $s \xrightarrow{l} t \in \bigotimes X$ implies $\alpha(s) \xrightarrow{\tau(l)} \alpha(t) \in \bigotimes X'$ for all s, l, t, and $\alpha(s_\star)$ is a goal state of $\bigotimes X'$ for all goal states $s_\star$ of $\bigotimes X$.

We call a transformation *transition-exact* if additionally it does not give rise to any "new" transitions or goal states. Formally, the transformation is transition-exact if it is transition-safe, $s' \xrightarrow{l'} t' \in \bigotimes X'$ implies $s \xrightarrow{l} t \in \bigotimes X$ for *all* $s \in \alpha^{-1}(s')$ and $t \in \alpha^{-1}(t')$ and *some* $l \in \tau^{-1}(l')$, and for all goal states $s'_\star$ of $\bigotimes X'$ *all* states in the preimage $\alpha^{-1}(s'_\star)$ are goal states of $\bigotimes X$.

We call a transformation *cost-safe* if it cannot increase label costs and *cost-exact* if additionally it cannot decrease label costs. Formally, a transition-safe transformation must satisfy $c(\tau(l)) \leq c(l)$ for all $l \in L$, and a cost-exact one must satisfy $c(\tau(l)) = c(l)$ for all $l \in L$.

Finally, a transformation is *safe* if it is transition-safe and cost-safe and *exact* if it is transition-exact and cost-exact.

It is easy to verify that if each step in a sequence of transformations has one of these properties (e. g., is transition-safe), then the overall transformation also has it. (To prove this, compose the α and τ functions of each step.) Safe transformations give rise to admissible and consistent heuristics, and exact transformations give rise to perfect heuristics. Hence, it is important to verify that all transformations used in a merge-and-shrink heuristic computation are safe, and exact transformations are especially desirable.

Previous work on merge-and-shrink (e. g., Helmert et al. 2014) established that *merging* is always exact, *shrinking* is always safe, and shrinking based on *perfect bisimulation* is exact. We now establish that in the new theory, label reduction is always safe.

Consider a label reduction with mapping τ that transforms $X = \{\Theta_1, \ldots, \Theta_n\}$ into $X' = \{\tau(\Theta_1), \ldots, \tau(\Theta_n)\}$. We first show that this label reduction is transition-safe. Here and in the following, we write states of $\bigotimes X$ and $\bigotimes X'$ as tuples $\langle s_1, \ldots, s_n \rangle$ where each s_i is a state of Θ_i. Consider some transition $\langle s_1, \ldots, s_n \rangle \xrightarrow{l} \langle t_1, \ldots, t_n \rangle \in \bigotimes X$. By the definition of products, we have $s_i \xrightarrow{l} t_i \in \Theta_i$ for all $1 \leq i \leq n$; by the definition of label reduction, we have $s_i \xrightarrow{\tau(l)} t_i \in \tau(\Theta_i)$ for all $1 \leq i \leq n$; finally, again by definition of products we have $\langle s_1, \ldots, s_n \rangle \xrightarrow{\tau(l)} \langle t_1, \ldots, t_n \rangle \in \bigotimes X'$. With α set to the identity function, this proves that label reduction is transition-safe. (Label reduction does not change the set of goal states.) Due to the condition on τ in the definition of label reduction, the transformation is also cost-safe. In summary, label reduction is safe.

Exact Label Reduction

Previous papers that study label reduction in the merge-and-shrink framework (Nissim, Hoffmann, and Helmert 2011a; Helmert et al. 2014) focus on the question which conditions are required to make label reduction exact. In particular, exact label reduction is a critical ingredient in the polynomial-time perfect heuristics obtained in some planning domains (Nissim, Hoffmann, and Helmert 2011a).

Helmert et al. (2014) discuss conditions for exactness of label reduction that are sufficient and in a certain sense necessary, thus seemingly closing the topic of exact label reduction. However, these results do not directly apply to our theory, as they rely on the limitations of the previous theory. We revisit the topic here, proving a sufficient and necessary condition for exact label reduction that generalizes the previous result.

