Optimal Symbolic Planning with Action Costs and Preferences

Stefan Edelkamp
TZI, Universität Bremen, Germany
edelkamp@tzi.de

Peter Kissmann
TU Dortmund, Germany
peter.kissmann@tu-dortmund.de

Abstract

This paper studies the solving of finite-domain action planning problems with discrete action costs and soft constraints. For sequential optimal planning, a symbolic perimeter database heuristic is addressed in a bucket implementation of A*. For computing net-benefits, we propose symbolic branch-and-bound search together with some search refinements. The net-benefit we optimize is the total benefit of satisfying the goals, minus the total action cost to achieve them. This results in an objective function to be minimized that is a linear expression over the violation of the preferences added to the action cost total.

1 Introduction

Optimal planning for actions with costs is a natural requirement for many practical applications, while preference constraints influence the plan benefit [Gerevini *et al.*, 2009].

The trade-off between the benefit of the goals and the efforts to achieve them has lead to maximizing the net-benefit, i.e., the total benefit for the satisfaction of the goals, minus the action cost total.

Hence, we consider sequential optimal planning, minimizing total action costs, as well as planning for optimal net-benefits, where additionally the satisfaction of preferences has to be maximized.

We globally assume discrete and finite-domain variables for modeling action costs and inside linear expressions. This implies that while the total benefit is bounded *a priori*, the action cost total is not.

For computing sequential optimal plans, we apply symbolic search planning with binary decision diagrams (BDDs). Firstly, we contribute bidirectional symbolic shortest-path search that constructs and exploits symbolic perimeter databases. Secondly, we include a proposal to deal with preference constraints in a new variant of symbolic branch-and-bound search.

The paper is structured as follows. First, we briefly introduce symbolic planning with BDDs. Next, we consider action cost optimal planning and the construction and integration of symbolic perimeter databases. We recall branch-and-bound search and extend the algorithm to cover action costs. This guides us towards the algorithm for computing the optimal net-benefit. In the experimental part, results in domains of the international planning competition (IPC-2008) are presented together with additional ones based on a sparse matrix representation for large action costs.

2 Symbolic Planning

Symbolic search is based on Boolean satisfiability. The idea is to make use of Boolean functions to reduce the memory blow-up for growing state sets.

There are different options to come up with an encoding of states for a problem. We implemented the approach of Edelkamp and Helmert [1999], which automatically transforms the propositional planning problem into a so-called SAS$^+$ problem on finite domain variables, clustering atoms that are mutually exclusive [Helmert, 2008].

Based on such a fixed-length binary planning state encoding, a characteristic function is used to represent a set of states. As the mapping between the set of states and the characteristic function is a bijection, the latter can be identified with the state set it represents. Actions can also be formalized as relations, characterizing sets of predecessor and successor state pairs. The transition relation *Trans* for the entire problem is the disjunction of individual state transition relations Trans_a, with $a \in \mathcal{A}$ being an action.

What we are interested in is to compute the *image* $\bigvee_{a\in\mathcal{A}}(\exists x.\, \mathit{States}(x) \wedge \mathit{Trans}_a(x,x'))$ of a planning state set represented by the characteristic function *States*. The result is a representation of all states reachable in one step. The inverse operation, called the *preimage* is defined as $\bigvee_{a\in\mathcal{A}}(\exists x'.\, \mathit{States}(x') \wedge \mathit{Trans}_a(x,x'))$.

Note that in symbolic search we have two sets of states, one (x) for the current states and another (x') for the successor states. The image takes a set of states from x and generates its successors in x'. To continue the search, we use the *replace* function $\exists x'.\, \mathit{States}(x') \wedge (x = x')$, which transforms the successor states represented by *States* from x' back to x.

