Improving Cost-Optimal Domain-Independent Symbolic Planning

Peter Kissmann and **Stefan Edelkamp**
TZI Universität Bremen, Germany
{kissmann, edelkamp}@tzi.de

Abstract

Symbolic search with BDDs has shown remarkable performance for cost-optimal deterministic planning by exploiting a succinct representation and exploration of state sets. In this paper we enhance BDD-based planning by applying a combination of domain-independent search techniques: the optimization of the variable ordering in the BDD by approximating the linear arrangement problem, pattern selection for improved construction of search heuristics in form of symbolic partial pattern databases, and a decision procedure for the amount of bidirection in the symbolic search process.

Introduction

Symbolic search with Binary Decision Diagrams (Bryant 1986), BDDs for short, has shown considerable success for *cost-optimal* planning, so that our planner GAMER (Edelkamp and Kissmann 2009) won the corresponding track in the international planning competition IPC 2008.

While *explicit-state search* is concerned with the expansion of single states and the calculation of successors of a single state, in *symbolic search* (McMillan 1993) with BDDs sets of states are handled. E. g., the assignments satisfying a Boolean formula can be seen as sets of states. Similarly, we can represent any state vector (in binary representation) as a Boolean formula with one satisfying assignment. To achieve this, we represent any state as a conjunction of (binary) variables. Thus, a set of states is the disjunction of such a conjunction of variables, so that we can easily represent it in form of a BDD.

This paper contains three contributions to improve our existing symbolic search technology to come up with a new version of GAMER that clearly outperforms the 2008 version.

1. The variable ordering has a significant effect on the size of the BDD representation for many Boolean functions (Bryant 1986), but the automated inference of a good ordering has not been studied for symbolic planning. As the problem of optimizing the ordering in a BDD is hard in general and existing reordering techniques show performance drawbacks in practice, this paper follows an alternative approach based on exploiting causal dependencies of state variables.
2. The BDD exploration on state sets can be improved by including heuristics into the search process. Based on projecting away all but a selected *pattern* of state variables, in backward search symbolic versions of pattern databases (Culberson and Schaeffer 1998) can be constructed. In this paper, we provide an alternative realization for greedy pattern selection based on constructing symbolic pattern databases.
3. While symbolic backward search is conceptually simple and often effective in some domains, due to a large number of invalid states the sizes of the BDDs increase too quickly to be effective for a bidirectional exploration of the original state space. For this case we integrate a simple decision procedure to allow backward search in abstract state space search only.

This paper is structured as follows. First, we introduce deterministic planning with finite domain variables and the causal dependencies among them. Next, we turn to the linear arrangement problem that we greedily optimize to find a good BDD variable ordering. Then, we present symbolic search strategies including symbolic BFS, symbolic single-source shortest-path and symbolic A* as needed for step- and cost-optimal planning. We continue with a new automated construction of search heuristics in form of symbolic pattern databases. Then, we consider a decision procedure of whether or not to start backward search. Finally, we evaluate the new version of GAMER on the full set of problems from IPC 2008.

Planning and Causal Graphs

In (deterministic) planning we are confronted with a description of a planning problem and are interested in finding a good solution (a good plan) for this problem. Such a plan is a set of actions whose application transforms the initial state to one of the goal states. In *propositional* planning, states are sets of Boolean variables. However, propositional encodings are not necessarily the best state space representations for solving propositional planning problems. A multi-valued variable encoding is often better. It transforms a propositional planning task into an *SAS$^+$ planning* task (Bäckström and Nebel 1995).

