Symbolic Shortest Paths Planning

Stefan Edelkamp[*]
Computer Science Department
University of Dortmund, Germany
stefan.edelkamp@cs.uni-dortmund.de

Abstract

This paper studies the impact of pattern databases for solving shortest path planning problems with limited memory. It contributes a bucket implementation of Dijkstra's algorithm for the construction of shortest path planning pattern databases and their use in symbolic A* search. For improved efficiency, the paper analyzes the locality for weighted problem graphs and shows that it matches the duplicate detection scope in best-first search graphs. Cost-optimal plans for compiled competition benchmark domains are computed.

Introduction

Optimal planning for action with costs is a natural requirement for many applications. It is common that costs are positive bounded integers[1]. As an example, take macros actions (Korf 1985) that dramatically reduce search efforts. Unfortunately, planners for macro actions are often not optimal (Botea, Müller, & Schaeffer 2005). Consider a (deterministic) planning problem $\mathcal{P} = (\mathcal{S}, \mathcal{A}, \mathcal{I}, \mathcal{G})$, and the search for a cost-optimal sequence of actions $\pi = (a_1 \ldots, a_k)$, $a_i \in \mathcal{A}$, that leads from the initial state $\mathcal{I} \subseteq \mathcal{S}$ to the planning goal $\mathcal{G} \subseteq \mathcal{S}$. In other words, the objective is to minimize the costs of applying all actions in the plan π. We distinguish the following cost models:

C1 uniform action costs ; e.g., $c(a) = 1$ for $a \in \mathcal{A}$;
C2 function $c(a)$ of action $a \in \mathcal{A}$;
C3 function $c(a, u)$ of action $a \in \mathcal{A}$ and state $u \in \mathcal{S}$; and
C4 arbitrary cost function encoded as part of the problem.

So far, the series of international planning competitions has focused on action counting in cost model **C1** (Bacchus 2001) full metric planning in cost model **C4** (Fox & Long 2003), and preference constraints (Gerevini & Long 2005), which penalize plans that go through certain states in cost models **C3** and **C4**. According to a current proposal, tackling cost model **C2** to compute the sum $\sum_{a_i \in \pi} c(a_i)$ is one central aims for the deterministic part of the sixth international planning competition (IPC-6).

[*]Thanks to DFG for support in project ED 74/4

[1]For fractional values it is also possible and beneficial to achieve this by rescaling

In PDDL shortest-path planning in cost model **C2** can be modeled by specialized variables increased by a constant amount in the effects. Alternatively, we may extend PDDL by introducing a tag (`cost`) for each action, whose contribution is monitored in the plan objective (`total-cost`).

This paper contributes symbolic single-source shortest-path search for additive cost functions in cost model **C2**. In contrast to existing cost-optimal symbolic search algorithms, not all states are visited. Instead we conduct symbolic shortest-path pattern database A* search.

The paper is structured as follows. First we recall symbolic planning. To indicate the relation to the state-of-the-art, we draw some initial experiments. We then discuss the symbolic implementation of Dijkstra's single-source shortest-paths algorithm and its complexity. Next we address symbolic shortest-path planning pattern databases and their construction for cost model **C2**. To restrict the scope for duplicate detection, we extend the concept of *locality* from breadth-first to best-first search graphs. We show how to combine a set of disjoint symbolic pattern databases into one. In the experiments, we provide promising results for cost-optimal variants of existing competition benchmarks.

Symbolic Planning

Symbolic planning often refers to analyzing planning graphs (Blum & Furst 1995) or by checking the satisfiability of formulas (Kautz & Selman 1996; Biere *et al.* 1999). Here, we refer to the symbolic exploration in the context of using BDDs (Bryant 1985), although more general automata concepts may apply (Borowski & Edelkamp 2007). While invented in model checking, BDDs contribute to many successful AI planning systems (Cimatti, Roveri, & Traverso 1998; Jensen 2003; Jensen *et al.* 2006; Edelkamp 2003). The idea is to lessen the costs associated with the exponential memory requirements for the state sets involved as problem sizes get bigger. We assume a binary (or at least a finite domain) encoding of a planning problem.

Action are formalized as relations, representing sets of tuples of predecessor and successor states. This allows to compute the image as a conjunction of the state set (formula) and the transition relation (formula), existentially quantified over the set of predecessor state variables. This way, all states reached by applying one action to one state in the input set are determined. Iterating the process (starting with the representation of the initial state) yields a symbolic implementation of breadth-first search (BFS). Fortunately, by keeping sub-relations Trans_a attached to each action $a \in \mathcal{A}$ it is not required to build a monolithic transition relation. The image of state set S then reads as $\bigvee_{a\in\mathcal{A}} \left(\exists x \left(\mathit{Trans}_a(x, x') \wedge S(x)\right)\right)$.

Step-Optimal Symbolic Planning

For step-optimal symbolic planning, uni- and bidirectional symbolic BFS (Edelkamp & Helmert 2001), as well as different symbolic implementations of A* (Edelkamp & Reffel 1998; Hansen, Zhou, & Feng 2002; Jensen, Bryant, & Veloso 2002; Qian 2006) and branch-and-bound (Jensen *et al.* 2006) have been proposed.

Our planner MIPS-BDD first generates a grounded planning instance in PDDL (ground) together with a minimized state encoding (Edelkamp & Helmert 1999; Helmert 2004), and then starts a symbolic exploration (bdd-solver).

By the dominance of parallel-optimal especially SAT-based planners, the state-of-the-art in solving planning problems step-optimal is less known. To illustrate that BDDs are competive, we compare symbolic BFS in MIPS-BDD with other step-optimal planners. That blind BFS often performs well wrt. heuristic search is an observation also encountered in explicit search (Helmert & Röger 2007).