BDDs [Bryant, 1986] are an efficient data structure for representing and manipulating the relations. To evaluate a given input, a path is traced from the root node to one of the sinks, quite similar to the way decision trees are used. What distinguishes BDDs from decision trees is the use of certain reductions, detecting unnecessary variable tests and isomorphisms

Algorithm 1 Symbolic Version of Dijkstra's Algorithm.

```
Input: Symbolic state space planning problem with relations
       $Init(x)$, $Goal(x)$, and $Trans_a(x, x')$
Output: Optimal solution path

$Open[0](x) \leftarrow Init(x)$
for all $f \in \{0, 1, 2, \ldots\}$
  $Min(x) \leftarrow Open[f](x)$
  if $(Min(x) \wedge Goal(x) \neq \bot)$
    return $ConstructPlan(Min(x) \wedge Goal(x))$
  for all $i \in \{1, \ldots, C\}$
    $Succ_i(x') \leftarrow \bigvee_{a \in \mathcal{A}, c(a)=i} \exists x.\ Min(x) \wedge Trans_a(x, x')$
    $Succ_i(x) \leftarrow \exists x'.\ Succ_i(x') \wedge (x = x')$
    $Open[f + i](x) \leftarrow Open[f + i](x) \vee Succ_i(x)$
```

in subgraphs. This leads to a unique and compact representation for many planning domains [Ball and Holte, 2008].

BDDs have been used for propositional action planning [Cimatti *et al.*, 1997]. The first symbolic variant of A* search, called BDDA* [Edelkamp and Reffel, 1998], has been proposed as a variant of Dijkstra's algorithm. Hansen *et al.* [2002] have proposed an alternative implementation with ADDs (arithmetic decision diagrams), while Jensen *et al.* [2008] have introduced branching partitions. A proposal to deal with preference constraints has been proposed by [Edelkamp, 2006] in a refinement to branch-and-bound BDD-based search [Jensen *et al.*, 2006].

Edelkamp [2002] has proposed BDDs to construct and represent pattern databases, which he called symbolic pattern databases.

2.1 Shortest Path Search

Sequential action cost optimal planning covers classical propositional, non-temporal planning with actions having associated non-negative costs (not necessarily uniform).

The symbolic adaptation of Dijkstra's single-source shortest-paths search [1959] in Algorithm 1 relies on discrete action costs in $\{1, \ldots, C\}$. For fractional numbers it is often possible to rescale the problem. A priority queue data structure provides access to states in the search frontier with increasing action cost total. As we deal with discrete costs, we partition the priority queue into buckets *Open*$[0]$, *Open*$[1]$, *Open*$[2]$, etc. [Dial, 1969]. *Open*$[0]$ is initialized to the representation of the initial state. Unless a goal state is reached, in each iteration we first choose the next f-value together with all states in the priority queue having this value. Then for each action $a \in \mathcal{A}$ with cost $c(a) = i$ the transition relation $Trans_a(x, x')$ is applied to determine the BDD representing the subset of all successor states that can be reached with cost i. In order to attach a new f-value to this set, we append the result to bucket $f + i$.

The algorithm finds an optimal solution if one exists. If all previous layers remain in main memory, sequential solution reconstruction is sufficient. To do this, *ConstructPlan* is given the BDD representing the reached goal states. Starting at these, it performs a backward search in the space of the previous layers. It finds an action leading from a state S in a previous layer i to one of the reached goals, then another from a state in another previous layer $j < i$ to S and so on, until the initial state is reached. If layers are eliminated as in breadth-first heuristic search [Zhou and Hansen, 2004], additional relay layers have to be maintained. The relay layers are then used for divide-and-conquer solution reconstruction.

For large values of C, multi-layered bucket and radix-heap data structures are appropriate, as they improve the time for scanning intermediate empty buckets [Ahuja *et al.*, 1990]. For covering zero-cost actions, we invoke another symbolic BFS to compute the closure for each bucket: once a zero-cost action is encountered for a bucket to be expanded, a zero-cost fixpoint is computed. This results in the representation of all states that are reachable by applying one non-zero cost action followed by a sequence of zero-cost actions.

2.2 Perimeter Search

Perimeter search [Dillenburg and Nelson, 1994] tries to reap the benefits of front-to-front evaluations in bidirectional search, while avoiding the computational efforts involved in re-targeting the heuristics towards a continuously changing search frontier. The search direction changes only once. It conducts a cost-bounded best-first search starting from the goal nodes; the nodes on the final search frontier, called the perimeter, are stored in a lookup table. Then a forward search employs a front-to-front evaluation with respect to these nodes.

