ECAI 2012
Luc De Raedt et al. (Eds.)

doi:10.3233/978-1-61499-098-7-306

Symbolic A* Search with Pattern Databases and the Merge-and-Shrink Abstraction

Stefan Edelkamp and **Peter Kissmann**[1] and **Álvaro Torralba**[2]

Abstract. The efficiency of heuristic search planning crucially depends on the quality of the search heuristic, while succinct representations of state sets in decision diagrams can save large amounts of memory in the exploration. BDDA* – a symbolic version of A* search – combines the two approaches into one algorithm. This paper compares two of the leading heuristics for sequential-optimal planning: the merge-and-shrink and the pattern databases heuristic, both of which can be compiled into a vector of BDDs and be used in BDDA*. The impact of optimizing the variable ordering is highlighted and experiments on benchmark domains are reported.

1 INTRODUCTION

Explicit-state heuristic search planners have shown advantages to symbolic planners with binary decision diagrams (BDDs) [5] in cost-optimal planning, suggesting that the increased quality of search heuristics sometimes exceeds the structural savings for representing and exploring large state sets in advanced data structures.

For the automated construction of search heuristics for BDD-based planning, *symbolic pattern databases* (PDBs) have been proposed [10]. They correspond to a complete (or partial) backward exploration of the concrete (or abstracted) state space. These planning heuristics can be exploited in a symbolic version of A* search, BDDA* for short [13].

The *merge-and-shrink* (M&S) heuristic is among the strongest estimates for explicit-state space planning [18]. Newer proposals further improve its quality [22, 20] and outperform other state-of-the-art heuristics like *LM-cut* [16] on a sizable number of domains. Furthermore, it can compute perfect heuristics for some simpler benchmark domains in polynomial time.

In this paper we extract the memory structure of the M&S heuristic in form of an algebraic decision diagram (ADD) [2]. This allows to enrich a symbolic heuristic planner to exploit this expressive estimate. The precomputed ADD is converted to a vector of BDDs and plugged into BDDA* for computing cost-optimal plans. It exactly matches the estimate quality of the explicit-search variants and is general to all existing M&S variants. We will also look at refinements to BDDA* and propose List BDDA*, which exploits a list representation of the search frontier (rather than a matrix representation).

PDBs perform surprisingly well compared to M&S. In our experiments the former yield the perfect heuristic in more instances than the latter. In several cases the construction does not even need to perform abstraction, but resorts to (possibly truncated) backward search in the concrete state space. While the M&S heuristic generalizes PDBs [18], in case of a linear merge strategy, it has the same expressiveness than symbolic search. Therefore, partial symbolic PDBs including all the variables are not dominated by M&S. Furthermore, we show that ADD reduction can yield smaller structures than the one applied in the M&S abstraction. We will also see that the variable ordering in the two heuristics is a crucial parameter to the exploration and produces outcomes of large variety.

The paper is structured as follows. First, we reconsider explicit-state and symbolic heuristic search. Next, we turn to the set of refinements including a new BDDA* version and to the extraction of the M&S heuristic as an alternative to the symbolic PDB heuristic. Limits and possibilities are discussed. In the experimental results we compare the two heuristics and discuss the outcome and effects of changing the variable ordering in both cases.

2 HEURISTIC SEARCH PLANNING

A *planning task* consists of variables of finite domain so that states are assignments to the variables, an initial state, the goal, and a finite set of operators each being a pair of preconditions and effects. In *cost-based planning*, operators are associated with cost values, which often are integers – or alternatively rational numbers, which can be cast to integer values. The task is to find a path (the *plan*) from the initial state to the goal. The plan is *optimal* if its cost is smallest among all possible plans. A heuristic is a mapping from states to natural numbers (or infinity), and admissible if for all possible states the value is not greater than the cost of an optimal plan. We refer to a finite domain variable encoding of the planning problem, abbreviated as *SAS*$^+$ *planning*. A planning task *abstraction* is a planning task based on a mapping for the initial state, goal state as well as the operators. We consider two heuristics based on abstraction.

2.1 Pattern Databases

The pattern database (PDB) heuristic, inspired by a selection of tiles in the sliding-tile puzzle [6], has been extended to the selection of state variables in other domains and in planning [9]. More general definitions have been applied, shifting the focus from the mere selection of a subset of SAS$^+$ variables to different state-space *abstractions* that are computed prior to the search. A PDB stores the shortest path distance from each abstract state to the set of abstract goal states. *Partial PDBs* [1] refer to not conducting the backward search to completion but truncating the search at goal distance d and assigning all remaining states the heuristic value $d + 1$. As a slightly better estimate, we can take the minimum value of $(i + j) > d$ of the goal distance i of a state within the PDB plus the cost j of an operator.