First we select competition results of top-performing (parallel-)optimal planners SATPLAN (Kautz, Selman, & Hoffmann 2006) and MAXPLAN (Xing, Chen, & Zhang 2006) at IPC-5. The results for *Openstack*, (obtained with matching resource limits) are shown in Figure 1. The solution length match, as for this domain all possible plans are sequential. We see a clear advantage of applying BDDs.

Problem	OPT	MIPS-BDD	SAT-PLAN	MAX-PLAN	CPT2
01	23	1.02	>1,800	979	>1,800
02	23	0.98	>1,800	1,353	>1,800
03	23	1.00	>1,800	1,148	>1,800
04	23	0.99	>1,800	841	>1,800
05	23	1.00	>1,800	1,438	>1,800
06	45	3.33	>1,800	>1,800	>1,800
07	46	3.44	>1,800	>1,800	>1,800
08	87	1,132	>1,800	>1,800	>1,800
09	87	544	>1,800	>1,800	>1,800

Figure 1: Step-optimal search in *Openstack* (IPC-5).

A recent paper by Helmert, Haslum, & Hoffmann (2007) pushes the envelope for step-optimal planning. Their planner LFPA outperformed other step-optimal heuristic search planners like HSP (max-pair), BFHS (Zhou & Hansen 2004), PDB (Haslum *et al.* 2007), and FDP (Grandcolas & Pain-Barre 2007)[2] (on the selected benchmark problems). It was also compared with MIPS-BDD IPC-5 competition results. In one domain (*TPP*), MIPS-BDD was the clear-cut winner, while in the other domain (*Pipesworld*) its performance degrades. This is partly due to the fact that for this particular domain MIPS-BDD's backward BFS iterations produce extra-ordinary large BDDs[3].

In new runs[4] we could validate **all** published solutions length of LFPA to be optimal, while showing how well blind symbolic BFS actually scales. The results are shown in Table 2 (*PSR*), Table 3 (*Pipesworld/Tankage*), Table 4 (*Logistics*), and Table 5 (*Satellite*). Note that heuristic search abstractions as applied in LFPA require some additional input parameters, while symbolic BFS in MIPS-BDD does not.

Problem	OPT	MIPS-BDD	LFPA	PDB	BFS	HSP	BFHSP
29	21	2.35	3.47	255	2.30	3.66	1.43
36	22	6.85	67.94	1026	16.82	27.49	25.15
40	20	2.26	38.29	1309	11.91	15.11	7.94
48	37	128	36.16	787	457	>1,800	>1,800
49	47	3,255	67.75	>1,800	>1,800	>1,800	>1,800

Figure 2: Step-optimal search in *PSR* (IPC-4).

Problem	OPT	MIPS-BDD	LFPA	PDB	BFS	HSP	BFHSP	FDP
01	5	1.12	1.00	10.59	0.00	0.00	0.15	0.05
02	12	1.30	1.91	20.48	0.01	0.04	0.20	0.41
03	8	4.43	3.99	99.27	1.45	1.60	5.21	2.17
04	11	5.98	9.98	244	5.66	17.56	20.44	27.75
05	8	9.13	8.68	162	0.34	1.97	5.04	1.27
06	10	13.22	17.43	409	1.63	8.17	16.35	8.57
07	8	91.33	25.84	>1,800	379	1,295	>1,800	12.41
08	11	601	43.70	>1,800	>1,800	>1,800	>1,800	1,092
11	22	757	26.22	795	55.91	436	>1,800	>1,800
13	16	2,261	70.14	>1,800	>1,800	>1,800	>1,800	>1,800
15	30	3,247	161	>1,800	>1,800	>1,800	>1,800	>1,800
21	14	854	52.79	>1,800	121	736	>1,800	375
31	39	10,730	50.36	>1,800	24.53	673	>1,800	>1,800

Figure 3: Step-optimal search in *Pipesworld* (IPC-4).

Cost-Optimal Symbolic Planning

In the context of introducing preferences (Gerevini & Long 2005), cost-optimal symbolic search (wrt. the problem's metric) applies (Edelkamp 2006). The approach generates the entire search space via BFS, incrementally improving the solution quality with increasing depth.

In Figure 6 we show that MIPS-BDD computes optimal solutions in planning domains with preferences (and state trajectory constraints), producing significantly better plans than sub-optimal solvers. As expected, the price for optimality is a drastic increase in the search time. The last result shows the time when the optimal solution was generated, while optimality was proven after 4,607s.

[2]Unfortunately, another recent step-optimal planner *Petrify* (Hickmott *et al.* 2006) was not evaluated.

[3]Implementation refinements wrt MIPS-BDD IPC-5 include improvements to the ordering of the minimized state encoding (e.g. by imposing conflict-dependent sorting ordering), an transition relation that is kept partitioned wrt. action and cost, improved garbage-collection and a 64-bit version for handling larger BDDs.

[4]Experiments were conducted on a 64-bit computer with a 2.4 GHz CPU (with 8 GB RAM). The other planners (in this section) are evaluated on 3GHz (with 1.5 GB RAM).

Problem	OPT	MIPS-BDD	LFPA	PDB	BFS	HSP	BFHSP
4-0	20	1.15	0.10	3.21	0.09	0.06	0.11
4-1	19	1.15	0.09	5.48	0.08	0.04	0.16
5-0	27	1.23	0.87	16.75	1.12	1.03	2.41
5-1	17	1.13	0.88	4.58	0.22	0.09	0.18
6-0	25	1.20	3.65	16.48	5.96	3.34	3.62
6-1	14	1.09	3.85	3.16	0.28	0.06	0.11
7-0	36	10.70	24.56	91.12	>1,800	>1,800	>1,800
7-1	44	21.82	26.84	156	>1,800	>1,800	>1,800
8-0	31	5.94	37.09	79.49	>1,800	>1,800	>1,800
8-1	44	13.69	40.77	160	>1,800	>1,800	>1,800
9-0	36	8.42	55.47	127	>1,800	>1,800	>1,800
9-1	30	4.52	53.42	92.85	>1,800	>1,800	>1,800
10-0	45	694	117	497	>1,800	>1,800	>1,800
10-1	42	577	129	405	>1,800	>1,800	>1,800
11-0	48	546	129	377	>1,800	>1,800	>1,800
11-1	60	2,320	284	>1,800	>1,800	>1,800	>1,800
12-0	42	473	185	545	>1,800	>1,800	>1,800
12-1	68	15,112	221	>1,800	>1,800	>1,800	>1,800

Figure 4: Step-optimal search in *Logistics* (IPC-2).