Although larger perimeters provide better heuristics, they take increasingly longer to compute. As computational requirements for constructing and storing the states inside the perimeter are considerable, a symbolic representation pays off. When a memory or time limit is reached, the cost-layers of the perimeter search (flushed on disk) yield a heuristic partitioning of the search space that is both admissible and consistent. Perimeter heuristics (usually) require full duplicate detection. States outside the perimeter are assigned to the next possible value of the largest goal cost value obtained so far.

For BDD-based construction of a symbolic perimeter database, we execute the symbolic version of Dijkstra's algorithm (Algorithm 1) in reverse direction, starting from the abstract goal, successively computing preimages.

A* using the symbolic perimeter database heuristics *Heur*$[0]$, *Heur*$[1]$, …, *Heur*$[\max_h]$ is depicted in Algorithm 2. Here, Dijkstra's *Open* list becomes a matrix for the g and h values, with g being the distance from the start, h the heuristic estimate on the distance to a goal. The search starts at bucket $[0, h_0]$ (h_0 is the estimated distance from the initial state to the goal) and expands the $f = g + h$ diagonals in increasing order. Once a bucket $[g, h]$ is expanded, its successors with heuristic value h' are inserted into another bucket $[g + i, h']$ for actions with cost i.

To handle larger action cost values, we use a hash map instead of a matrix, as the matrix might become sparse but too large to fit in main memory.

3 Total Benefit

In optimal planning with preferences, see, e.g., in the papers of van den Briel *et al.* [2004] or Smith [2004], we no

Algorithm 2 Symbolic Version of A*.

```
Input: Symbolic state space planning problem with relations
       $Init(x)$, $Goal(x)$, and $Trans_a(x, x')$
Output: Optimal solution path

for all $h \in \{0, \dots, \max_h\}$
  $Open[0, h](x) \leftarrow Init(x) \wedge Heur[h](x)$
for all $f \in \{0, 1, 2, \dots\}$, $g \in \{0, \dots, f\}$
  $Min(x) \leftarrow Open[g, f - g](x)$
  if ($Min(x) \wedge Goal(x) \neq \bot$)
    return $ConstructPlan(Min(x) \wedge Goal(x))$
  for all $i \in \{1, \dots, C\}$
    $Succ_i(x') \leftarrow \bigvee_{a \in \mathcal{A}, c(a)=i} \exists x.\ Min(x) \wedge Trans_a(x, x')$
    $Succ_i(x) \leftarrow \exists x'.\ Succ_i(x') \wedge (x = x')$
    for all $h \in \{0, \dots, \max_h\}$
      $Open[g + i, h](x) \leftarrow Open[g + i, h](x) \vee$
          $Succ_i(x) \wedge Heur[h](x)$
```

longer have a monotonically increasing cost function to be minimized. Hence, we hardly can omit states with an evaluation larger than the current one. Note that compiling of the linear expression in the metric to the actions is not always possible as preference violations are usually state dependent, not action dependent, and not known before the start of the algorithm.

Algorithm 3 Symbolic Version of Breadth-First BnB.

```
Input: Problem with transition relations $Trans_a$, $a \in \mathcal{A}$
       Cost function $m$ to be minimized, preferences $\Phi_p$
Output: Cost-optimal plan

$U \leftarrow \max_m + 1$
$Bound(v) \leftarrow \bigvee_{i=\min_m}^{U-1} (v = i)$
$Closed(x) \leftarrow Open(x) \leftarrow Init(x)$
loop
  if ($Open(x) = \bot$) or ($U = \min_m$)
    return $RetrieveStoredPlan$
  $Intersection(x) \leftarrow Open(x) \wedge Goal(x)$
  $Eval(v, x) \leftarrow Intersection(x) \wedge \bigwedge_p \Phi_p(v, x)$
  $Metric(v, x) \leftarrow Eval(v, x) \wedge Bound(v)$
  if ($Metric(v, x) \neq \bot$)
    $U \leftarrow \min_m$
    while (($Eval(v, x) \wedge (v = U)) = \bot$)
      $U \leftarrow U + 1$
    $ConstructAndStorePlan(Eval(v, x) \wedge (v = U))$
    $Bound(v) \leftarrow \bigvee_{i=\min_m}^{U-1} (v = i)$
  $Succ(x') \leftarrow \bigvee_{a \in \mathcal{A}} \exists x.\ Trans_a(x, x') \wedge Open(x)$
  $Succ(x) \leftarrow \exists x'.\ Succ(x') \wedge (x = x')$
  $Open(x) \leftarrow Succ(x) \wedge \neg Closed(x)$
  $Closed(x) \leftarrow Closed(x) \vee Succ(x)$
```