[1] TZI Universität Bremen, Germany, email: {edelkamp, kissmann}@tzi.de
[2] PLG Universidad Carlos III de Madrid, Spain, email: atorralb@inf.uc3m.es

2.2 Merge-and-Shrink

The merge-and-shrink (M&S) heuristic is induced by a *distance-preserving abstraction*, originally proposed in the context of directed model checking [7, 8]. The abstract state space in this heuristic is built incrementally, starting with a set of abstractions associated with each SAS$^+$ variable and merging two abstractions in each step, by computing their cross-product. Most current proposals work on a *linear* arrangement, meaning that one variable is added at a time. The rough idea is that SAS$^+$ variables are greedily chosen to construct a larger state space by computing the (synchronized) product of the existing state space and the one induced by the next SAS$^+$ variable. If the state space becomes too large pairs of states are unified.

Thus, the approach is layered so that each layer correspond with an intermediate abstraction. When a new SAS^+ variable is merged, the synchronized product produces an state in the new layer for each combination of value of the variable and state in the previous layer. Hence, states in each intermediate abstraction are associated to the states in the next layer resulting from their synchronized products.

Most successful shrinking approaches are based on the notion of *bisimulation* [22]. Two states s and s' are bisimilar if they agree on whether or not the goal is true and every planning operator leads to the same abstract state from both s and s' [3]. If only bisimilar states are aggregated, then M&S is guaranteed to be perfect. The bisimulation shrinking strategy computes the coarsest bisimulation, and in the shrinking step it aggregates only bisimilar (abstract) states. In most benchmark domains, however, coarsest bisimulations are still large even under operator projection.

Greedy bisimulation (gop) is a relaxed variant of bisimulation, which demands the bisimulation property only for transitions (s, s'), where the abstract goal distance from s' is at most as large as the abstract goal distance from s. This relaxation forfeits the guarantee of providing a perfect estimate.

Motivated by the size of bisimulations, a more approximate shrinking strategy (gop') builds the coarsest bisimulation and keeps unifying states until the size limit M is reached. The latter may happen before a bisimulation is obtained, in which case it looses information. The strategy attempts to make errors only in more distant states, where the errors will hopefully not be as relevant.

3 SYMBOLIC A* SEARCH

The main limitation for applying PDBs in search practice is the restricted amount of RAM. For the exploration of large state spaces, symbolic search can save huge amounts of memory and computation time. State sets [23] are represented and modified by accessing their characteristic functions.

Decision diagrams [25, 2, 5] are a memory-efficient data structure used to represent Boolean (or integer-valued) functions as well as to perform set-based search, where the diagram represents all binary state vectors that evaluate to certain values. More precisely, a BDD (an ADD) is a directed acyclic graph with one root and two (several) terminal nodes, called sinks. Each internal node corresponds to a binary variable of the state vector and has two successors (low and high), one representing that the variable is false and the other representing that it is true. For any assignment of the variables on a path from the root to a sink the represented function will be evaluated to the value labeling the sink. Moreover, decision diagrams are unique given a fixed variable ordering by applying the two reduction rules of (1) eliminating nodes with the same low and high successors and (2) merging two nodes representing the same variable that share the same low successor as well as the same high successor.

In order to perform symbolic search we need two sets of variables, one (x) representing the current states and another (x') representing the successor states. To find the successors of a set of states S represented in the current state variables given a BDD T for the entire set of operators (i. e., the transition relation) we use the $image$ operator, i. e., $image(S, x) = \exists x.S(x) \wedge T(x, x')[x' \leftrightarrow x]$, where $[x' \leftrightarrow x]$ denotes the swap of the two sets of variables. Similarly, we can perform search in backward direction by using the $pre\text{-}image$ operator, i. e., $pre\text{-}image(S, x') = \exists x'.S(x') \wedge T(x, x')[x \leftrightarrow x']$.

Symbolic PDBs [10] are PDBs that have been constructed symbolically as decision diagrams for later use either in symbolic or explicit heuristic search. Their construction exploits that the transition relation is defined as a relation. The savings observed by the symbolic representation are substantial for many planning domains. Different to the posterior compression [3], the construction in [10] works on compressed representation, allowing much larger databases to be constructed. For such *PDB construction*, backward symbolic search is used. In the case of partial PDBs, the construction is truncated at some fixed point in time. While this works in the concrete state space, PDB construction usually takes place in abstract space, imposed by an abstraction function that often projects some variables to don't cares. The automated selection of variables is important for its success but involved [9, 11, 14, 21].