It is obvious that a label reduction is cost-exact iff it only combines labels of the same cost (i. e., $\tau(l) = \tau(l')$ implies $c(l) = c(l')$), and of course we must always set $c(\tau(l)) := c(l)$ to be cost-exact. It remains to discuss under which conditions label reduction is transition-exact. We start by introducing some additional terminology.

Definition 1. *Let X be a set of transition systems with labels L. Let $l, l' \in L$ be labels, and let $\Theta \in X$.*

Label l is alive *in X if all transition systems $\Theta' \in X$ have some transition $s \xrightarrow{l} t \in \Theta'$. Otherwise, l is* dead.

Label l locally subsumes *label l' in Θ if for all $s \xrightarrow{l'} t \in \Theta$ we also have $s \xrightarrow{l} t$. Label l* globally subsumes *label l' in X if l locally subsumes l' in all $\Theta' \in X$.*

Labels l and l' are locally equivalent *in Θ if they label the same transitions in Θ, i. e., if l and l' locally subsume each other in Θ.*

Labels l and l' are Θ-combinable *in X if they are locally equivalent in all transition systems $\Theta' \in X \setminus \{\Theta\}$. (It does not matter whether or not they are locally equivalent in Θ.)*

It is easy to see that dead labels induce no transitions in $\bigotimes X$. Consequently, it is an exact transformation to remove all dead labels (and their transitions) from X. Hence, it suffices to consider the case where X has no dead labels.

Moreover, we can restrict attention to label reductions τ that combine two labels l_1 and l_2 into some new label l_{12} ($\tau(l_1) = \tau(l_2) = l_{12}$) while leaving all other labels unchanged ($\tau(l') = l'$ for all $l' \notin \{l_1, l_2\}$). Other label reductions can be represented as chains of such "minimal" label reductions. We are now ready to state our major result.

Theorem 1. *Let X be a set of transition systems without dead labels. Consider a label reduction on X which combines labels l_1 and l_2 and leaves other labels unchanged.*

This label reduction is exact iff $c(l_1) = c(l_2)$ and

1. *l_1 globally subsumes l_2, or*
2. *l_2 globally subsumes l_1, or*
3. *l_1 and l_2 are Θ-combinable for some $\Theta \in X$.*

Proof. Let τ be the described label mapping, let $X = \{\Theta_1, \ldots, \Theta_n\}$ and let $X' = \{\tau(\Theta_1), \ldots, \tau(\Theta_n)\}$ be the result of label reduction. Let $l_{12} := \tau(l_1) = \tau(l_2)$.

Clearly, the label reduction is cost-exact iff $c(l_1) = c(l_2)$. We need to show that it is transition-exact iff 1., 2., or 3. holds. We prove this in three parts:

(A) If neither 1. nor 2. nor 3. holds, then the label reduction is not exact.

(B) If 1. or 2. holds, then the label reduction is exact.

(C) If 3. holds, then the label reduction is exact.

Label reduction is always transition-safe and leaves the set of goal states unchanged, so we only need to consider the second condition in the definition of transition-exactness.

On (A): We must show that no function α satisfies the criterion of transition-exactness. It is sufficient to consider the case where α is a bijection because $\bigotimes X$ and $\bigotimes X'$ have the same number of states, so non-bijective α cannot possibly work. Renaming states does not affect the notion of exactness, so we can further limit attention to α being the identity function without loss of generality.

We say that a transition system $\Theta \in X$ has an l_1-*only* transition if there exists a transition $s \xrightarrow{l_1} t \in \Theta$ with $s \xrightarrow{l_2} t \notin \Theta$. Symmetrically, it has an l_2-*only* transition if there exists a transition $s \xrightarrow{l_2} t \in \Theta$ with $s \xrightarrow{l_1} t \notin \Theta$.