3.1 Cost-Optimal BFS

Algorithm 3 displays the pseudo-code for a breadth-first exploration, incrementally improving an upper bound U on the solution length. The set-based branch-and-bound search strategy (see the paper of Jensen *et al.* [2006] for comparison) hides the analysis of a layer, in which more than one goal is contained (*ConstructAndStorePlan*). In this function one goal state with minimum cost has to be filtered for solution reconstruction from the set of goal states in the layer.

Based on the range of the variables in the domain metric we search the interval between the minimum and maximum value ($\min_m$ and $\max_m$) from below. For the symbolic representation of metric m, the minimal and maximal values of m define the range that has to be encoded binary. The work of Bartzis and Bultan [2006] implies that the BDD for representing a linear function can be constructed space- and time-efficiently.

For preference constraints of type (*preference* p ϕ_p), we associate a variable v_p (denoting the violation of p), such that $\min(v_p) = 0$ and $\max(v_p) = 1$. Hence, for $\alpha_p \geq 0$ we have $\min_m = \sum_p \alpha_p \cdot \min(v_p) = 0$ and $\max_m = \sum_p \alpha_p \cdot \max(v_p) = \sum_p \alpha_p$.

Maximization problems can be transformed easily into minimization problems. Also, adding a constant offset does not change the set of optimal plans. If we define $\Phi_p(v, x) := (v_p \Leftrightarrow \neg\phi_p)$, then the formula $\bigwedge_p \Phi_p(v, x)$ can be used to evaluate all preferences together in one operation.

Algorithm 3 incrementally improves an upper bound U on the plan cost. The search frontier *Open* – denoting the current BFS layer – is tested for an intersection with the boolean formula representing the goal. This intersection is not processed if it does not contain any state that improves the current bound. This is achieved by the conjunction of the intersection with an indicator function *Bound* for the current search interval. As there can be many different goal states contained in the remaining set, the planning process then determines a state in the intersection that has minimal cost. This is achieved by intersecting with $v = i$, for $i \in \{\min_m, \dots, U - 1\}$ with U being the old bound.

The algorithm applies full duplicate detection. Only clearly inferior states in *Intersection* are neglected. Eventually, the *Open* list runs empty or the upper bound equals the lower one ($U = \min_m$) and the best plan stored is an optimal one.

3.2 Cost-Optimal Search with Action Costs

Following the approach of Dijkstra's algorithm, integrating action costs (without being optimized in the objective function) is rather straight-forward. Instead of breadth-first levels, we now generate levels for each possible cost value. This is implemented in Algorithm 4.

The algorithm mimics Algorithm 3 with the exception that its buckets are organized along total action cost instead of total number of plan steps. Without pruning wrt. the upper bound U, the algorithm would traverse the entire planning state space, expanding each possible planning state exactly once. Only inferior states are omitted when stopping the increase of U or terminating in case $U = \min_m$. Otherwise, the *Open* list runs empty, which is detected if C adjacent buckets are empty (no action cost exceeds C, such that the successors are within C adjacent buckets). Subsequently, the best plan stored is the optimal one.