Algorithmically, we start with the abstract goal set and iterate to successively compute the pre-image. Each state set in a layer is efficiently represented by a corresponding characteristic function. We may assume that the variable ordering is fixed and has been optimized prior to the search. For a given abstraction function the symbolic PDB $Heur(value, x)$ is initialized with the projected goal. As long as there are newly encountered states we take the current backward search frontier and generate the predecessor list with respect to the abstracted transition relation. Then we attach the current BFS level to the new states, merge them with the set of already reached states, and iterate. When operator costs are integers this process can be extended from breadth-first to cost-first levels, and it is possible to combine different symbolic heuristics by taking their maximum or by a controlled combination of their sum. The variables encoded in *value* are often queried at the bottom or at the top (in which case we obtain the equivalent to a vector of BDDs). For BDDA* it is more convenient to choose the one where the heuristic relation is partitioned into $Heur_0(x)$, ..., $Heur_k(x)$, with $Heur(value, x) = \bigvee_{i=0}^{k}(value = i) \wedge Heur_i(x)$.

BDDA* [13], a.k.a. SetA* [19], operates on a BDD priority queue $Open$. In case of discrete cost-values the $Open$ sets can be represented by BDDs. For the organization of the search that avoids BDD arithmetics, it is convenient to partition the state space. As we aim at *cost-optimal symbolic sequential planning*, the matrix-based version of BDDA* works on a partitioning of the search space in g- and h-values, where g is the cost of the path traversed so far and h is the heuristic estimate on the cost to reach the goal. To guarantee optimal cost, BDDA* expands this matrix along the f-diagonals with increasing g-values. The successors of the BDD $Open_{g,h}$ for a chosen transition with cost c are unified with the BDD $Open_{g+c,h'}$, where $h' \in \{0, \ldots, k\}$ is the partitioning obtained by the heuristic evaluation of the successor set.

[3] Label reduction may be applied preserving the heuristic optimality value while exponentially reducing the abstraction size.

3.1 Basic Improvements

Our starting point is the IPC 2011 version of the planner GAMER[4], described in [21]. It applies symbolic PDB construction (15 minutes) and BDDA* search (15 minutes) for cost domains or bidirectional BFS (30 minutes) for unit-cost domains. If backward search takes too long, abstractions are applied, otherwise a (partial) PDB in the concrete search space is constructed. In IPC 2011 GAMER did not score as well as it did in 2008. Of the twelve planners it finished ninth with only 148 solved instances, while one of the FAST DOWNWARD STONE SOUP versions won with a total of 185 solutions. If we compare the number of solved instances of the domains with and without operator costs the results are quite peculiar. For the unit-cost domains GAMER found only 23 solutions; only one participant was worse than that. For the domains with operator costs GAMER found 125 solutions; only three other planners were able to find more (with the maximum being 131). Based on the results of the competition we implemented some small improvements, which are as follows.

In the *solution reconstruction* for bidirectional BFS, GAMER supposed that at least one forward and at least one backward step were performed. The two easiest problems of VISITALL require only a single step, so that the solution reconstruction crashed.

In some cases, *parsing* the ground input took more than 15 minutes, so that actually no search whatsoever was performed in the domains with operator costs: At first, it was parsed in order to generate a PDB, this was killed after 15 minutes, and then the input was parsed again for BDDA*. In the unit-cost domains it sometimes also dominated the overall runtime. Thus, we decided to use a parser generator for Java programs, with which the parsing typically takes at most a few seconds.

In the most complex cases, generating the BDDs for the *transition relation* takes a lot of time, as well. The planner had to generate them twice in case of domains with operator costs if it did not use the abstraction, once for the PDB generation and once for BDDA*. Instead, we store the transition relation BDDs, the BDD for the initial state and that for the goal condition on the hard disk; reading them from disk often is a lot faster than generating them again from scratch.

In two of the new domains, namely PARKING and TIDYBOT, we found that the first backward step takes too long, often even more than 30 minutes, so that the decision whether to use bidirectional or unidirectional BFS could not be finished before the overall time ran out. In these cases, the single images for all the operators were quite fast, but the disjunction took very long. Thus, during the disjunction steps we entered the possibility to check whether too much time, in this case 30 seconds, has passed. If it has we *stop the disjunctions* and the planner only performs unidirectional BFS (or PDB generation using abstractions). This enabled us to find some solutions in TIDYBOT. The problem here is that the goal description allows for too many possibilities, because variables from only very few of the SAS^+ groups are present.

While the original planner used bidirectional BFS for unit-cost domains, we tried running *BDDA* in all cases*, no matter if we are confronted with them or not. Thus, for the unit-cost domains we treated all operators as if they had a cost of 1. We call this implementation *Matrix BDDA**.

3.2 List BDDA*

In Matrix BDDA*, all successors are classified according to their h-value by applying conjunctions with all the heuristic BDDs. When the number of heuristic values grows, this can be inefficient, since some of these conjunctions could be avoided.