We try to find two transition systems $\Theta_i, \Theta_j \in X$ with $i \neq j$ such that there is an l_1-only transition $s_i \xrightarrow{l_1} t_i \in \Theta_i$ and an l_2-only transition $s_j \xrightarrow{l_2} t_j \in \Theta_j$. Then $\Theta_i \otimes \Theta_j$ does not contain a transition $\langle s_i, s_j \rangle \xrightarrow{l} \langle t_i, t_j \rangle$ for either $l = l_1$ or $l = l_2$, but $\tau(\Theta_i) \otimes \tau(\Theta_j)$ does contain the transition $\langle s_i, s_j \rangle \xrightarrow{l_{12}} \langle t_i, t_j \rangle$. By induction over the remaining transition systems, it is then easy to show that $\bigotimes X'$ contains a transition that does not correspond to a transition in $\bigotimes X$, proving inexactness. (Here, we use that there are no dead labels: the argument fails if l_1 and l_2 are dead.) It remains to show that l_1-only and l_2-only transitions in different transition systems of X exist.

Because 1. does not hold, there exists an l_2-only transition in some transition system $\Theta \in X$. Because 2. does not hold, there exists an l_1-only transition in some transition system $\Theta' \in X$. If Θ and Θ' are different transition systems, we have found the required transitions and are done.

So let us assume that $\Theta = \Theta'$. Because 3. does not hold, there exist at least two transition systems where l_1 and l_2 are not locally equivalent, so there is at least one transition system $\Theta'' \neq \Theta$ where they are not locally equivalent. This means that Θ'' must have an l_1-only transition or an l_2-only transition. In the former case, we select the l_1-only transition in Θ'' and the l_2-only transition in Θ. Otherwise, we select the l_2-only transition in Θ'' and the l_1-only transition in Θ' $(= \Theta)$.

On (B): Consider Case 1., where l_1 globally subsumes l_2. Case 2. is identical with l_1 and l_2 swapped. As the function α in the definition of transition-exactness, we choose the identity mapping. Then the condition for transition-exactness we need to verify simplifies to: for all $s \xrightarrow{l'} t \in \bigotimes X'$, there exists a label $l \in \tau^{-1}(l')$ with $s \xrightarrow{l} t \in \bigotimes X$. For $l' \neq l_{12}$, this is trivial because $\bigotimes X$ and $\bigotimes X'$ are exactly identical regarding labels other than l_1, l_2 and l_{12}. So consider the case $l' = l_{12}$. Let $s = \langle s_1, \ldots, s_n \rangle$ and let $t = \langle t_1, \ldots, t_n \rangle$. From $s \xrightarrow{l_{12}} t \in \bigotimes X'$ we get $s_i \xrightarrow{l_{12}} t_i \in \tau(\Theta_i)$ for all $1 \leq i \leq n$, and hence $s_i \xrightarrow{l_1} t_i \in \Theta_i$ or $s_i \xrightarrow{l_2} t_i \in \Theta_i$ for all $1 \leq i \leq n$. Because l_1 globally subsumes l_2, this implies $s_i \xrightarrow{l_1} t_i \in \Theta_i$ for all $1 \leq i \leq n$, and hence $s \xrightarrow{l_1} t \in \bigotimes X$, concluding this part of the proof.

On (C): As in (B), we set α to the identity function and only need to consider transitions $s \xrightarrow{l_{12}} t \in \bigotimes X'$. Let $s = \langle s_1, \ldots, s_n \rangle$ and let $t = \langle t_1, \ldots, t_n \rangle$. Again, we obtain that for all $1 \leq i \leq n$, $s_i \xrightarrow{l_{12}} t_i$ and hence $s_i \xrightarrow{l_1} t_i$ or $s_i \xrightarrow{l_2} t_i$. Choose $l \in \{l_1, l_2\}$ such that $s_j \xrightarrow{l} t_j$, where $j \in \{1, \ldots, n\}$ is chosen in such a way that l_1 and l_2 are Θ_j-combinable in X. (Such a transition system Θ_j exists because we are in Case 3.) By the definition of Θ-combinable, l_1 and l_2 are locally equivalent for all transition systems in X other than Θ_j, and hence ($s_i \xrightarrow{l_1} t_i$ or $s_i \xrightarrow{l_2} t_i$) implies ($s_i \xrightarrow{l_1} t_i$ *and* $s_i \xrightarrow{l_2} t_i$) for all $i \neq j$. This shows that $s_i \xrightarrow{l} t_i \in \Theta_i$ for all $1 \leq i \leq n$, and hence $s \xrightarrow{l} t \in \bigotimes X$, concluding the final part of the proof. □

We conclude the section with a brief discussion of the conditions in Theorem 1. Although all conditions can be checked in low-order polynomial time, there is a practical difference in complexity. Finding Θ-combinable labels essentially consists in computing the local equivalence relations of all $\Theta \in X$, which is possible in linear time in the representation size of X. In contrast, finding globally subsumed labels involves finding subset relationships in a set family, for which to the best of our knowledge no linear-time algorithms are known.