Algorithm 4 Cost-First BnB Planning Algorithm.

```
Input: Problem with $Trans_a$, and action costs $c(a)$, $a \in \mathcal{A}$
       Cost function $m$ to be minimized, preferences $\Phi_p$
Output: Cost-optimal plan

$U \leftarrow \max_m + 1$
$Bound(v) \leftarrow \bigvee_{i=\min_m}^{U-1} (v = i)$
$Open[\min_m](x) \leftarrow Init(x)$
for all $f \in \{\min_m, \min_m + 1, \min_m + 2, \ldots\}$
  $Open[f](x) \leftarrow Open[f](x) \wedge \neg \bigvee_{i=0}^{f-1} Open[i](x)$
  if $(\bigvee_{i=f-C}^{f} Open[i] = \perp)$ or $(U = \min_m)$
    return RetrieveStoredPlan
  $Intersection(x) \leftarrow Open(x) \wedge Goal(x)$
  $Eval(v, x) \leftarrow Intersection(x) \wedge \bigwedge_p \Phi_p(v, x)$
  $Metric(v, x) \leftarrow Eval(v, x) \wedge Bound(v)$
  if $(Metric(v, x) \neq \perp)$
    $U \leftarrow \min_m$
    while $((Eval(v, x) \wedge (v = U)) = \perp)$
      $U \leftarrow U + 1$
    $ConstructAndStorePlan(Eval(v, x) \wedge (v = U))$
    $Bound(v) \leftarrow \bigvee_{i=\min_m}^{U-1} (v = i)$
  for all $i \in \{1, \ldots, C\}$
    $Succ_i(x') \leftarrow \bigvee_{a \in \mathcal{A}, c(a)=i} \exists x.\ Open[f](x) \wedge Trans_a(x, x')$
    $Succ_i(x) \leftarrow \exists x'.\ Succ_i(x') \wedge (x = x')$
    $Open[f + i](x) \leftarrow Open[f + i](x) \vee Succ_i(x)$
```

This cost-first version of the branch-and-bound planning procedure does not contribute much to the existing portfolio.

It mainly serves as an intermediate step towards net-benefit optimization, our next topic. The metric $m(\pi)$ is independent of the action cost. Only inferior states are pruned. The two computed plans share the same optimal cost value, but are not necessarily the same.

Figure 1 illustrates that the search space is stretched along the benefit (the buckets that are generated in the search process are filled with states, goal states are highlighted with a second circle).

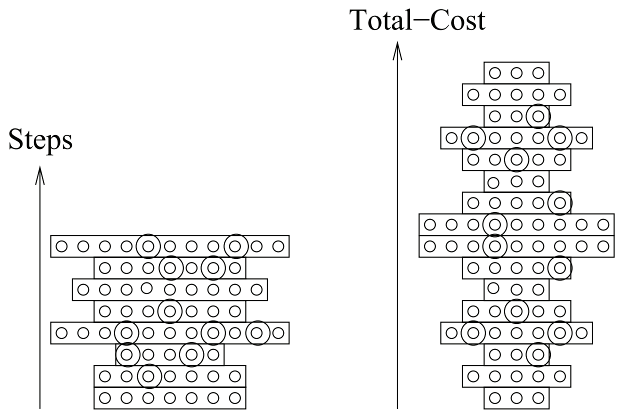

Figure 1: Breadth-First and Cost-First BnB Layers.

Algorithm 5 Net-Benefit Planning Algorithm.

```
Input: Problem with $Trans_a$, and action costs $c(a)$, $a \in \mathcal{A}$
       Cost function $m$ to be minimized, preferences $\Phi_p$
Output: Cost-optimal plan

$U \leftarrow \max_m + 1$
$V \leftarrow \infty$
$Bound(v) \leftarrow \bigvee_{i=\min_m}^{U-1} v = i$
$Open[\min_m](x) \leftarrow Init(x)$
for all $f \in \{\min_m, \min_m + 1, \min_m + 2, \ldots\}$
  $Open[f](x) \leftarrow Open[f](x) \wedge \neg \bigvee_{i=0}^{f-1} Open[i](x)$
  if $(\bigvee_{i=f-C}^{f} Open[i] = \perp)$ or $(U = \min_m)$
    return RetrieveStoredPlan
  $Intersection(x) \leftarrow Open(x) \wedge Goal(x)$
  $Eval(v, x) \leftarrow Intersection(x) \wedge \bigwedge_p \Phi_p(v, x)$
  $Metric(v, x) \leftarrow Eval(v, x) \wedge Bound(v)$
  if $(Metric(v, x) \neq \perp)$
    $U' \leftarrow \min_m$
    while $((Eval(v, x) \wedge (v = U')) = \perp)$ and $(U' + f < V)$
      $U' \leftarrow U' + 1$
    if $(U' + f < V)$
      $V \leftarrow U' + f$
      $U \leftarrow V - f$
      $ConstructAndStorePlan(Eval(v, x) \wedge (v = U'))$
      $Bound(v) \leftarrow \bigvee_{i=\min_m}^{U-1} (v = i)$
  for all $i \in \{1, \ldots, C\}$
    $Succ_i(x') \leftarrow \bigvee_{a \in \mathcal{A}, c(a)=i} \exists x.\ Open[f](x) \wedge Trans_a(x, x')$
    $Succ_i(x) \leftarrow \exists x'.\ Succ_i(x') \wedge (x = x')$
    $Open[f + i](x) \leftarrow Open[f + i](x) \vee Succ_i(x)$
```