Problem	OPT	MIPS-BDD	LFPA	PDB	BFS	HSP	BFHSP
01	9	1.04	0.01	0.27	0.00	0.00	0.03
02	13	1.35	0.08	0.32	0.01	0.01	0.08
03	11	1.99	3.16	2.10	0.30	0.18	0.16
04	17	2.52	6.92	11.54	9.34	6.26	3.08
05	15	37.78	47.74	110	>1,800	>1,800	119
06	20	27.40	21.19	634	>1,800	707	265

Figure 5: Step-optimal search in *Satellite* (IPC-3).

Symbolic Shortest Paths Planning

For positive action costs, the first plan reported by Dijkstra's 1959 algorithm is already optimal. For implicit graphs, we need two data structures, one to access nodes in the search frontier and one to detect duplicates. In model **C2**

Algorithm 1 Symbolic-Shortest-Path.

```
Input: State space planning problem $\mathcal{P} = (\mathcal{S}, \mathcal{A}, \mathcal{I}, \mathcal{G})$ in symbolic form
       with $\mathcal{I}(x)$, $\mathcal{G}(x)$, and $Trans_a(x, x')$
Output: Optimal solution path

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

the priority queue can be partitioned into a list of buckets $Open[0], \ldots, Open[f_{\max}]$. We assume that the largest action cost (inducing the difference between the largest and smallest key) is bounded by some constant C. An according symbolic search procedure is implemented in Algorithm 1. The algorithm works as follows. The BDD $Open[0]$ is initialized to the representation of the start state. Unless one goal state is reached, in one iteration we first choose the next f-value

Problem	MIPS-BDD		SG-PLAN		MIPS-XXL		HPlan-P	
1	0	0.01	8	0.00	0	0.09	0	0.17
2	1	0.02	13	0.00	1	3.08	1	3.73
3	2	0.31	26	0.01	10	299	17	160
4	5	1,026	39	0.02	44	6,043	36	287

Figure 6: Cost-optimal symbolic search in *Storage*, *Qualitative Preferences* (IPC-5).

together with the BDD *Min* of all states in the priority queue having this value. Then for each $a \in A$ with $c(a) = i$ the transition relation $Trans_a(x, x')$ is applied to determine the BDD for the subset of all successor states that can be reached with cost i. In order to attach new f-values to this set, we insert the result into bucket $f + i$.

A slightly advanced implementation for the priority queue is a one-level bucket (Dial 1969). This priority queue implementation consists of an array of size $C + 1$, each of which is the link to a BDD for the elements.

If all previous layers remain in main memory, sequential solution reconstruction is sufficient. If layers are eliminated as in frontier search (Korf *et al.* 2005) or breadth-first heuristic search (Zhou & Hansen 2004), additional relay layers have to be maintained. The state closest to the start state in the relay layer is used for divide-and-conquer solution reconstruction. Alternatively, already expanded buckets are flushed to the disk (Edelkamp 2005). 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).

Theorem 1 *(Optimality and Complexity of Algorithm 1) For transition weights $w \in \{1, \ldots, C\}$, the symbolic version of Dijkstra's algorithm in a one-level bucket priority queue finds the optimal solution with at most $O(C \cdot f^*)$ full and $O(C \cdot |\mathcal{A}| \cdot f^*)$ partitioned images, where f^* is the optimal solution cost.*

Proof: Since f is monotonically increasing, the first goal expanded with cost f^* delivers a cost-optimal plan. Given that the action costs are positive, we compute at most $O(C \cdot f^*)$ full and $O(C \cdot |\mathcal{A}| \cdot f^*)$ partitioned images.

The above algorithm traverses the search tree expansion of the problem graph. It is sound as it finds an optimal solution if it exists. In the above implementation, however, it is not complete, as it does not necessarily terminate if there is no solution. We consider termination in form of delayed duplicate detection in the next two sections.

Symbolic Pattern Databases

Abstraction is the key to the automated design of search heuristics. Applying abstractions simplifies a problem, and exact distances in these relaxed problems can serve as lower bound estimates for the concrete state space (provided that each concrete path maps to an abstract path). Moreover, the combination of heuristics based on different abstractions often leads to a better search guidance. Pattern databases (Culberson & Schaeffer 1998) completely evaluate the abstract search space $\mathcal{P}' = (\mathcal{S}', \mathcal{A}', \mathcal{I}', \mathcal{G}')$ prior to the concrete, base-level search in $\mathcal{P}$. More formally, a pattern database is a lookup table indexed by $u' \in \mathcal{S}'$ containing the shortest path cost from u' to the abstract goal $\mathcal{G}'$. The size of a pattern database is the number of states in $\mathcal{P}'$.

Symbolic pattern databases (Edelkamp 2002) are pattern databases that have been constructed symbolically for later use either in symbolic or explicit heuristic search. They are based on the advantage of the fact that *Trans* has been defined as a relation. In backward search we successively compute the preimage according to the formula $\exists x \ \bigvee_{a \in \mathcal{A}}(S(x) \wedge \mathit{Trans}_a(x', x))$. Each state set in a shortest path layer is efficiently represented by a corresponding BDD. Different to the posterior compression of the state set, the construction itself works on a compressed representation, allowing the generation of much larger databases.

In its original form, symbolic pattern databases are relations of tuples (f, x), which evaluate to *true* if the heuristic estimate of a states encoded in x matches the heuristic value encoded in f. Such relation can represented as a BDD for the entire problem space. Equivalently, a symbolic pattern database can be maintained by set of BDDs $\mathit{PDB}[0], \ldots, \mathit{PDB}[h_{\max}]$. For such construction of a symbolic shortest-path pattern database, Algorithm 1 is adapted as follows. The list is initialized with the abstracted goal (setting $\mathit{PDB}[0]$ to $\mathcal{G}'$) and, as long as there are newly encountered states, we take the current frontier and generate the set of predecessors with respect to the abstract transition relation. Then we attach the matching bucket index to the new state set, and iterate the process.