[4] available at http://www.plg.inf.uc3m.es/ipc2011-deterministic

Algorithm 1 List-BDDA*.

```
Input:  $\mathcal{O}$: finite set of operators.
        $\mathcal{I}$: initial state.
        $\mathcal{G}$: goal description.
        $c : \mathcal{O} \mapsto \{1, \ldots, C\}$: operator costs.
        $Heur_h$: heuristic (with $h$ being the maximal
            heuristic value).
        $T_a$: transition relation for operator $a \in \mathcal{O}$.
Output: cost-optimal plan.
$Open_0 \leftarrow \mathcal{I}$
for all $f = 0, \ldots$
  for all $g = 0, \ldots, f$
    $h \leftarrow f - g$
    $S \leftarrow Open_g \wedge Heur_h$
    if $(h = 0)$ and $(S \wedge \mathcal{G} \neq 0)$ return ConstructSolution
    for all $i = 1, \ldots, C$
      $Succ_i(x') \leftarrow \bigvee_{a \in \mathcal{A}, c(a)=i} \exists x \,.\, S(x) \wedge T_o(x, x')$
      $Open_{g+i} \leftarrow Open_{g+i} \vee Succ_i[x' \leftrightarrow x]$
```

The representation in the matrix can be simplified to the vector for the states in the $Open$ list ordered along the g-value. The reasoning behind this strategy is to defer the heuristic calculation by computing the conjunction of the successor set with the heuristic estimate only when it is needed for expansion in the currently traversed f-diagonal.

The pseudo-code of the resulting algorithm *List BDDA** for non-zero cost operators is shown in Algorithm 1. All inputs to the algorithm $\mathcal{O}, \mathcal{I}, \mathcal{G}, c, Heur_h, T_a$ are represented as BDDs.

It is simple to add duplicate detection to the algorithm using another set *Closed* for the set of expanded states. The handling of zero-cost operators adds another BFS loop to the code as these operators are to be preferred in the exploration. While Matrix BDDA* uses BFS to get the states reachable with zero-cost operators independent of their actual h values, the list version applies a conjunction with the heuristic value to get only those states in the current f-diagonal.

4 SYMBOLIC MERGE-AND-SHRINK

It is not difficult to observe that the precomputed memory structure of the M&S heuristic can be cast as a symbolic representation of an integer-valued function. This function can be extracted in form of an ADD [2] allowing to enrich a symbolic heuristic planner to exploit this expressive estimate. The precomputed ADD is converted to a vector of BDDs [5] and can be plugged into an optimal symbolic heuristic search planner. Such an approach is general to all M&S variants using a linear merge strategy, including the latest improvements based on bisimulation reductions [22].

Every intermediate abstraction corresponds to a layer in the ADD and each abstract state corresponds to an ADD node. ADD nodes representing an abstract state are connected with the nodes representing the states which result from its synchronized product with the next variable. As M&S works with finite domain variables but the ADD is defined for binary variables, each node with k successors is converted to a binary tree with $\log_2 k$ layers.

To compute the ADD of an M&S heuristic we start by generating the sink nodes associated with the different heuristic values of the abstract states in the last layer. Then, recursively, nodes in the previous layer can be constructed pointing to the nodes in layers already computed. During the ADD construction we ensure the application of the reduction rules, so that the size of the final ADD is usually smaller than the M&S heuristic structure.

4.1 ADD Complexity

The symbolic ADD representation of the M&S heuristic can be computed in time and space $O(nM)$, where n is the size of the Boolean state vector and M is the pre-defined maximum number of states. Moreover, the representation of the heuristic as a sequence of BDDs $h_0, \dots, h_{\max}$ can be computed in time and space $O(h_{\max} nM)$. The time and space complexities are implied by the maximum sizes of the state spaces for the construction of the next-variables tables in the explicit search construction of the M&S heuristic. BDD reduction is a linear time operation [24] and only decreases the size.

The ADD sizes for the two M&S heuristics are shown in Table 1. For each domain we provide the number of instances in which the heuristic computation was finished in 30 minutes as well as the sizes of the largest ADD for each domain. Surprisingly the ADDs are small, especially for the greedy version of M&S, showing that not much memory is spent once the ADD has been computed.

Table 1. Number i of instances with M&S heuristic and maximum number n of ADD nodes over all instances for all domains of the sequential optimal track of IPC 2011.