4 Net-Benefit

Planning for optimized net-benefits trades utility received by achieving soft goals for the total cost of the actions used to achieve them.

For the sake of brevity, we assume no scaling of the total cost value. Let *benefit*$(\pi) = \sum_p \alpha_p v_p$. Then the metric we consider is $m(\pi) =$ *benefit*$(\pi) +$ *total-cost*(π). Integrating *total-cost* into the cost metric using BDD arithmetic is not possible as the value *total-cost* is not bounded from above. Fortunately, with the bucket to be expanded we already have the current f-value for evaluating *total-cost* at hand. This allows us to use a different upper bound in the branch-and-bound algorithm.

The pseudo-code of the algorithm is presented in Algorithm 5. With V we denote the best solution obtained so far according to the evaluation of $m(\pi)$, which improves over time. As the f-value increases monotonically, we can also adapt V to improve over time. The pseudo-code also adapts a small refinement, by observing that the upper bound U for *benefit*(π) is bounded by V, which is effective if the impact *total-cost* is small.

As with the other branch-and-bound algorithm, the net-benefit procedure looks at all goal states that are not dominated. The state space is traversed cost-first with growing f-

value. For finding the best solutions we observe that all states in one bucket have the same f-value. Since *total-cost*$(\pi) = f$, this allows to stop searching for the best goal state once one has been found from below by only looking at *benefit*(π). We ensure that no inferior plan with a value smaller than the currently best is stored. Testing $U' + f < V$ twice is mandatory: first, to avoid progressing too far and, secondly, to decide, which of the two while-conditions is satisfied.

When looking at U, we observe that the f-value monitoring *total-cost*(π) increases monotonically. This allows to reduce the contribution of *benefit*(π) stored in U to $V - f$. Moreover, if $U = \min_m = 0$, all preference constraints are fulfilled and the remaining quantity will only grow. Therefore, we terminate the search.

5 Experiments

Next, we evaluate the algorithms on the problems of the last international planning competition IPC[1]. To enforce domain-independency of the planners, the strict rules of the competition included the submission of the planner sources prior to the presentation of the domains, on which the planners were run. Moreover, to enable progress for the field, source codes have been published after the competition. There was a timeout of 30 minutes and a memory limit of 2 GB for each of the 30 problems of each domain.

Our planner, called GAMER, finished first in both optimizing tracks (sequential and net-benefit).

To evaluate the improvements in GAMER we implemented afterwards (especially the use of a hash map to handle large action costs), we performed the same experiments on our machine[2], which is slightly slower than the one used in the competition. We provide cumulative bar charts for both tracks, with one bar for each of the participating planners.

Figure 2 shows the results in the sequential optimal track[3]. We set the timeout for backward search to construct the perimeter database to 15 minutes, at which the process that writes immediate results to disk is killed, and forward symbolic A* search is invoked.

As the competition version failed completely in the parc-printer domain, which has very large action costs, we conducted results using the sparse A* matrix representation to cope with these. The new implementation (Gamer) was run on each domain and gives a comparable performance to the old one (Gamer (comp)), but solves some instances of this domain for a better overall performance.

Concerning the runtime, our approach often is slower than the competitors', which is surely due to the 15 minute timeout we used for the database construction. In the non-trivial problems, this backward search did not come to an end before the timeout, while the forward search could be finished in a shorter time. Anyway, the 15 minutes seem reasonable, as this way up to half the time is spent in backward direction and the rest in forward direction.