Definition 1 (Cost-Based SAS$^+$ Planning Task). *A cost-based SAS$^+$* planning task $\mathcal{P} = \langle \mathcal{S}, \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{T}, c \rangle$ *consists of a set* $\mathcal{V} = \{v_1, \dots, v_n\}$ *of state variables for states in* $\mathcal{S}$, *where each $v \in \mathcal{V}$ has finite domain D_v. Some sets of states are* partial assignments*, i. e., functions s over $\mathcal{V}$, such that $s(v) \in D_v$, if $s(v)$ is defined. The initial state $\mathcal{I}$ is a* full assignment*, i. e., a total function over $\mathcal{V}$. The goal states $\mathcal{T}$ are specified in form of a* partial assignment*. An action $a \in \mathcal{A}$ is given by a pair $\langle pre, eff \rangle$ of preconditions and effects, where pre and eff are partial assignments. The application of action a in a state $S \in \mathcal{S}$ results in the successor state S', i. e., $a(S) = S'$. The cost function $c : \mathcal{A} \mapsto \mathbb{N}_0^+$ specifies the cost of each action.*

Definition 2 (Plan and Cost of a Plan). *A* plan *P is a sequence of actions $(a_1, a_2, \dots, a_n) \in \mathcal{A}^n$ with $a_n(a_{n-1}(\dots a_1(\mathcal{I})\dots)) \in \mathcal{T}$. The* total cost *$\mathcal{C}(P)$ of a plan P is the sum of the costs of all actions within P, i. e., $\mathcal{C}(P) := c(a_1) + c(a_2) + \dots + c(a_n)$.*

Definition 3 (Optimal Plan). *A plan P is called* optimal*, if there is no plan P' with $\mathcal{C}(P') < \mathcal{C}(P)$.*

The following definition specifies the dependencies among the variables, resulting in a causal graph

Definition 4 (Causal Graph). *The* causal graph *of an SAS$^+$ planning task $\mathcal{P}$ with variable set $\mathcal{V}$ is a directed graph (V, E) with $V = \mathcal{V}$ and $(u, v) \in E$ if and only if $u \neq v$ and there is an action $\langle pre, eff \rangle \in \mathcal{A}$, such that $eff(v)$ is defined and $pre(u)$ or $eff(u)$ is defined. This implies that an edge is drawn from one variable to another if the change of the second variable is dependent on the current assignment of the first variable.*

We use what we call a *symmetric causal graph*, i. e., a causal graph with the dependencies applied in both directions, so that we arrive at a symmetrical relation.

Optimal Linear Variable Arrangement

In short, BDDs are memory-efficient data structures used to represent Boolean functions as well as to perform set-based search. A BDD is a directed acyclic graph with one root and two terminal nodes, the 0- and the 1-sink. Each internal node corresponds to a binary variable and has two successors, one representing that the current variable is *false* and the other representing that it is *true*. The assignment of the variables derived from any path from the root to the 1-sink corresponds to an assignment for which the represented function evaluates to *true*.

The variable ordering problem in a BDD is co-NP-complete (Bryant 1986), so that optimal algorithms are practically infeasible even for small-sized planning problems. Dynamic reordering algorithms as provided in current BDD packages require sifting operations on existing BDDs and are often too slow to be effective in planning. One reason is that they do not exploit any knowledge on variable dependencies.

Thus, we decided to approximate another optimization problem to find a good variable ordering without BDDs.

Definition 5 (Optimal Linear Arrangement). *Given a weighted graph $G = (V, E, c)$ on n vertices, in the* optimal linear arrangement *problem the goal is to find a permutation $\pi : V \to \{1, \dots, n\}$ that minimizes $\sum_{e=(u,v)\in E} c(e) \cdot |\pi(v) - \pi(u)|$.*

We set the weights $c(e)$ to 1, so that we actually solve the version called *simple optimal linear arrangement*. This problem is (still) NP-hard (via a reduction to Max-Cut and Max-2-Sat (Garey, Johnson, and Stockmeyer 1976)). It is also hard to provide approximations within any constant factor (Devanur et al. 2006).