A comparison to the results of Helmert et al. (2014) shows that the Θ-combinability condition strictly generalizes the previous conditions on exactness. Hence, the new theory permits a larger number of exact label reductions even if we only use Θ-combinability and do not consider global subsumption of labels. For this reason, coupled with efficiency concerns, we only perform exact label reductions based on Θ-combinability in our experiments, which we describe next.

Experiments

As discussed in the preceding sections, the new theory of label reduction is significantly more general and at the same time much less complicated than previous work. However, we have yet to establish that it is useful for practical implementations of merge-and-shrink heuristics.

Firstly, we need to show that label reduction is actually a practically useful element of the merge-and-shrink toolbox. Although previous papers on merge-and-shrink heuristics already mentioned significant performance improvements due to label reduction, these are not a central focus of any previous experiment, and we think it is important to give solid quantitative evidence in favour of label reduction.

Secondly, while the semantic (rather than syntax-based, as in previous work) basis for exact label reduction has the advantage of being much more flexible and easier to implement than previous label reduction theory, it does carry a nontrivial computational overhead. If this overhead were so large that implementations based on the new theory performed significantly worse than ones based on the older theory, the usefulness of the new theory would be diminished.

Thirdly, a major drawback of previous label-reduction approaches are the limitations and difficulties in using them for *non-linear* merge strategies. Consequently, we are not aware of any implementations of non-linear merge strategies in the planning literature. The new theory removes these weaknesses, so it is appropriate to test it with non-linear merging.

In this section, we report on experiments that address these three aspects.

Experiment Description

Our experiments were conducted with the Fast Downward planning system (Helmert 2006), which already features the merge-and-shrink framework including the previous label reduction approach. We evaluate on all benchmarks from the International Planning Competitions for optimal planning (up to 2011) that only use language features supported by the merge-and-shrink framework (44 domains and 1396 instances in total). The experiments were performed on Intel Xeon E5-2660 CPUs running at 2.2 GHz, using a time bound of 30 minutes and a memory bound of 2 GB per run.

All planning algorithms we evaluate employ an A* search with a merge-and-shrink heuristic, which we varied along three dimensions: *label reduction method*, *merge strategy* and *shrink strategy*.

Label Reduction Methods We consider the case without label reduction (*none*), the *old* label reduction method based on the syntactic descriptions of operators (Nissim, Hoffmann, and Helmert 2011a; Helmert et al. 2014) and the *new* concept of label reduction described in this paper.

Our implementation of the new method only performs exact label reduction, combining labels whenever the Θ-combinability condition in Theorem 1 applies. Specifically, the computation proceeds as follows: whenever label reduction makes sense (after each merge or shrink step), we compute the local equivalence relations for labels in each transition system, then use these to test for Θ-combinable labels in each transition system Θ. If such labels exist, they are

combined in all transition systems, and the local equivalence relations are recomputed. The process repeats until no further Θ-combinable labels exist for any transition system Θ. Local equivalence relations are cached so that they are only recomputed from scratch if the given transition system has changed since the last computation.

Merge Strategies We consider two merge strategies. Firstly, in order to represent the state of the art, we report results for the (linear) *reverse-level* (*RL*) strategy used in previous work (Nissim, Hoffmann, and Helmert 2011a; 2011b).

However, to more fully utilize the potential of the new label reduction approach, we also evaluate it on a *non-linear* merge strategy, for which the previous label reduction approach is comparatively ill-suited and no implementations were previously available. Therefore, as a case study, we implemented the originally proposed non-linear strategy by Dräger, Finkbeiner, and Podelski (2006) from model checking, which we call the *DFP* merge strategy in the following.