Different to Algorithm 1, the exploration has to terminate, once the abstract search space has been fully explored. Therefore, duplicates have to be detected and eliminated. If the entire list of BDDs is available we simply subtract the $\mathit{PDB}[0] \vee \ldots \vee \mathit{PDB}[i-1]$ from the current layer $\mathit{PDB}[i]$. If memory is sparse, a strategy to reduce the duplicate detection scope becomes crucial.

Shortest-Path Locality

How many layers are sufficient for full duplicate detection in general is dependent on a property of the search graph called locality (Zhou & Hansen 2004). In the following we generalize the concept from unweighted to weighted search graphs. Let $\mathit{succ}(u) = \{v \in \mathcal{S} \mid \exists a \in \mathcal{A} : a(u) = v\}$.

Definition 1 *(Shortest Path Locality) For a problem graph G with cost function c and δ being defined as the minimal cost between two states, the* shortest path locality *is*

$$\mathrm{L} = \max_{u \in \mathcal{S}, v \in \mathrm{succ}(u)} \{\delta(\mathcal{I}, u) - \delta(\mathcal{I}, v) + c(u, v)\}.$$

In unweighted, undirected graphs $\delta(\mathcal{I}, u)$ and $\delta(\mathcal{I}, v)$ differ by at most one, so that the locality is two[5]

We will see that the locality determines the *thickness* of the search frontier needed to prevent duplicates in the search. In contrast to explicit-state search, in symbolic planning there are no duplicates within one bucket, since the BDD representation is unique.

While the locality is dependent on the graph the duplicate detection scope also depends on the search algorithm applied. For BFS, the search tree is generated with increasing path lengths (number of edges), while for weighted graphs the search tree is generated with increasing path cost (this corresponds to Dijkstra's exploration strategy in the one-level bucket priority queue data structure). The following result extends a finding for breadth-first to best-first graphs.

Theorem 2 *(Shortest-Path Locality determines Boundary for Best-First Search Graphs) In a positively weighted search graph the number of shortest-path buckets that need to be retained to prevent duplicate search effort is equal to the shortest path locality of the search graph.*

Proof: Let us consider two nodes u and v, with $v \in \mathit{succ}(u)$. Assume that u has been expanded for the first time, generating the successor v which has already appeared in the layers $0, \ldots, \delta(\mathcal{I}, u) - L$ implying $\delta(\mathcal{I}, v) \leq \delta(\mathcal{I}, u) - L$. We have

$$\begin{aligned} L &\geq \delta(\mathcal{I}, u) - \delta(\mathcal{I}, v) + c(u, v) \\ &\geq \delta(\mathcal{I}, u) - (\delta(\mathcal{I}, u) - L) + c(u, v) = L + c(u, v) \end{aligned}$$

This is a contradiction to $c(u, v) > 0$.

The condition $\delta(\mathcal{I}, u) - \delta(\mathcal{I}, v) + c(u, v)$ maximized over all nodes u and $v \in \mathit{succ}(u)$ is not a property that can be easily checked before the search. To determine the number of shortest-path layers prior to the search, it is important to establish sufficient criteria for the locality of a search graph. The question is, if we can establish a sufficient condition for an upper bound.

Theorem 3 *(Upper Bound on Shortest Path Locality) In a positively weighted search graph the shortest path locality can be bounded by the minimal distance to get back from a successor node v to u, maximized over all u, plus C.*

Proof: For any states $\mathcal{I}, u, v$ in a graph, the triangular property of shortest path $\delta(\mathcal{I}, u) \leq \delta(\mathcal{I}, v) + \delta(v, u)$ is satisfied, in particular for $v \in \mathit{succ}(u)$. Therefore $\delta(v, u) \geq \delta(\mathcal{I}, u) - \delta(\mathcal{I}, v)$ and $\max\{\delta(v, u) \mid u \in \mathcal{S}, v \in \mathit{succ}(u)\} \geq \max\{\delta(\mathcal{I}, u) - \delta(\mathcal{I}, v) \mid u \in \mathcal{S}, v \in \mathit{succ}(u)\}$. In positively weighted graphs, we additionally have $\delta(v, u) \geq 0$ such that $\max\{\delta(v, u) | \ u \in \mathcal{S}, v \in \mathit{succ}(u)\} + C$ is larger than the shortest path locality.

Theorem 4 *(Upper Bounds on Shortest-Path Locality in Undirected Graphs) For undirected weighted graphs with maximum edge weight C we have* $\mathrm{L} \leq 2C$.

Proof: For undirected graphs with with maximum edge cost C we have $L \leq \max\{\delta(v, u) \mid u \in \mathcal{S}, v \in \mathit{succ}(u)\} + C = \max\{\delta(u, v) \mid u \in \mathcal{S}, v \in \mathit{succ}(u)\} + C = \max\{c(u, v) \mid u \in \mathcal{S}, v \in \mathit{succ}(u)\} + C = 2C$.

Automated Pattern Selection

In domain-dependent planning, the abstraction functions are selected by the user. For domain-independent planning the system has to infer the abstractions automatically. Unfortunately, there is a huge number of feasible abstractions to choose from (Hoffmann, Sabharwal, & Domshlak 2006).

[5]Due to a more general setting, our definition for unweighted graphs is off by 1 compared to the definition of (Zhou & Hansen 2004), where locality does not include the edge cost $\max\{\max_{u \in \mathcal{S}, v \in \mathit{succ}(u)}\{\delta(\mathcal{I}, u) - \delta(\mathcal{I}, v)\}, 0\})$.

Nonetheless, first progress in computing abstractions automatically has been made (Haslum, Bonet, & Geffner 2005). One natural option applicable to propositional domains is to select a pattern set R and apply $u \cap R$ to each planning state $u \in \mathcal{S}$. The interpretation is that all variables not in R are mapped to *don't care*. More formally, the abstraction $\mathcal{P}' = (\mathcal{S}', \mathcal{A}', \mathcal{I}', \mathcal{G}')$ of a (propositional) planning problem $\mathcal{P} = (\mathcal{S}, \mathcal{A}, \mathcal{I}, \mathcal{G})$ wrt. R is defined by setting $\mathcal{S}' = \{u \in \mathcal{S} \mid R \cap u\}$, $\mathcal{G}' = R \cap \mathcal{G}$, and $\mathcal{A}' = \{a' = (P \cap R, A \cap R, D \cap R) \mid a = (P, A, D) \in \mathcal{A}\}$. This definition naturally extends to finite domain planning.

Algorithm 2 Symbolic-SSSP-PDB-Construction