There are different known generalizations to the optimal linear arrangement problem. The *quadratic assignment* problem $\sum_{e=(u,v)\in E} c(e) \cdot c'(\pi(u), \pi(v))$ for some weight c' is a generalization, which includes TSP as one case (Lawler 1963). Here, we consider the optimization function

$$\Phi(\pi) = \sum_{(u,v)\in E} d(\pi(u), \pi(v)),$$

subject to the metric $d(x, y) = ||x - y||_2^2 = \sum_{i=1}^n (x_i - y_i)^2$.

In our case G is the symmetric causal graph, i. e., the nodes in V are the multi-valued variables in the SAS$^+$ encoding and the edges in E reflect the causal dependencies.

As the problem is complex we apply a greedy search procedure for optimization with two loops. The outer loop calculates a fixed number ρ of random permutations, while the inner loop performs a fixed number η of transpositions. To increase the values of ρ and η we decided to incrementally compute $\Phi(\pi) = \sum_{(u,v)\in E} d(\pi(u), \pi(v))$ as follows. Let $\tau(i, j)$ be the transposition applied to the permutation π to obtain the new permutation π', and let x and y be the variables associated to the indices i and j in π. We have

$$\begin{aligned}\Phi(\pi') &= \sum_{(u,v)\in E} d(\pi'(u), \pi'(v)) \\ &= \Phi(\pi) - \sum_{(w,x)\in E} d(\pi(w), i) - \sum_{(w,y)\in E} d(\pi(w), j) \\ &\quad + \sum_{(w,x)\in E} d(\pi(w), j) + \sum_{(w,y)\in E} d(\pi(w), i).\end{aligned}$$

The complexity for computing $\Phi(\pi')$ reduces from quadratic to linear time for the incremental computation. This has a considerable impact on the performance of the optimization process as it performs millions of updates in a matter of seconds instead of minutes (depending on the symmetric causal graph and the domain chosen).

Now that the variable ordering has been fixed we consider how to optimally solve deterministic planning problems.

Step-Optimal Symbolic Planning

To calculate the set of successors of a given set of states, the $image$ operator is used. Though we are able to find efficient variable orderings for many problems, we cannot expect to be able to calculate exponential search spaces in polynomial time. This comes from the fact that the calculation of the image is NP-complete (McMillan 1993). It is, however, not essential to compute one image for all actions in common. Instead, we apply the $image$ operator to one action after the other and calculate the union of these afterwards.

Using the $image$ operator the implementation for a symbolic breadth-first search (BFS) is straight-forward. All we need to do is to apply $image$ first to the initial state and afterwards to the last generated successor states. The search ends when a fix-point is reached, i. e., when the application of $image$ does not yield any new states.

For the search in backward direction we use the $pre\text{-}image$ operator, which is similar to $image$ but calculates all predecessors of a given set of states.

Given the $image$ and $pre\text{-}image$ operators we also devise a symbolic bidirectional breadth-first search. For this we start one search in forward direction (using $image$) at the initial state and another in backward direction (using $pre\text{-}image$) at the terminal states. Once the two searches overlap we stop and generate an optimal solution.

Cost-Optimal Symbolic Sequential Planning

Handling action costs is somewhat more complicated. Now, an optimal plan is no longer one of minimal length but rather one with minimal total cost, i. e., the sum of the costs of all actions within the plan needs to be minimal.

Dijkstra's single-source shortest-paths search (1959) is a classical algorithm used for state spaces with edge weights. In its normal form it consists of a list of nodes denoted with their distance to the start node. Initially, the distance for the start node is 0 and that for the other nodes ∞.

It chooses the node u with minimal distance, removes it from the list and updates the distances of the neighboring nodes. For each neighbor v it recalculates its distance to the start node to $\min\left(d\left(v\right), d\left(u\right) + c\left(u, v\right)\right)$, with $c\left(u, v\right)$ being the cost of edge $\left(u, v\right)$ and $d\left(w\right)$ the distance of node w to the start node calculated so far.

For the symbolic version of this algorithm we need a way to access the states having a certain distance from $\mathcal{I}$. For this typically a priority queue is used. As we are concerned only with discrete action costs we can partition the priority queue into buckets, resulting in an *open list* (Dial 1969). In this list we store all the states that have been reached so far in the bucket representing the distance from $\mathcal{I}$ that they have been found in.