Roughly speaking, the DFP merge strategy is based on the idea of preferably merging transition systems which must synchronize on labels that occur close to a goal state. We refer to the original paper by Dräger et al. (2006) for details. We remind the reader that the work of Dräger et al. preceded the concept of label reduction, so the combination of non-linear merge strategies with label reduction is novel.

Shrink Strategies We report results on shrink strategies based on bisimulation (Nissim, Hoffmann, and Helmert 2011a; Helmert et al. 2014), which set the current state of the art. Specifically, we consider a shrink strategy based on *greedy bisimulation* with no limit on transition system size (*G-N∞*) as well as shrink strategies based on (exact) *bisimulation* with different size limits N for the intermediate transition system size (*B-N10k*, *B-N50k*, *B-N100k*, *B-N200k*, *B-N∞*). For example, with $N = 10000$ (strategy B-N10k), shrinking is performed to guarantee that no intermediate transition system has more than 10,000 abstract states, while with $N = \infty$ (strategy B-N∞) there is no size bound, so that a perfect heuristic is constructed.

The *threshold* parameter (Helmert et al. 2014) was set to N for the strategies with bounded transition system size and to 1 for the unbounded ones (G-N∞ and B-N∞), following Nissim, Hoffmann, and Helmert (2011a). This configuration space includes the shrink strategies used in the *merge-and-shrink* planner that participated in IPC 2011 (Nissim, Hoffmann, and Helmert 2011b).

Experimental Results

Table 1 provides a result overview for coverage, i. e., the number of instances solved by each planner configuration within our resource bounds. The top half of the table presents results for the linear merge strategy (RL), the bottom half presents results for the non-linear DFP strategy.

Usefulness of Label Reduction Table 1 shows that planner configurations with label reduction dramatically outperform the corresponding ones without. (For readers less familiar with optimal planning, we point out that these tasks

merge/shrink strategy	none	old	new
RL-G-N∞	417	485	465
RL-B-N10k	590	624	617
RL-B-N50k	577	618	634
RL-B-N100k	560	599	**639**
RL-B-N200k	544	590	630
RL-B-N∞	257	302	302
DFP-G-N∞	415	—	465
DFP-B-N10k	597	—	622
DFP-B-N50k	565	—	**644**
DFP-B-N100k	551	—	632
DFP-B-N200k	522	—	625
DFP-B-N∞	253	—	302

Table 1: Total coverage for several merge-and-shrink configurations, using no label reduction (*none*), the previous (*old*) or the *new* label reduction. See the text for descriptions of the merge and shrink strategies. Best results for each merge strategy in bold.

	RL-B-100K			DFP-B-50K	
	none	old	new	none	new
mprime (35)	8	+6	**+15**	6	**+17**
miconic (150)	60	**+13**	**+13**	58	**+14**
gripper (20)	7	**+13**	**+13**	7	**+11**
freecell (80)	6	−2	**+13**	9	**+11**
mystery (30)	8	+1	**+8**	8	**+8**
zenotravel (20)	9	**+3**	**+3**	10	**+2**
pipesworld-tankage (50)	8	+2	**+3**	12	**+2**
nomystery-opt11-strips (20)	17	**+1**	**+1**	16	**+2**
woodworking-opt08-strips (30)	11	−1	**+1**	11	**+2**
blocks (35)	**25**	−3	−3	25	**+2**
grid (5)	1	**+2**	**+2**	1	**+1**
floortile-opt11-strips (20)	5	**+1**	**+1**	4	**+1**
rovers (40)	7	**+1**	**+1**	7	**+1**
satellite (36)	5	**+1**	**+1**	5	**+1**
scanalyzer-08-strips (30)	12	**+1**	**+1**	12	**+1**
scanalyzer-opt11-strips (20)	9	**+1**	**+1**	9	**+1**
woodworking-opt11-strips (20)	6	−1	**+1**	6	**+1**
pipesworld-notankage (50)	**14**	**±0**	**±0**	14	**+1**
sokoban-opt08-strips (30)	24	±0	**+2**	**25**	**±0**
trucks-strips (30)	6	±0	**+2**	**6**	**±0**
transport-opt11-strips (20)	6	**+1**	**+1**	**6**	**±0**
driverlog (20)	**13**	−1	−1	**12**	**±0**
Sum (**791**)	267	+39	+79	269	+79
Remaining domains (**605**)	**293**	**±0**	**±0**	**296**	**±0**
Sum (**1396**)	560	599	**639**	565	**644**