[1]See http://ipc.informatik.uni-freiburg.de

[2]Two AMD Opteron 250 processors with 2.4 GHz.

[3]For the description of the benchmark domains and the competitors as well as detailed runtime results we refer to the above-mentioned web page.

Figure 3 depicts the results in the net-benefit optimal track. As there were no domains with large action costs, we have visualized the outcome of the competition only.

The results show a clear advantage of our symbolic algorithm. The two competitors either exploit disk-space (Mips XXL), or enumerate all possible preference violations and run an ordinary planner on each of the sub-instances generated (hsp*p)[4].

Here, GAMER is faster than the competitors in the more complex cases (sometimes seconds vs. minutes). In the two domains Transport and Woodworking we were slower than hsp*p but still a lot faster than MIPS-XXL in most cases, which might be due to the fact that the latter uses external-memory algorithms and thus tends to longer runtimes.

6 Conclusion and Discussion

We have proposed a state-of-the-art optimal planning approach that is capable of solving possibly oversubscribed planning domains with action costs and preference constraints, competing against each other in a linear objective function. The limitations we impose are finite domain variables for modeling action costs and preference violation in the objective function.

The symbolic algorithms perform variants of uni- and bidirectional breadth-first search with no problem abstraction. This representational advantage of using BDDs outperformed the attempts of coming up with a suitable admissible (problem relaxation) heuristic working well across larger sets of benchmark domains.

One additional conclusion is that for step- and cost-optimal planning BDD-based planning seems to have an advantage to SAT-based technology [Kautz and Selman, 1996], which has dominated the area of optimal planning in the last decade.

An option worth trying are symbolic implementations of partial pattern databases [Anderson *et al.*, 2007].

Acknowledgements

We thank DFG for support in the project ED 74/3. We also acknowledge the hard work from various sides within the deterministic track of the 2008 international planning competition.

References

[Ahuja *et al.*, 1990] R. K. Ahuja, K. Mehlhorn, J. B. Orlin, and R. E. Tarjan. Faster algorithms for the shortest path problem. *Journal of the ACM*, 37(2):213–223, 1990.

[Anderson *et al.*, 2007] K. Anderson, R. Holte, and J. Schaeffer. Partial pattern databases. In *SARA*, Lecture Notes in Computer Science, pages 20–34. Springer, 2007.

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

[4]According to Patrik Haslum, the author of hsp*p, the number of problems with soft goals in Peg-Solitaire is too large, so that hsp*p can parse it but not solve it.

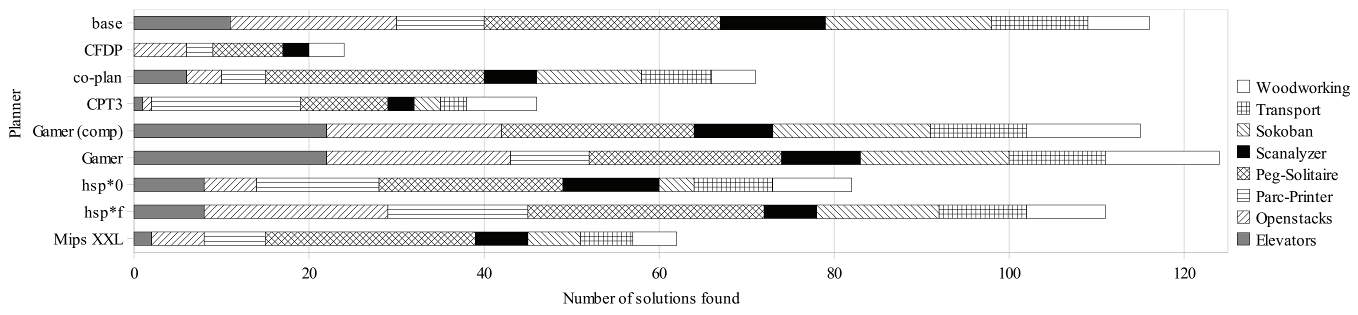

Figure 2: Performance Results on the Competition Domains in the Sequential Optimal Track.