In case of zero-cost actions we calculate a fix-point for the current bucket, and perform a BFS using only the zero-cost actions. The resulting states correspond to the application of one non-zero-cost action followed by a number of zero-cost actions. For duplicate elimination, we might also keep a BDD storing all the states we have already expanded.

A* (Hart, Nilsson, and Raphael 1968) corresponds to Dijkstra's algorithm with the new costs $\hat{c}\left(u, v\right) = c\left(u, v\right) - h\left(u\right) + h\left(v\right)$. Both algorithms, Dijkstra's algorithm with reweighted costs and A*, perform the same, if the heuristic function is consistent, i. e., if for all $a = \left(u, v\right) \in \mathcal{A}$ we have $h\left(u\right) \leq h\left(v\right) + c\left(u, v\right)$.

Efficient symbolic adaptations of A* with consistent heuristics are SetA* by Jensen, Bryant, and Veloso (2002) and BDDA* by Edelkamp and Reffel (1998). While for Dijkstra's algorithm a bucket list was sufficient (in the case of no zero-cost actions), here we need a two-dimensional matrix (cf. Figure 1). One dimension represents the distance from $\mathcal{I}$ (the g value), the other one the heuristic estimate on the distance to a goal (the h value).

A* expands according to $f = g + h$, which corresponds to a diagonal in the matrix. If the heuristic is consistent we will never have to reopen an f diagonal that we already expanded. Furthermore, as we do not allow negative action costs, the g value will never decrease. So, for each f diagonal we start at the bucket with smallest g value, expand this, go to the next one on the same f diagonal, until either the diagonal has been completely expanded or we find a goal state $t \in \mathcal{T}$. In the first case we go on to the next f diagonal, in the second case we are done.

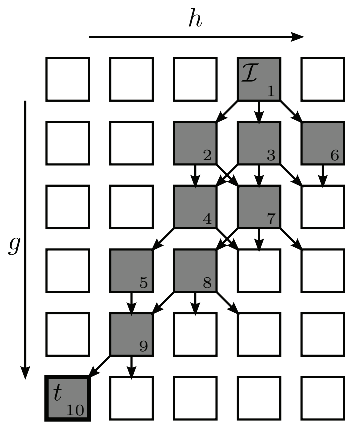

Figure 1: The working of BDDA*. The buckets that are expanded in the course of the search are shaded in gray, the numbers in them denote the order of their expansion and the arrows point to the buckets containing successors of the current one.

For the case of zero-cost actions, things again get a bit more difficult, as we – similar to the symbolic version of Dijkstra's algorithm – need to store a list of BDDs in each bucket of the matrix to keep track of the different layers of the zero-cost BFS.

A question that still remains is how we can come up with a consistent symbolic heuristic. For this, in the following we will present how to compute symbolic pattern databases.

Symbolic Planning Pattern Databases

Pattern databases have originally been proposed by Culberson and Schaeffer (1998). They were designed to operate in the abstract space of a problem.

Definition 6 (Restriction Function). *Let $\mathcal{V}' \subseteq \mathcal{V}$. For a set of variables $v \in \mathcal{V}$, a* restriction function *$\phi_{\mathcal{V}'} : 2^{\mathcal{V}} \mapsto 2^{\mathcal{V}'}$ is defined as the projection of S to $\mathcal{V}'$ containing only those variables that are also present in $\mathcal{V}'$, while those in $\mathcal{V} \setminus \mathcal{V}'$ are removed. $\phi_{\mathcal{V}'}\left(S\right)$ is also denoted as $S|_{\mathcal{V}'}$.*