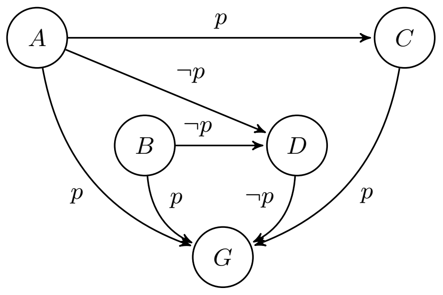
	M&S (gop')		M&S (gop)	
Problem	i	n	i	n
BARMAN	20	177,294	20	45
FLOORTILE	20	1,278,950	8	6,283
NOMYSTERY	20	197,445	20	915
PARKING	0	—	20	2,260
TIDYBOT	0	—	20	15
VISITALL	20	3,225,813	20	7,381
ELEVATORS	20	62,594	0	—
OPENSTACKS	20	102,486	4	134,780
PARC-PRINTER	19	4,606,533	20	11,788
PEG-SOLITAIRE	20	42,170	0	—
SCANALYZER	16	356,698	6	29,921
SOKOBAN	20	1,339	1	33,525
TRANSPORT	20	257,898	20	753
WOODWORKING	20	248,263	20	439,489

4.2 Limits and Possibilities

M&S heuristic strictly generalizes the PDB heuristic as with only merging variables by computing their synchronized product all pattern database heuristics based on projecting the variables can be constructed. Moreover, [17] states that M&S can compute perfect heuristics in polynomial time, where PDBs cannot. The distinguishing example is the GRIPPER domain.

In a symbolic setting, this reasoning, however, is no longer immediate. If all the variables are included in the pattern, the original state space can be fully traversed resulting in the optimal heuristic. As shown in [12], the BDD exploration that computes the perfect heuristic in GRIPPER is polynomial. As the representational power of both alternatives is equivalent (an ADD for M&S and a list of BDDs for the symbolic PDB) both approaches can potentially derive optimal heuristics in the same domains.

However, even if the M&S bisimulation gets the perfect heuristic, it does not always result in a reduced representation of the ADD. First, we observe that for any (e.g., the perfect) heuristic – no matter how it is computed – by the uniqueness property, the according ADDs (following the same variable ordering) have to be the same. Secondly, we can construct an intuitive example, where shrinking is not able to compute the most reduced form of the heuristic.



$$\begin{tikzcd}
A \arrow[rrr, "p"] \arrow[drr, "\neg p"] \arrow[ddr, bend right, "p"'] & & & C \arrow[ddl, bend left, "p"] \\
 & B \arrow[r, "\neg p"] \arrow[d, bend right, "p"] & D \arrow[dl, bend left, "\neg p"] & \\
 & G & &
\end{tikzcd}$$

Figure 1. Example of bisimulation. A, B, C, D and G are states in one level of the M&S process, while p and $\neg p$ are variable assignments that serve as a precondition of the according operators.

Figure 1 shows an example where there are not any bisimilar states. The transition labels have already been reduced so that they refer to variables that have not yet been merged. In the example these labels are preconditions and they only refer to a binary variable p. All the transitions have unit costs and the goal is to reach state G.

We say that two abstract states s and s' are *equivalent*, if and only if, for every value assignment to the variables that have not yet been merged the goal distance remains the same. If two abstract states are equivalent, their corresponding ADD nodes can be unified according to the ADD reduction rule (2). It is easy to see that states A and B in the example are equivalent because in case that p holds both have a cost of 1, while if $\neg p$ holds both have a cost of two. However, they are not bisimilar because B does not have any transition to C. Obviously, states C and D are not bisimilar, given that their transitions have different labels. Hence, no pair of states is reduced by bisimulation.

However, since in the end only the distance to the goal matters, those transitions that are not part of an optimal path should not be taken into account. In the example, if the transition $A \rightarrow C$ is not necessary then A and B are equivalent. Checking if a transition is necessary in any optimal path is not trivial as it needs to consider all the combinations of values of the variables that have not been merged. Some approaches try to consider only a subset of transitions when computing the bisimulation [20] but either do not guarantee optimal heuristics or do not reduce all the equivalent states.

It is possible to extend the example by adding an exponential number of equivalent states that are not bisimilar because they have different transitions that are not needed by any of their optimal paths. Therefore, this may cause an exponential gap between the size of the intermediate abstraction and the final reduced heuristic.

On the other hand, symbolic backward search iteratively constructs the reduced BDDs for every cost. In the absence of 0-cost operators, the intermediate BDDs are always fully reduced. Thus, in some domains the size of the BDDs used by symbolic partial PDBs may be exponentially smaller than the M&S representation. The counterpart is that these BDDs are computed with images of the transition relation, which in some domains may be expensive.

5 EXPERIMENTS

We use two planners as the basis for our experiments, namely FAST DOWNWARD[5] (FD) [15] offering the M&S heuristic for explicit-state planning, and GAMER for executing symbolic heuristic search. Both systems include their latest improvements.