Table 2: Per-domain coverage. Columns 2–4 compare no (*none*), *old* and *new* label reduction for the linear merge strategy *reverse level* (RL) in its best configuration (RL-B-100K). Columns 5–6 compare no (*none*) and *new* label reduction for the non-linear DFP merge strategy in its best configuration (DFP-B-50K). For *old* and *new*, the columns show increase/decrease in coverage compared to *none*. Domains where label reduction showed no increase/decrease in coverage are omitted. The best results for the given merge strategy are highlighted in bold.

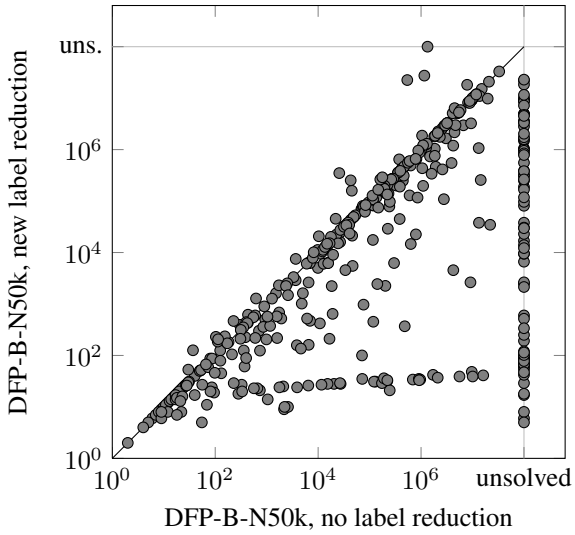


Figure 2: Number of expanded states for DFP-B-N50k: no label reduction vs. new label reduction.

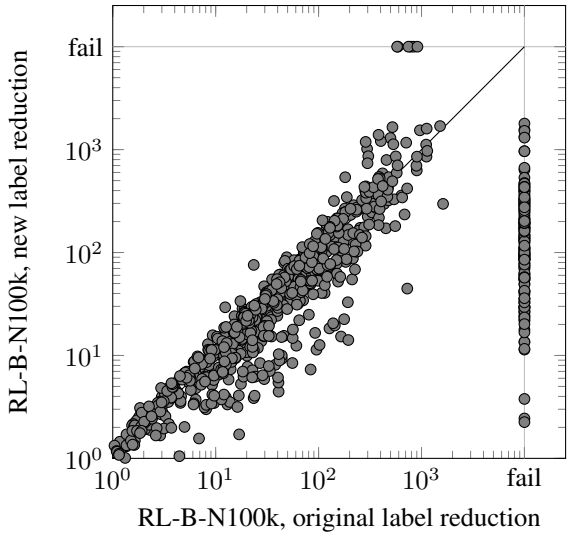


Figure 3: Construction time (in seconds) for RL-B-N100k: old label reduction vs. new label reduction. Almost all failures are due to running out of memory.

tend to scale exponentially in difficulty, so that even small improvements in coverage tend to be very hard to obtain.)

Table 2 shows detailed coverage results for the individual planning domains in the benchmark set for the best-performing shrink strategies for each merge strategy. The table shows that label reduction is very useful across the board, over a wide range of domains.

For the linear RL merge strategy, the new label reduction approach increases coverage in 19 domains compared to the baseline where no labels are reduced, while decreasing coverage in 2 domains. For the non-linear DFP merge strategy, label reduction increases coverage in 18 domains and decreases it in none.

To provide another detailed view, Figure 2 shows the number of expanded states with the strongest configuration, DFP-B-N50k, with and without label reduction. The figure plots the results without label reduction against the results with our new label reduction approach, over all instances in the benchmark suite. The figure clearly shows the significant impact that label reduction has on performance in many cases.