Definition 7 (Abstract Planning Problem). *Let $\mathcal{P} = \langle \mathcal{S}, \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{T}, c \rangle$ be a planning problem and $\mathcal{V}' \subseteq \mathcal{V}$ a set of variables. An* abstract planning problem *$\mathcal{P}|_{\mathcal{V}'} = \langle \mathcal{S}|_{\mathcal{V}'}, \mathcal{V}', \mathcal{A}|_{\mathcal{V}'}, \mathcal{I}|_{\mathcal{V}'}, \mathcal{T}|_{\mathcal{V}'}, c \rangle$ is $\mathcal{P}$ restricted to $\mathcal{V}'$, where $\mathcal{S}|_{\mathcal{V}'} = \{s|_{\mathcal{V}'} \mid s \in \mathcal{S}\}$, and $\mathcal{A}|_{\mathcal{V}'} = \{\langle pre|_{\mathcal{V}'}, eff|_{\mathcal{V}'} \rangle \mid \langle pre, eff \rangle \in \mathcal{A} \wedge eff|_{\mathcal{V}'} \neq \emptyset\}$*

In other words, the abstraction of a planning problem results in a (typically smaller) planning problem where certain variables (those in $\mathcal{V} \setminus \mathcal{V}'$) are ignored.

Definition 8 (Pattern Database). *A* pattern database *(PDB)* Φ *for an abstract planning problem* $\mathcal{P}_{\mathcal{V}'} = \langle \mathcal{S}|_{\mathcal{V}'}, \mathcal{V}', \mathcal{A}|_{\mathcal{V}'}, \mathcal{I}|_{\mathcal{V}'}, \mathcal{T}|_{\mathcal{V}'}, c\rangle$ *is a set of pairs* (d, s) *where* $d \in \mathbb{N}_0^+$ *denotes the minimal distance of the state* $s \in \mathcal{S}|_{\mathcal{V}'}$ *to one of the abstract goal states.*

The construction of symbolic PDBs (Edelkamp 2002) performs symbolic Dijkstra search. Starting at the goal state $\mathcal{T}|_{\mathcal{V}'}$ it operates in backward direction until all states are inserted into the database, which consists of a vector of BDDs, the BDD within each bucket of this vector representing the abstract states that have a corresponding distance to an abstract goal state.

If the abstract spaces are too large, the complete database calculation might take too long. Thus, we probably do not want the full information we could get. Therefore, we consider combining PDBs with *perimeter search* (Dillenburg and Nelson 1994).

Felner and Ofek (2007) propose two algorithms combining perimeter search and PDBs. The first one, called *simplified perimeter PDB* (SP_PDB) uses a classical PDB and a perimeter for a distance of d. This perimeter is used to correct too low heuristic estimates from the PDB up to the depth bound of $d + 1$.

The second approach, which they called *perimeter PDB* (P_PDB), performs a perimeter search up to a depth of d as a first step, followed by a PDB calculation starting at the states on the perimeter. The heuristic estimates result from the PDB and give estimates on the distance to the perimeter. The forward search is then performed using these heuristic estimates until a state on the perimeter is reached and chosen for expansion.

The idea of *partial PDBs* is due to Anderson, Holte, and Schaeffer (2007). A partial PDB is a PDB that is not calculated completely. Similar to perimeter search, it is created up to a maximal distance d to an abstract goal state. For all states that have been reached during the PDB construction process the heuristic estimate is calculated according to the PDB, while for states not within the PDB the estimate can be set to $d + 1$ $d + 1$, if all states up to distance d are stored in the PDB.

An adaptation to symbolic search is again straightforward (Edelkamp and Kissmann 2008). All we need to do is perform symbolic Dijkstra search starting at the goal states $\mathcal{T}|_{\mathcal{V}'}$ up to a maximal distance d and then stop the PDB creation process. In our implementation we do not set the value of d explicitly but generate the PDB until a certain timeout is reached. The maximal distance of expanded states. As we are concerned with symbolic search and have one BDD for all states sharing the same distance to the goal states, all unexpanded states have a distance that is higher than d, so that we assign all such states a distance of $d + 1$.