The software infrastructure is taken from the resources of the International Planning Competition 2011. Also, we used all the problems of that competition, i.e., 14 domains, four of those with being unit-cost domains, with 20 problems each. We implemented the proposed refinements in GAMER (Matrix BDDA* and List BDDA*) using the CUDD library of Fabio Somenzi (compiled for 64-bit Linux using the GNU gcc compiler, optimization option -O3). For the experiments we used our own machine (Intel Core $i7$ 920 CPU with 2.67 GHz and 24 GB RAM) with the same settings concerning timeout (30 minutes) and maximal memory usage (6 GB) as in the competition. While the time and memory settings are the same as in the competition, other parameters of the computer are different.

We did experiments with two different configurations of the M&S heuristic, one using only greedy bisimulation (gop) and one using DFP-gop (gop') with parameter $M = 200{,}000$. The M&S planner presented in the competition used these two strategies serially, first the version with gop for 800 seconds and afterward the one with DFP-gop for 1,000 seconds. We decided to run both parts independently to see how well we perform against a more traditional, i. e., non-portfolio, planner. The pattern selection for symbolic PDBs is the same as used by GAMER in the competition [21].

We look at two different variable ordering strategies used by the competition planners, the one applied in FD and the one applied in GAMER. The FD ordering looks at strongly connected components and weights of the causal graph to place variables after the ones they depend on [15]. The GAMER ordering also looks at the dependency of variables and is the result of a random local search to improve the ordering according to incrementally computing the optimization function $\sum_{1\leq i,j,\leq n,(u_i,v_j)\in D}(\pi(i)-\pi(j))^2$, where π denotes the applied permutation and D denotes the set of causal dependencies. Thus, highly related variables are pushed to the middle of the ordering. Both orderings are contradictory, since goal variables are usually the most related with others but also the most influenced, so that GAMER attempts to place them in the middle and FD at the end.

The results are shown in Table 2. All the small improvements in Matrix A* compared to GAMER helped mainly in the unit-cost domains. There we are now able to find the two trivial solutions in VISITALL, one solution in PARKING and eight in TIDYBOT– in the competition we completely failed in both domains. In the domains with operator costs the new parser helped us to find three additional solutions in the SCANALYZER domain. Overall, Matrix BDDA* solves 158 problems, which is 12 problems more than the competition version of GAMER run on our machine.

When comparing both implementations of BDDA*, List BDDA* is better than Matrix BDDA* when the M&S heuristic is used or when PDBs are used in FAST DOWNWARD ordering; in case of PDBs and GAMER ordering both find the same number of solutions. On the other hand, explicit A* beats both BDDA* versions when using FAST DOWNWARD ordering and one of the M&S heuristics, while with GAMER ordering it is better with the gop M&S heuristic but worse with the gop' heuristic. With FAST DOWNWARD ordering and the PDB heuristic both BDDA* versions are better than explicit A*, while with GAMER ordering there is no difference in the total number of found solutions.

The symbolic PDB heuristic did not use abstraction in most of the domains. With GAMER ordering abstraction is used in some problems of PARC-PRINTER, PARKING, SOKOBAN and TIDYBOT. With FAST DOWNWARD ordering abstraction is also used in FLOORTILE and OPENSTACKS. In all the other domains the heuristic were computed by symbolic backward search until all the states had been reached or the time limit of 15 minutes had been expired. The perfect heuristic was found for 70 problems when using GAMER ordering and for 60 with FAST DOWNWARD ordering.

The results are highly influenced by the choice of the variable ordering. Overall the FAST DOWNWARD ordering is better for the M&S heuristic, while GAMER's ordering helps the symbolic exploration. Due to this, the integration of symbolic search and the M&S heuristic is difficult because both have to use the same ordering.

The variable ordering matters not only for the kind of planner used but also for the domain it is used on. For example, in FLOORTILE, NOMYSTERY, PARKING, VISITALL, PARC-PRINTER, and SCANALYZER the FAST DOWNWARD ordering is better in most cases for all planners, while in ELEVATORS, OPENSTACKS, SOKOBAN, and WOODWORKING the GAMER ordering is the better choice.

Overall, the best choice is to use any of the three planners with PDBs and GAMER ordering. However, as the M&S heuristic takes less time to be computed, it is more suitable to be run more than once. Using the same configuration as in the competition, FD with the M&S heuristic can solve 171 problems, 2 more than in the competition, probably due to some bugfixes and performance boosts in that planner as well. On our machine, neither of the two versions took more than 600 seconds for any of the problems, so that a combination of both really is reasonable. The memory limit of 6 GB is what prevents them from finding more solutions.

Given a perfect oracle that tells us for each domain which heuristic, which ordering, and which version of BDDA* to use we would be able to find 185 solutions. In order to come up with such an oracle we have to investigate the differences between the planners, heuristics and orderings further and find out why some work better than others in certain domains.

6 CONCLUSION

According to the outcome of the last two international planning competitions, heuristic and symbolic search are two of the leading methods for sequential optimal planning.