```
Input: Abstract state space problem $\mathcal{P}' = (\mathcal{S}', \mathcal{A}', \mathcal{I}', \mathcal{G}')$ in symbolic form
       with $\mathcal{G}'(x)$, $Trans'_a(x, x')$, shortest path locality $L$
Output: Shortest-path symbolic pattern database

$PDB[0](x') \leftarrow \mathcal{G}'(x')$
for all $f = 0, \ldots, f_{\max}$
  for all $l = 1, \ldots, L$ with $g - l \geq 0$
    $PDB[g](x') \leftarrow PDB[g](x') \setminus PDB[g-l](x')$
  $Min(x') \leftarrow PDB[f](x')$
  for all $i = 1 \ldots, C$
    $Succ_i(x) \leftarrow \bigvee_{a \in \mathcal{A}', c(a)=i} (\exists x' (Min(x') \wedge Trans'_a(x, x'))$
    $Succ_i(x') \leftarrow \exists x (Succ_i(x) \wedge x = x')$
    $PDB[f+i](x') \leftarrow PDB[f+i](x') \vee Succ_i(x')$
```

A shortest-path symbolic pattern database is the outcome of a backward exploration in abstract space. The pseudocode of the construction is shown in Algorithm 2. We see that up to L many previous layers are subtracted before a bucket is expanded. Multiple pattern database can be combined by either taking the maximum (always applicable), or the sum of individual pattern database entries (only applicable if the pattern databases are additive) (Korf & Felner 2002). One possible approach to select a disjoint pattern partition automatically (Edelkamp 2001; Haslum, Bonet, & Geffner 2005) uses bin packing to divide the state vector into parts $R_1, \ldots, R_k$, with $R_i \neq R_j$ for $i \neq j$. It restricts the candidates for $R_1, \ldots, R_k$ to the ones with an expected pattern database size (e.g. $2^{|R_1|} \cdot \ldots \cdot 2^{|R_k|}$) smaller than a pre-specified memory limit M.

The effectiveness of a pattern database heuristic can be predicted by its mean. In most search spaces, a linear gain in the mean corresponds to an exponential gain in the search. The mean can be determined by sampling the problem space or by constructing the pattern database. For computing the strength for a multiple pattern databases we compute the mean heuristic value for each of the databases individually and add (or maximize) the outcome. More formally, if PDB_i is the i-th pattern database in the disjoint set, $i \in \{1, \ldots, k\}$, then the *strength* of a disjoint pattern database set is

$$\sum_{i=1}^{k} \sum_{j=0}^{\max_h} j \cdot |PDB_i[j]| / \sum_{j=0}^{\max_h} |PDB_i[j]|.$$

The definition applies to both unweighted and weighted pattern databases. Once the BDD for $PDB_i[j]$ is created, $|PDB_i[j]|$ can be efficiently computed (model counting).

Heuristic Symbolic Planning

BDDA* (Edelkamp & Reffel 1998) can be casted as a variant of the BDD-based implementation of Dijkstra's algorithms with consistent heuristics. The algorithm was invented in the context of solving the single-agent challenges. ADDA* developed by Hansen, Zhou, & Feng (2002) is an alternative implementation of BDDA* with ADDs, while SetA* by Jensen, Bryant, & Veloso (2002)introduces branching partitioning. Symbolic branch-and-bound search has been proposed by Jensen *et al.* (2006). In an experimental study Qian & Nymeyer (2003) suggest that weaker heuristics perform often better.

The unified symbolic A* algorithm we consider uses a two-dimensional layout of BDDs. The advantage is that each state set already has the g- and the h-value attached to it, and such that arithmetics to compute f-values for the set of successors are not needed. To ensure completeness of the algorithm, we subtract previous buckets from the search. In order to save RAM, all buckets $Open[g, h]$ can be maintained on disk (Edelkamp 2005).

Algorithm 3 Shortest-Path-A*.