Old vs. New Label Reduction Method Focusing on the comparison between the old and new label reduction method with a linear merge strategy (top half of Table 1), we see that despite the larger effort involved in determining reducible labels, the results are in fact quite a bit *better* with new label reduction compared to the old technique. In particular, the best overall result of 639 solved tasks (RL-B-100k) is considerably higher than the best result with the previous state of the art (624 solved tasks with RL-B-10K and the old label reduction method).

There are two shrink strategies that show the opposite trend, namely the ones that tend to compute the simplest abstractions among the six strategies we consider: greedy bisimulation (RL-G-N∞) and exact bisimulation with the smallest size bound (RL-B-N10k). One possible explanation for this behaviour is that for the shrink strategies that compute more complex abstractions, the additional label reductions afforded by the new method are critical for computing the merge-and-shrink abstraction within the given limits for time and especially memory. With the shrink strategies that compute simpler abstractions, on the other hand, memory for computing the abstraction is less of a concern, and the new label reduction method suffers from the higher computational cost for determining combinable labels.

This interpretation is supported by Figure 3, which compares the time to construct the abstraction heuristic for the old and new label reduction method for the strategy RL-B-N100k. The new strategy tends to construct abstractions faster and runs out of memory far less frequently. Figure 4 compares state expansions for the same configurations, showing that the heuristics are similarly informative in both cases, and it is mainly the ability to complete the computation of the abstraction (see Figure 3) that makes the difference between the old and new label reduction here.

In the case of perfect bisimulations (RL-B-N∞), there is no difference in coverage between the two label reduction methods for a different reason: unless the given planning task exhibits significant amounts of symmetry, unrestricted bisimulation tends to exhaust the available memory very quickly, and hence the perfect abstraction heuristic is either computed quickly or not at all. In all cases not solved by the perfect bisimulation approaches, this is due to running out of memory while computing the abstraction.

Non-Linear Merge Strategy Shifting attention to the results for the non-linear DFP merge strategy (bottom half of Table 1), we see that the results with the new label reduction method are excellent. In particular, the best configuration (DFP-B-N50k) solves 644 tasks, again setting a new best re-

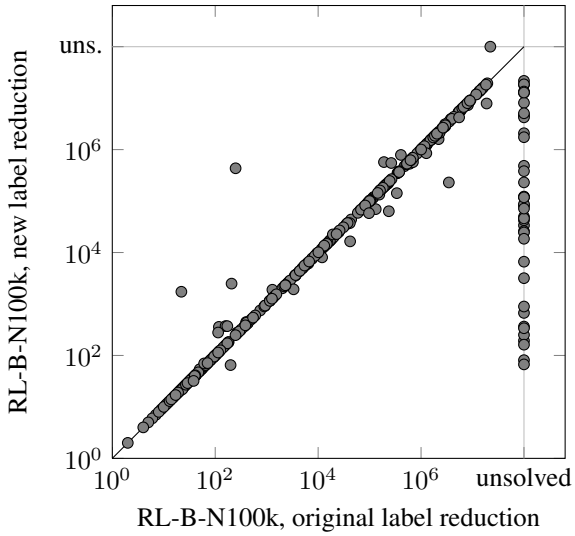


Figure 4: Number of expanded states for RL-B-N100k: old label reduction vs. new label reduction.

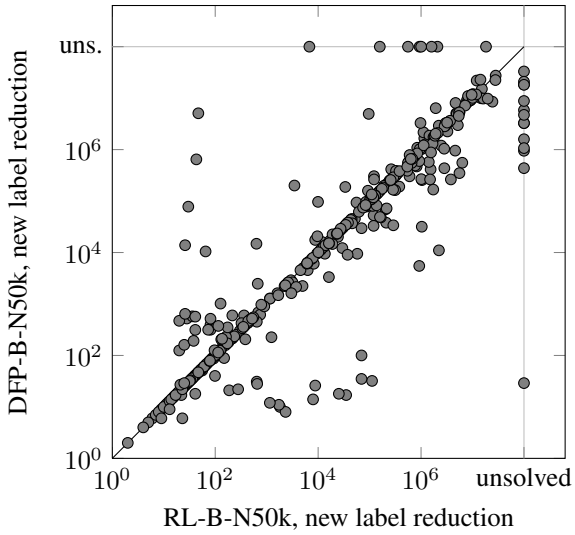


Figure 5: Number of expanded states for RL-B-N50k vs. DFP-B-N50k, both using new label reduction.

sult (compared to 639 solved by RL-B-N100k, also with our new label reduction method).