In this paper we have seen a head-to-head race of two symbolic high-quality estimates, namely the PDB and the M&S heuristic. Surprisingly, the former won (if we stick to a single planner run). With PDB search, we arrive at 158 solutions in the set of competition problems (30 to 32 in the unit-cost domains and 126 to 128 in those with operator costs, dependent on the actual planner used). If we would use Matrix BDDA* for the unit-cost domains and List BDDA* for the domains with costs we arrive at 160 problems being solved. With this we are still slightly behind the number of 171 problems solved by explicit search with two different versions of the M&S heuristic. Nevertheless, the performance of solving 128 problems with costs is closing the gap to the state-of-the-art. With 131 found solutions only one of the FAST DOWNWARD STONE SOUP planners are better. Moreover, if we were to exclude the PARC-PRINTER domain, which is special due to the extremely high and diverse operator costs, the picture would actually be fortunate for us.

The variable ordering for the M&S heuristic influences both the quality of the estimate and the symbolic exploration. The heuristic

[5] Retrieved on February 22nd 2012 from the FAST DOWNWARD repository at `http://hg.fast-downward.org`.

Table 2. Number of solved problems for all domains of the sequential optimal track of IPC 2011.

Domain	FAST DOWNWARD Ordering									GAMER Ordering								
	Explicit A*			Matrix BDDA*			List BDDA*			Explicit A*			Matrix BDDA*			List BDDA*		
	gop	gop’	PDB	gop	gop’	PDB	gop	gop’	PDB	gop	gop’	PDB	gop	gop’	PDB	gop	gop’	PDB
NOMYSTERY	13	**20**	14	14	**20**	16	15	**20**	16	13	12	12	14	17	14	13	18	13
PARKING	**7**	0	0	3	0	0	3	0	0	0	0	0	0	0	1	0	0	1
TIDYBOT	**13**	0	12	6	0	6	6	0	6	**13**	0	8	9	0	6	6	0	5
VISITALL	13	11	10	5	5	12	12	12	12	11	9	11	11	10	11	11	10	11
Total (unit cost)	**46**	31	36	28	25	34	36	32	34	37	21	31	34	27	32	30	28	30
BARMAN	4	4	4	4	4	4	4	4	4	4	4	4	**6**	5	4	4	4	4
ELEVATORS	0	11	18	0	16	**19**	0	16	**19**	6	12	**19**	6	14	**19**	5	17	**19**
FLOORTILE	3	7	**12**	3	7	**12**	3	7	**12**	3	4	11	3	4	8	3	4	8
OPENSTACKS	4	16	15	4	15	14	4	15	15	5	16	16	4	**20**	**20**	5	**20**	**20**
PARC-PRINTER	11	**14**	10	8	11	7	10	11	9	11	12	10	8	9	7	10	7	8
PEG-SOLITAIRE	0	**19**	**19**	0	**19**	**19**	0	**19**	**19**	0	**19**	**19**	0	**19**	17	0	**19**	17
SCANALYZER	6	**10**	9	6	9	9	6	9	9	3	8	9	3	8	9	3	7	9
SOKOBAN	1	**20**	4	1	13	12	1	12	12	3	**20**	17	2	18	19	2	18	19
TRANSPORT	6	7	9	6	7	9	6	7	**10**	6	6	9	6	6	7	6	6	8
WOODWORKING	9	6	7	10	7	5	10	7	7	9	9	13	6	8	**16**	13	12	**16**
Total (others)	44	114	107	42	108	110	44	107	116	50	110	127	44	111	126	51	114	**128**
Total (all)	90	145	143	70	133	144	80	139	150	87	131	**158**	78	138	**158**	81	142	**158**

choice applied in FD pleases the M&S heuristic, while the optimization applied in GAMER pleases symbolic exploration. Future work is needed to combine the two for a competitive BDDA* exploration with the M&S heuristic.

The small ADD sizes for the M&S heuristic documented in this paper suggest that there is sufficient memory for computing the maximum of more than one heuristic (in ADD representation). This results in a consistent, strictly more informed heuristic for the (BDD)A* exploration and provides a way of combining the accuracy of PDBs and M&S heuristics.

In many instances that are solved by BDDA* with PDBs no abstraction is applied, meaning that blind symbolic backward search in the concrete state space is either finalized or truncated by the time limit. As a consequence at least in domains where backward search does not explode immediately (due to illegal states produced), bidirectional blind symbolic search is best.

Another competitor to look at would be the planning heuristic h^+. In a restricted setting it has been compiled into a logic program and to a d-DNNF, where d-DNNFs are another succinct representation for Boolean functions [4].

ACKNOWLEDGEMENTS

Thanks to Deutsche Forschungsgesellschaft (DFG) for support in the project ED 74/11-1. Thanks to the Spanish Government for support in the MICINN projects TIN2011-27652-C03-02, TIN2008-06701-C03-03 and to the Comunidad de Madrid for support in the project CCG10-UC3M/TIC-5597.