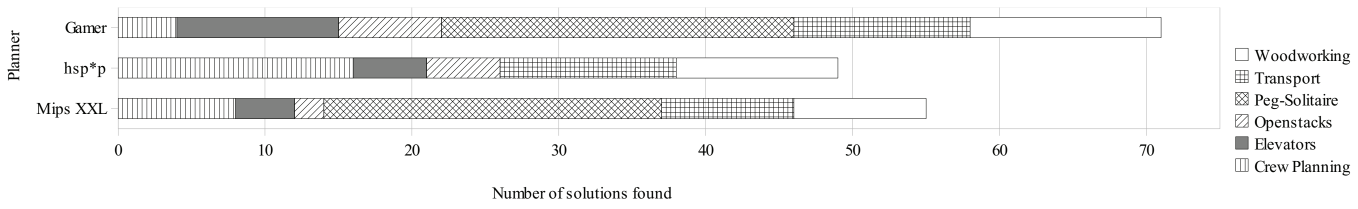

Figure 3: Performance Results on the Competition Domains in the Net-Benefit Optimal Track.

[Bartzis and Bultan, 2006] C. Bartzis and T. Bultan. Efficient BDDs for bounded arithmetic constraints. *STTT*, 8(1):26–36, 2006.

[Bryant, 1986] R. E. Bryant. Graph based algorithms for boolean function manipulation. *IEEE Transaction on Computing*, 35(8):677–691, 1986.

[Cimatti *et al.*, 1997] A. Cimatti, E. Giunchiglia, F. Giunchiglia, and P. Traverso. Planning via model checking: A decision procedure for AR. In *ECP*, Lecture Notes in Computer Science, pages 130–142. Springer, 1997.

[Dial, 1969] R. B. Dial. Shortest-path forest with topological ordering. *Communications of the ACM*, 12(11):632–633, 1969.

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

[Dillenburg and Nelson, 1994] J. F. Dillenburg and P. C. Nelson. Perimeter search. *Artificial Intelligence*, 65(1):165–178, 1994.

[Edelkamp and Helmert, 1999] S. Edelkamp and M. Helmert. Exhibiting knowledge in planning problems to minimize state encoding length. In *ECP*, Lecture Notes in Computer Science, pages 135–147. Springer, 1999.

[Edelkamp and Reffel, 1998] S. Edelkamp and F. Reffel. OBDDs in heuristic search. In *KI*, pages 81–92, 1998.

[Edelkamp, 2002] S. Edelkamp. Symbolic pattern databases in heuristic search planning. In *AIPS*, pages 274–293, 2002.

[Edelkamp, 2006] S. Edelkamp. Cost-optimal symbolic planning with state trajectory and preference constraints. In *ECAI*, pages 841–842, 2006.

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

[Hansen *et al.*, 2002] E. A. Hansen, R. Zhou, and Z. Feng. Symbolic heuristic search using decision diagrams. In *SARA*, Lecture Notes in Computer Science, pages 83–98. Springer, 2002.

[Helmert, 2008] M. Helmert. *Understanding Planning Tasks: Domain Complexity and Heuristic Decomposition*, volume 4929 of *Lecture Notes in Computer Science*. Springer, 2008.

[Jensen *et al.*, 2006] R. Jensen, E. Hansen, S. Richards, and R. Zhou. Memory-efficient symbolic heuristic search. In *ICAPS*, pages 304–313, 2006.

[Jensen *et al.*, 2008] R. M. Jensen, M. M. Veloso, and R. E. Bryant. State-set branching: Leveraging BDDs for heuristic search. *Artificial Intelligence*, 172(2-3):103–139, 2008.

[Kautz and Selman, 1996] H. Kautz and B. Selman. Pushing the envelope: Planning propositional logic, and stochastic search. In *AAAI*, pages 1194–1201, 1996.

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

[van den Briel *et al.*, 2004] M. van den Briel, R. Sanches, M. B. Do, and S. Kamphampati. Effective approaches for partial satisfaction (over-subscription) planning. In *AAAI*, pages 562–569, 2004.

[Zhou and Hansen, 2004] R. Zhou and E. A. Hansen. Breadth-first heuristic search. In *ICAPS*, pages 92–100, 2004.