Generally speaking, the non-linear merge strategy appears to benefit even more from label reduction than the linear one on average. One possible explanation for this observation is that non-linear merge strategies involve more complex products (merges) than linear ones, and hence benefit more from label reduction collapsing multiple parallel transitions into one. In linear merge strategies, at least one of the merged transition systems is always atomic, and atomic transition systems tend to have a comparatively low density of transitions. An alternative possibility is that label reduction interacts favourably with the DFP merge strategy, which – unlike merge strategies previously considered in planning – takes the labels into account directly in order to decide which transition systems to merge next.

Figure 5 compares the number of state expansions for the linear and non-linear merge strategy on an otherwise identical configuration (shrink strategy B-N50k, new label reduction). The comparison shows that the two merge strategies are quite complementary, with both strategies greatly outperforming each other on a significant number of instances.

Conclusions

We have introduced a general theory of label reduction that addresses several drawbacks in the previous development of this topic. Compared to the previous theory, the new theory of label reduction is easier to understand, easier to reason about, and more general.

Under the new theory, label reduction can always be safely applied. Moreover, we have provided efficiently checkable necessary and sufficient criteria for label reduction to be *exact*, i. e., preserve all relevant information. The new theory allows identifying more cases where exact label reduction is possible, leading to improved performance of merge-and-shrink heuristics based on label reduction.

Unlike the previous theory of label reduction, the new theory allows for a straight-forward application of non-linear merge strategies. We conducted the first experiments of this kind by adapting the originally proposed non-linear merge strategy from model checking to planning. In the future, we hope that the development of strong non-linear merge strategies can further increase the scalability of merge-and-shrink heuristics.

Another possible direction for future work is the exploration of *inexact* label reduction. Inexact label reduction is a general abstraction method just like shrinking, and similar intuitions to those that have guided the development of state-of-the-art shrink strategies could be used to develop useful inexact label reduction methods. For example, one might try to abstract a factored transition system by combining labels that only occur far away from goal states, similarly to the way that current shrink strategies prefer to combine abstract states that are far away from the goal.

Acknowledgments

We thank the anonymous reviewers for their comments, which helped improve the paper. This work was supported by the Swiss National Science Foundation (SNSF) as part of the project "Abstraction Heuristics for Planning and Combinatorial Search" (AHPACS).

References

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

Dräger, K.; Finkbeiner, B.; and Podelski, A. 2006. Directed model checking with distance-preserving abstractions. In Valmari, A., ed., *Proceedings of the 13th International SPIN*

Workshop (SPIN 2006), volume 3925 of *Lecture Notes in Computer Science*, 19–34. Springer-Verlag.

Dräger, K.; Finkbeiner, B.; and Podelski, A. 2009. Directed model checking with distance-preserving abstractions. *International Journal on Software Tools for Technology Transfer* 11(1):27–37.

Helmert, M.; Haslum, P.; Hoffmann, J.; and Nissim, R. 2014. Merge-and-shrink abstraction: A method for generating lower bounds in factored state spaces. *Journal of the ACM*. In press.

Helmert, M.; Haslum, P.; and Hoffmann, J. 2007. Flexible abstraction heuristics for optimal sequential planning. In Boddy, M.; Fox, M.; and Thiébaux, S., eds., *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling (ICAPS 2007)*, 176–183. AAAI Press.

Helmert, M.; Haslum, P.; and Hoffmann, J. 2008. Explicit-state abstraction: A new method for generating heuristic functions. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI 2008)*, 1547–1550. AAAI Press.

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

Nissim, R.; Hoffmann, J.; and Helmert, M. 2011a. 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 2011)*, 1983–1990.

Nissim, R.; Hoffmann, J.; and Helmert, M. 2011b. The Merge-and-Shrink planner: Bisimulation-based abstraction for optimal planning. In *IPC 2011 planner abstracts*, 106–107.