```
Input: State space planning problem $\mathcal{P} = (\mathcal{S}, \mathcal{A}, \mathcal{I}, \mathcal{G})$ in symbolic form
       with $\mathcal{I}(x)$, $\mathcal{G}(x)$, and $Trans_a(x, x')$, shortest-path locality $L$
Output: Optimal solution path

for all $h = 0, \ldots, h_{\max}$
  $Open[0, h](x) \leftarrow Evaluate(\mathcal{I}, h)$
for all $f = 0, \ldots, f_{\max}$
  for all $g = 0, \ldots, f$
    $h \leftarrow f - g$
    for all $l = 1, \ldots, L$ with $g - l \geq 0$
      $Open[g, h](x) \leftarrow Open[g, h](x) \setminus Open[g-l, h](x)$
    $Min(x) \leftarrow Open[g, h](x)$
    if $(Min(x) \wedge \mathcal{G}(x) \neq \bot)$
      return $Construct(Min(x) \wedge \mathcal{G}(x))$
    for all $i = 1, \ldots, C$
      $Succ_i(x') \leftarrow \exists x \bigvee_{a \in \mathcal{A}, c(a)=i} (Min(x) \wedge Trans_a(x, x'))$
      $Succ_i(x) \leftarrow \exists x (Succ_i(x') \wedge x = x')$
      for each $h \in \{0, \ldots, h_{\max}\}$
        $Open[g+d, h](x) \leftarrow Open[g+d, h](x) \vee Evaluate(Succ_i, h)$
return false
```

In the extension of BDDA* to weighted graphs shown in Algorithm , we determine all successors of the set of states with minimum f-value, current cost total g and action cost i. It remains to determine their h-values by a lookup in a multiple pattern database. The main problem is to merge the individual pattern database distributions into one. Inferring a joint distribution prior to the search is involved; it may easily exceed the time and space needed for searching the problem.

Therefore, we have decided to perform the lookup and the combination of multiple pattern databases entries on-the-fly for each encountered successor set $Succ_i$. Algorithm shows a possible implementation for additive costs. An algorithm for maximizing pattern database costs simply substitutes $i_1 + \ldots + i_k = h$ with $\max\{i_1, \ldots, i_k\} = h$.

Theorem 5 *(Optimality and Complexity of Algorithm) For*

Algorithm 4 Evaluate

```
Input: State set $States(x)$, value $h$
Global: Disjoint pattern databases $PDB_1, \ldots, PDB_k$
Output: $Result(x)$ representing subset of $States(x)$ with
        $h = PDB_1(x) + \ldots + PDB_k(x)$