Automated Pattern Selection

The automated selection of patterns for PDBs has been considered by Haslum et al. (2007). For one PDB the approach greedily constructs a pattern $\mathcal{V}'$ by adding one variable $v \in \mathcal{V}$ of the unchosen ones at a time. For the greedy selection process the quality of the unconstructed PDBs $\mathcal{V}' \cup \{v\}$ is evaluated by drawing and abstracting n random samples in the original state space. These subproblems are solved using heuristic search wrt. the already constructed PDB $\mathcal{V}'$. If v is fixed then the PDB for $\mathcal{V}' \cup \{v\}$ is constructed and the process iterates. The decision criterion for selecting the next candidate variable is the search tree prediction formula for iterative-deepening A* proposed by Korf, Reid, and Edelkamp (2001).

Similarly, our incremental symbolic PDB is constructed by greedily extending the variable set that is mentioned in the goal using backward search. However, we neither use sampling nor heuristics but construct the symbolic PDBs for all candidate variables. Moreover, the result is a single PDB, not a selection as in Haslum et al. (2007). If the pattern $\mathcal{V}' \subset \mathcal{V}$ is chosen, we construct the symbolic PDBs for $\mathcal{V}' \cup \{v\}$ for all variables $v \in \mathcal{V} \setminus \mathcal{V}'$ if the causal graph contains an edge from v to any of the variables in $\mathcal{V}'$. To select among the candidate patterns we then compute the mean heuristic value in the PDBs of the respective candidate variable by using model counting (sat-count as proposed by Bryant (1986)), which is an operation linear in the size of the constructed symbolic PDB. After we are done with testing the insertion of all variables we actually add all those that achieved the best mean heuristic value (if that is greater than the mean heuristic value of $\mathcal{V}'$) and start the process over.

In contrast to explicit search, backward symbolic search in the according abstract state spaces is often possible even if the set of goal states is large. If the time reserved for construction is exhausted we use the PDB constructed so far as a partial PDB.

Bidirection

We observed that in some (but rare) domains unabstracted symbolic backward search is rather complex and limited to very few backward images that dominate the search process and require lots of resources in time and space. The reason is that the goal specification is partial, and in these domains backward search generates lots of states unreachable in forward direction, which results in tremendous work.

Fortunately, for most example problems we investigated this effect can be detected in the first backward iteration, resulting in a manifold increase of the time to compute the first pre-image. Hence, we perform one image from $\mathcal{I}$ and one pre-image from $\mathcal{T}$ and impose a threshold α on the time ratio of the two. If the ratio exceeds α we turn off bidirectional search (for perimeter database construction in the original space) and insist on partial PDB construction in the abstract space (as in the previous section), where due to the smaller size of the state vector the effect of backward search for the exploration is less prominent.

Experiments

We performed all the experiments on one core of a desktop computer (Intel $i7$ CPU with 2.67 GHz and 24 GB RAM).

To evaluate the benefit of each of the approaches proposed in this paper we compare all possibilities of them. The underlying system in all cases is our planner GAMER, which participated in IPC 2008. The only difference is that we use a map instead of a matrix for BDDA* (as proposed before (Edelkamp and Kissmann 2009)) to handle large action costs as present in the PARC-PRINTER domain and slightly adapted the start script, so that it stops the backward search after half the available time (real-time, while in the competition we used CPU time).

The different versions are denoted by sub- and superscript. The versions using the perimeter database without abstraction are denoted by subscript p, those using abstraction by subscript a and those using the bidirectional choice by subscript b. The versions using variable reordering are denoted by superscript r, those without it use the default ordering from the 2008 version, i. e., a lexicographical ordering, which we used hoping that correlated predicates often have the same name. Thus, the version of IPC 2008 is GAMER$_p$, while the version we submitted for IPC 2011 is GAMER$_b^r$, which includes all proposed improvements.