References

[1] Keith Anderson, Jonathan Schaeffer, and Rob C. Holte, ‘Partial pattern databases’, in *SARA*, pp. 20–34, (2007).

[2] R. Iris Bahar, Erica A. Frohm, Charles M. Gaona, Gary D. Hachtel, Enrico Macii, Abelardo Pardo, and Fabio Somenzi, ‘Algebraic decision diagrams and their applications’, *Formal Methods in System Design*, **10**(2/3), 171–206, (1997).

[3] Marcel Ball and Robert C. Holte, ‘The compression power of symbolic pattern databases’, in *ICAPS*, pp. 2–11, (2008).

[4] Blai Bonet and Hector Geffner, ‘Heuristics for planning with penalties and rewards formulated in logic and computed through circuits’, *Artif. Intell.*, **172**(12–13), 1579–1604, (2008).

[5] Randal E. Bryant, ‘Symbolic manipulation of boolean functions using a graphical representation’, in *DAC*, pp. 688–694, (1985).

[6] Joseph C. Culberson and Jonathan Schaeffer, ‘Pattern databases’, *Computational Intelligence*, **14**(3), 318–334, (1998).

[7] Klaus Dräger, Bernd Finkbeiner, and Andreas Podelski, ‘Directed model checking with distance-preserving abstractions’, in *SPIN*, pp. 19–36, (2006).

[8] Klaus Dräger, Bernd Finkbeiner, and Andreas Podelski, ‘Directed model checking with distance-preserving abstractions’, *STTT*, **11**(1), 27–37, (2009).

[9] Stefan Edelkamp, ‘Planning with pattern databases’, in *ICAPS*, pp. 13–34, (2001).

[10] Stefan Edelkamp, ‘External symbolic heuristic search with pattern databases’, in *ICAPS*, pp. 51–60, (2005).

[11] Stefan Edelkamp, ‘Automated creation of pattern database search heuristics’, in *MOCHART*, (2007).

[12] Stefan Edelkamp and Peter Kissmann, ‘Limits and possibilities of bdds in state space search’, in *AAAI*, pp. 1452–1453, (2008).

[13] Stefan Edelkamp and Frank Reffel, ‘OBDDs in heuristic search’, in *KI*, eds., Otthein Herzog and Andreas Günter, volume 1504 of *Lecture Notes in Computer Science*, pp. 81–92. Springer, (1998).

[14] Patrick Haslum, Adi Botea, Malte Helmert, Blai Bonet, and Sven Koenig, ‘Domain-independent construction of pattern database heuristics for cost-optimal planning’, in *AAAI*, pp. 1007–1012, (2007).

[15] Malte Helmert, ‘The fast downward planning system’, *JAIR*, **26**, 191–246, (2006).

[16] Malte Helmert and Carmel Domshlak, ‘Landmarks, critical paths and abstractions: What’s the difference anyway?’, in *ICAPS*, pp. 162–169, (2009).

[17] Malte Helmert, Patrik Haslum, and Jörg Hoffmann, ‘Flexible abstraction heuristics for optimal sequential planning’, in *ICAPS*, eds., Mark S. Boddy, Maria Fox, and Sylvie Thiébaux, pp. 176–183. AAAI, (2007).

[18] Malte Helmert, Patrik Haslum, and Jörg Hoffmann, ‘Explicit-state abstraction: A new method for generating heuristic functions’, in *AAAI*, pp. 1547–1550, (2008).

[19] Rune M. Jensen, Manuela M. Veloso, and Randal E. Bryant, ‘State-set branching: Leveraging BDDs for heuristic search’, *Artif. Intell.*, **172**(2–3), 103–139, (2008).

[20] Michael Katz, Jörg Hoffmann, and Malte Helmert, ‘How to relax a bisimulation?’, in *ICAPS*, (2012). To Appear.

[21] Peter Kissmann and Stefan Edelkamp, ‘Improving cost-optimal domain-independent symbolic planning’, in *AAAI*, (2011).

[22] Raz Nissim, Jörg Hoffmann, and Malte Helmert, ‘Computing perfect heuristics in polynomial time: On bisimulation and merge-and-shrink abstraction in optimal planning’, in *IJCAI*, pp. 1983–1990, (2011).

[23] Bo Pang and Robert C. Holte, ‘State-set search’, in *SOCS*, (2011).

[24] Detlef Sieling and Ingo Wegener, ‘Reduction of OBDDs in linear time’, *Inf. Process. Lett.*, **48**(3), 139–144, (1993).

[25] Ingo Wegener, *Branching Programs and Decision Diagrams*, SIAM, 2000.