$Result(x) \leftarrow \bot$
for each $i_1, \ldots, i_k$ with $i_1 + \ldots + i_k = h$
    $Result(x) \leftarrow Result(x) \vee (States(x) \wedge PDB_1[i_1](x) \wedge \ldots \wedge PDB_k[i_k](x))$
return $Result(x)$
```

transition weights $w \in \{1, \ldots, C\}$, the symbolic version of algorithm A on a two-level bucket priority queue operates finds the optimal solution with at most $O(C \cdot (f^*)^2)$ full and $O(C \cdot |\mathcal{A}| \cdot (f^*)^2)$ partitioned images, where f^* is the optimal solution cost.*

Proof: Optimality and completeness of BDDA* are inherited from explicit-state A*. As the g- and the h-value are both bounded by f^* it computes at most $O(C \cdot (f^*)^2)$ full and $O(C \cdot |\mathcal{A}| \cdot (f^*)^2)$ partitioned images.

To reconstruct the solution with the same algorithm suggested for Dijkstra's search we may unify the all (g, i)-buckets, $0 \leq i \leq h$ into one g-layer. If memory becomes sparse, similar to breadth-first heuristic search (Zhou & Hansen 2004), a recursive reconstruction based based on relay layers can be preferable, and as said, relay layers can be avoided by using disk space.

So far we have extended the symbolic versions of Dijkstra's algorithm and A* to deal with actions a of cost zero. In the concrete state space zero-cost operators are a natural concept, as actions can be transparent to the optimization criterion (e.g. boarding and debarking passengers while minimizing the total fuel-consumption of a plane). For the construction of disjoint pattern databases zero-cost operators are introduced when an action has effects in more than one abstractions. To avoid multiple counting of the costs of an action the cost of the abstract action is set to zero in all but one abstraction before constructing the pattern databases. This way the sum of the abstraction remains admissible.

Dijkstra's shortest path algorithm remains correct for problem graphs with edge cost zero, but the introduction of zero-cost actions in the one-level bucket-based implementation has to be dealt with care. In essence, the algorithm stay in a bucket for some time, which requires to separate the search frontier in a bucket from the set of expanded nodes.

For the construction of symbolic pattern databases the following solution introducing zero-cost actions turns out to be sufficient. It performs BFS to compute the closure for each bucket: once a zero-cost image is encountered for a bucket to be expanded, a fixpoint is computed. This results in the representation all states that are reachable by applying one non-zero cost action followed by a sequence of zero-cost actions. As a result the constructed pattern databases are admissible even if the partition into patterns does not perfectly separate the set of actions.

Results

As benchmarks for shortest path planning we casted temporal planning problems from the 2002 and 2006 international planning competitions (IPC-3 and IPC-5) as cost-optimization optimization problems, minimizing sequential total time, interpreted as the sum of the individual durations over all actions in the plan[6]

Unfortunately, none of the step-optimal planners yet includes action costs[7]. Therefore – despite the difference in the plan objective – we decided to cross-compare the performance of our BDD planning approach with the state-of-the-art temporal planner CPT by Vidal & Geffner.

CPT computes the *makespan*, i.e., the optimal duration of a parallel plan. As in propositional planning, the best parallel plan does not imply the best sequential plan, or vice versa. As the search depth and variation of states increase, it is likely that finding the optimal duration of a sequential plan is the *harder* optimization problem[8 9 10].

Problem	Mincost	MIPS-BDD	CPT	TP4	Makespan
1	173	0.96	0.02	0.07	173
2	642	1.15	0.07	0.28	592
3	300	1.23	0.09	0.43	280
4	719	9.29	1.09	>3,600	522
5	500	2.77	0.44	30.54	400
6	550	13.19	0.35	4.87	323
7	1,111	178	3.07	>3,600	665
8	942	1,345	17.52	>3,600	522
9	$\geq$ 1,286	>3,600	90.64	>3,600	522

Table 1: Results in *ZenoTravel*, *SimpleTime* (IPC-3).

Table 1 shows the results in *ZenoTravel*[11]. The symbolic implementation of Dijkstra's algorithms solved the first 7 problems. Value 1,111 shows that the approach scales to larger action cost values. For Problem 8 it generated a plan of quality 949, but (while terminated at cost value 820), it could not prove optimality within 1 hour. On the other hand, BDDA* (with bin packing) solved problem 8 (incl. pattern database construction) in about 20min. Sequential costs raised to over twice the parallel costs but finding plans took more time, such that larger problems could not be solved. In such cases we provide lower bounds.

Table 2 shows the results in *DriverLog*. Here we experimented with Dijkstra's algorithm only. For this case the

[6]As we don't know L for most of the planning domains, for pattern construction we used full duplication detection (subtracting all previous layers), for Dijkstra search we used no duplicate pruning at all, while for A* search we imposed a sufficient large number of $L = 10$ (not affecting the optimality).

[7]We are aware of one current implementation efforts for such planner by Menkes van den Briel. Unfortunately, due to the unfinished status of the planner, we were not able to cross-compare.

[8]For relaxed plans it is known that finding the sequential optimal plan is NP-hard, while finding the parallel optimal plan is polynomial (Hoffmann & Nebel 2001).

[9]For these experiments we imposed a time limit of 1 hour and a memory limit of 2 GB. The reference computer for CPT is has a 2.8GHz CPU equipped with 1 GB RAM (Vidal & Geffner 2006).

[10]We interpreted `total-time` as `total-cost` and avoid enforcing sequential plans as such comparison would likely be unfortunate for CPT.

[11]CPT also solved instance 10 and 11

Problem	Mincost	MIPS-BDD	CPT	TP4	Makespan
1	92	0.92	0.02	4.18	91
2	166	1.92	0.02	365.89	92
3	83	2.42	0.03	0.18	40
4	134	36.40	>3,600	>3,600	-
5	109	8.07	40.67	>3,600	51
6	107	100.57	>3,600	>3,600	-
7	84	60.23	0.43	45.52	40
8	153	3,256	>3,600	>3,600	-
9	120	3,284	>3,600	>3,600	-
10	72	1,596	6.16	>3,600	38
11	84	799	>3,600	>3,600	-
12	$\geq$ 115	>3,600	>3,600	>3,600	-

Table 2: Results in *DriverLog*, *SimpleTime* (IPC-3).

comparison with CPT/TP4 shows that, even though slower in simple instances, the weighted BDD approach solves more benchmarks. We experimented in the *Time* (instead of *SimpleTime*) domains, too. The costs for the first 7 problems are 303 (1.01s), 407 (7.63s), 215 (1.93s), 503 (59.71s), 182 (18.44s), 336 (129s), and 380 (1,715s).

Problem	Mincost	MIPS-BDD	CPT	TP4	Makespan
1	38	0.98	0.02	0.08	28
2	57	2.91	0.50	19.73	36
3	84	419	>3,600	>3,600	-
4	82	3,889	>3,600	>3,600	-

Table 3: Results in *Depots* (IPC-3).

Table 3 shows the results in *Depots*. Problem 3 and 4 are a challenge for cost-optimal plan finding and could not be solved with Dijkstra's algorithm in one hour. BDDA*, however, finds the optimal solutions in time[12].

Problem	Mincost	MIPS-BDD	CPT	TP4	Makespan
1	48	1.09	0.01	0.01	46
2	72	4.90	1.19	466.63	70
3	60	4.28	0.06	1.17	34
4	96	5.41	0.82	>3,600	58
5	84	97.72	1.55	>3,600	36
6	108	88.11	0.28	>3,600	46
7	$\geq$113	>3,600	1.10	>3,600	34

Table 4: Results in *Sattelite*, *SimpleTime* (IPC-3).

Table 4 shows the results for *Sattelite*. Problem 5 and 6 are a challenge and could not be solved with Dijkstra's algorithm in one hour. While problem 3 is solved in 30.28s, problem 4 already required 1,425s. BDDA*, however, successfully finds cost optimal plans for the the two harder. CPT can solve more problems (it also solves problems 9 – 11) with a makespan that shrinks to less than a third of the optimal cost value. We successfully ran some experiments on *Time* formulations generating plans of costs of 10,000 and more.