For further comparison we took the baseline planner BASE developed by the organizers of IPC 2008, which performs Dijkstra search (A* with $h = 0$) and actually was able to solve one more problem than GAMER in the course of the competition[1]. Another competitor is our planner GPUPLAN, which utilizes the calculation power of modern GPUs to increase the speed for successor generations (Sulewski, Edelkamp, and Kissmann 2011).

The setting is similar to the competition in that we use a timeout of 30 minutes, but without a memory limit (i. e., they can use up to 24 GB), as we believe the old limit of 2 GB to be outdated. We evaluated the planners on all domains of the sequential optimal track of IPC 2008, which consist of 30 instances each and contain non-uniform action costs. For domains having more than one description we use the one for each planner that it works best on.

The results in numbers of solved instances are depicted in Table 1. From that we see that GAMER$_b^r$ beats all other planners. The biggest advantage comes with the reordering. This helps especially in the OPENSTACKS and WOODWORKING domains. The abstraction helps in the SOKOBAN and WOODWORKING domains, increasing the number of solved instances by up to four, though in the ELEVATORS domain it is counter-productive and decreases the number of solved instances by up to four. Finally, the bidirectional choice often works good (in ELEVATORS and TRANSPORT) but not yet optimal (in SOKOBAN and WOODWORKING it did not always make the right choice).

In comparison to the two explicit-state planners we see that GAMER$_b^r$ clearly outperforms them in the total number of solved instances. In four domains it can find fewer solutions, but in all cases the number is smaller by at most three. On the other hand, in ELEVATORS and WOODWORKING it is a lot better than GPUPLAN and BASE, with 13 additional solutions in the latter, and in OPENSTACKS it also finds seven solutions that BASE did not find. As GAMER is able to find all 30 solutions it seems plausible that the advantage would have been even greater if there were more problems to solve in that domain.

[1]For detailed competition results see `http://ipc.informatik.uni-freiburg.de/Results`.

Comparing the runtimes of all the planners (Table 2) is somewhat critical, as GPUPLAN and BASE perform no backward search, while GAMER performs it for up to half the available time (i. e., up to 900 seconds). Thus, for all the instances where the backward search is not trivial and does not finish within the time limit, the runtime is dominated by it. Comparing the different versions of GAMER shows that in the SCANALYZER domain the reordering is counterproductive, while in all other domains it helps decrease the average runtime. Using abstraction decreases the time especially in SOKOBAN, but increases it in PARC-PRINTER and WOODWORKING. The latter is rather unexpected as the abstraction helped solve the most instances.

Katz and Domshlak (2010) propose the use of A* with a fork-decomposition heuristic. Though they performed their experiments on a different machine with a memory limit of only 2 GB, we still believe that GAMER$_b^r$ will solve more instances of the IPC 2008 domains, as their approach beats GAMER$_p$ by only four instances.

We also experimented with the net-benefit version of GAMER. Of the proposed improvements only the variable reordering can be applied. On the five domains of IPC 2008 we found 100 plans (using the same settings as before) compared to 95 by the version without reordering. An approach proposed by Keyder and Geffner (2009) is to compile away the soft goals to arrive at a propositionsl cost-optimal planning problem. In their article they compared their approach with GAMER, and beat it by seven instances; as that was the competition version and we only recently found some bug whose removal immediately led to six additional solutions, the new version will likely beat their planner again.

Conclusion

In this paper we pushed the envelope for symbolic planning with action costs and with soft constraints. Even though each applied technique is rather fundamental or aligns with ideas that have been applied to explicit-state planning, the combined impact with respect to the number of solved benchmarks is remarkable. The improvement wrt. GAMER across the domains for minimizing action costs is statistically significant.

In the future we will refine the variable ordering heuristic for the numerical fluents. Also, experimenting with more general abstraction schemes and selection algorithms is on our research agenda. Furthermore, we are searching for a method for finding and using multiple PDBs effective in most benchmark domains.

Acknowledgments

Thanks to Deutsche Forschungsgesellschaft (DFG) for support in the project ED 74/11-1.