[12]The search time for problem 4 lies within an hour if we subtract the construction time of the pattern database, about 6 min

Problem	Length	Mincost	MIPS-BDD	Makespan	CPT2
4	8	12	0.91	12	0.02
5	8	12	0.97	8	0.02
6	8	12	0.97	8	0.04
7	14	20	1.17	20	0.64
8	13	19	1.17	12	0.34
9	11	17	5.57	11	1.61
10	18	26	6.61	26	917
11	17	25	30.98	17	502
12	17	25	107	-	>3,600
13	18	28	83.20	28	1,159
14	19	29	576	17	52.99
15	18	30	266	18	24.76
16	22	34	1,685	-	>3,600
17	-	$\geq$ 39	>3,600	-	>3,600

Table 5: Results in *Storage*, *Time* (IPC-5).

Table 5 shows the results in the IPC-5 domain *Storage*. The planner we compare with is CPT2[13]. The symbolic implementation of Dijkstra's algorithms solved the first 15 problems cost-optimal, but problem 16 could not be solved in the allocated time slot. As additional orientation we provide the solution length (number of actions). On the other hand, BDDA* (with bin packing) could solve problem 16 in less than 30min with about 12min used for pattern database construction. Problem 17 exceeded our run-time limit, but provides a valuable lower bound.

Conclusion and Discussion

We extended BDD-based planning to include action costs. Symbolic weighted pattern databases were constructed based on a bucket implementations of Dijkstra's single-source shortest-paths algorithm and applied to help finding solutions to the concrete problems with symbolic A* search. The theoretical result on the duplicate detection scope for best-first search graphs additionally extends existing theory for breadth-first search graphs.

Partitioning along the action and cost is a compromise between the number and the hardness of computing images. If all costs were multiplied by a factor, then the only change would be that more intermediate empty buckets have to be scanned. Finding the next non-empty bucket, however, is fast compared to computing images of non-empty ones. On very sparse graph or very large action costs more advanced bucket implementations (e.g. radix-heaps) may apply. For cost model **C3**, costs are provided by a *state formula*, an expression over the action and the state's fluents (corresponding to PDDL 2, Level 2) or indicator variables (e.g. 1 for *true* and 0 for *false* for PDDL 3 propositions). One way of encoding such action cost model with BDDs are weighted transition relations. For each action a the weighted transition relation $\mathit{Trans}_a(i, x', x)$ evaluates to 1 if and only if the step from x' to x has cost $i \in \{1, \ldots, C\}$.

The advantage of BDD-based compared to SAT- and CSP-based planning is that the description complexity is independent wrt. to previous levels, while the complexities of the other two approaches raises with the search depth.

[13]Extension of CPT for IPC-5, operating on a 3 GHz CPU with 1 GB RAM limit and 30 minutes time bound.

References

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

Bacchus, F. 2001. The AIPS'00 planning competition. *aim* 22(3):47–56.

Biere, A.; Cimatti, A.; Clarke, E.; and Zhu, Y. 1999. Symbolic model checking without BDDs. In *Tools and Algorithms for the Construction and Analysis of Systems*.

Blum, A., and Furst, M. L. 1995. Fast planning through planning graph analysis. In *IJCAI*, 1636–1642.

Borowski, B., and Edelkamp, S. 2007. Optimal infinite-state planning with presburger automata. In *Planning and Configuration (PUK)*.

Botea, A.; Müller, M.; and Schaeffer, J. 2005. Learning partial-order macros from solutions. In *ICAPS*, 231–240.

Bryant, R. E. 1985. Symbolic manipulation of boolean functions using a graphical representation. In *DAC*, 688–694.

Cimatti, A.; Roveri, M.; and Traverso, P. 1998. Automatic OBDD-based generation of universal plans in non-deterministic domains. In *AAAI*, 875–881.

Culberson, J. C., and Schaeffer, J. 1998. Pattern databases. *Computational Intelligence* 14(4):318–334.

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

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

Edelkamp, S., and Helmert, M. 1999. Exhibiting knowledge in planning problems to minimize state encoding length. In *ECP*, 135–147.

Edelkamp, S., and Helmert, M. 2001. MIPS: the model-checking integrated planning system. *AI Magazine* 22(3):67–72.

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

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

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

Edelkamp, S. 2003. Taming numbers and durations in the model checking integrated planning system. *Journal of Artificial Intelligence Research* 20:195–238.

Edelkamp, S. 2005. External symbolic heuristic search with pattern databases. In *ICAPS*, 51–60.

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

Fox, M., and Long, D. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research* 20:61–124.

Gerevini, A., and Long, D. 2005. Plan constraints and preferences in PDDL3. Technical report, Department of Electronics for Automation, University of Brescia.

Grandcolas, S., and Pain-Barre, C. 2007. Filtering, decomposition and search-space reduction in optimal sequential planning. In *AAAI*.

Hansen, E. A.; Zhou, R.; and Feng, Z. 2002. Symbolic heuristic search using decision diagrams. In *SARA*, 83–98.

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

Haslum, P.; Bonet, B.; and Geffner, H. 2005. New admissible heuristics for domain-independent planning. In *AAAI*, 1163–1168.

Helmert, M., and Röger, G. 2007. How good is almost perfect? In *ICAPS-Workshop on Heuristics for Domain-Independent Planning*.

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

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

Hickmott, S.; Rintanen, J.; Thiebaux, S.; and White, L. 2006. Planning via petri net unfolding. In *ECAI-Workshop on Model Checking and Artificial Intelligence*.

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

Hoffmann, J.; Sabharwal, A.; and Domshlak, C. 2006. Friends or foes? An AI planning perspective on abstraction and search. In *ICAPS*, 294–304.

Jensen, R.; Hansen, E.; Richards, S.; and Zhou, R. 2006. Memory-efficient symbolic heuristic search. In *ICAPS*, 304–313.

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

Jensen, R. 2003. *Efficient BDD-based planning for non-deterministic, fault-tolerant, and adversarial domains*. Ph.D. Dissertation, Carnegie-Mellon University.

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

Kautz, H.; Selman, B.; and Hoffmann, J. 2006. Satplan: Planning as satisfiability. In *Proceedings of the* 5^{th} *International Planning Competition*.

Korf, R. E., and Felner, A. 2002. *Chips Challenging Champions: Games, Computers and Artificial Intelligence*. Elsevier. chapter Disjoint Pattern Database Heuristics, 13–26.

Korf, R. E.; Zhang, W.; Thayer, I.; and Hohwald, H. 2005. Frontier search. *Journal of the ACM* 52(5):715–748.

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

Qian, K., and Nymeyer, A. 2003. Heuristic search algorithms based on symbolic data structures. In *ACAI*, 966–979.

Qian, K. 2006. *Formal Verification using heursitic search and abstraction techniques*. Ph.D. Dissertation, University of New South Wales.

Vidal, V., and Geffner, H. 2006. Branching and pruning: An optimal temporal pocl planner based on constraint programming. *Artificial Intelligence* 170(3):298–335.

Xing, Z.; Chen, Y.; and Zhang, W. 2006. Maxplan: Optimal planning by decomposed satisfiability and backward reduction. In *Proceedings of the* 5^{th} *International Planning Competition*.

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