References

Anderson, K.; Holte, R.; and Schaeffer, J. 2007. Partial pattern databases. In *SARA*, 20–34.

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

Table 1: Number of solved instances for each domain and each planner.

Domain	GPUPlan	Base	Gamer$_p$	Gamer$_a$	Gamer$_b$	Gamer$_p^r$	Gamer$_a^r$	Gamer$_b^r$
Elevators (30)	19	17	**24**	21	**24**	**24**	20	**24**
Openstacks (30)	29	23	22	22	22	**30**	**30**	**30**
Parc-Printer (30)	9	11	11	**12**	**12**	11	11	11
Peg Solitaire (30)	**29**	28	25	25	25	27	27	27
Scanalyzer (30)	**12**	**12**	9	10	10	9	9	9
Sokoban (30)	23	**24**	19	23	20	21	**24**	22
Transport (30)	**12**	11	11	10	11	11	10	11
Woodworking (30)	9	9	14	17	14	22	**24**	22
Total (240)	142	135	135	140	138	155	155	**156**

Table 2: Average runtimes (in seconds) for problems solved by all planners with the fastest version of Gamer highlighted in each domain.

Domain	GPUPlan	Base	Gamer$_p$	Gamer$_a$	Gamer$_b$	Gamer$_p^r$	Gamer$_a^r$	Gamer$_b^r$
Elevators	73	190	816	865	816	**733**	814	734
Openstacks	26	51	554	560	554	**215**	292	**215**
Parc-Printer	161	13	**17**	97	65	20	119	77
Peg Solitaire	6	5	978	910	978	912	**844**	912
Scanalyzer	28	58	**36**	**36**	**36**	93	93	94
Sokoban	48	14	883	149	776	843	**118**	842
Transport	30	6	269	302	267	**260**	292	**260**
Woodworking	97	185	9	141	10	**6**	45	**6**
Total	48	60	583	490	571	491	**412**	496

Bryant, R. E. 1986. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Computers* 35(8):677–691.

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

Devanur, N. R.; Khot, S.; Saket, R.; and Vishnoi, N. K. 2006. Integrality gaps for sparsest cut and minimum linear arrangement problems. In *STOC*, 537–546.

Dial, R. B. 1969. Shortest-path forest with topological ordering. *Commun. ACM* 12(11):632–633.

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

Dillenburg, J. F., and Nelson, P. C. 1994. Perimeter search. *Artif. Intell.* 65(1):165–178.

Edelkamp, S., and Kissmann, P. 2008. Partial symbolic pattern databases for optimal sequential planning. In *KI*, volume 5243 of *LNCS*, 193–200. Springer.

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

Edelkamp, S., and Reffel, F. 1998. OBDDs in heuristic search. In *KI*, volume 1504 of *LNCS*, 81–92. Springer.

Edelkamp, S. 2002. Symbolic pattern databases in heuristic search planning. In *AIPS*, 274–283. AAAI Press.

Felner, A., and Ofek, N. 2007. Combining perimeter search and pattern database abstractions. In *SARA*, 155–168.

Garey, M. R.; Johnson, D. S.; and Stockmeyer, L. 1976. Some simplified NP-complete graph problems. *Theoret. Comput. Sci.* 1(3):237–267.

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

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

Jensen, R. M.; Bryant, R. E.; and Veloso, M. M. 2002. SetA*: An efficient BDD-based heuristic search algorithm. In *AAAI*, 668–673. AAAI Press.

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

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

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

Lawler, E. L. 1963. The quadratic assignment problem. *Management Science* 9(4):586–599.

McMillan, K. L. 1993. *Symbolic Model Checking*. Kluwer Academic Publishers.

Sulewski, D.; Edelkamp, S.; and Kissmann, P. 2011. Exploiting the computational power of the graphics card: Optimal state space planning on the GPU. In *ICAPS*, 242–249. AAAI Press.