Symbolic Shortest Path Planning

Stefan Edelkamp

Forschungsbericht Nr. 814
April 2007

Symbolic Shortest Path Planning

Stefan Edelkamp
Computer Science Department
University of Dortmund, Germany

April 26, 2007

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 show that it matches the duplicate detection scope in best-first search graphs. Cost-optimal plans for compiled competition benchmark domains are computed.

1 Introduction

Action costs are a very natural search concept. In many applications, costs can only be positive integers (sometimes for fractional values it is also possible and beneficial to achieve this by rescaling). As an example, take macros of actions [31], which dramatically reduce the search efforts for finding an optimal plan [29]. Unfortunately, existing planners that operate on automatically inferred macros [4] are not optimal.

In this paper we look at deterministic planning problems $\mathcal{P} = (\mathcal{S}, \mathcal{A}, \mathcal{I}, \mathcal{G})$, for which the output is a cost-optimal sequence of actions (the shortest path) $\pi = (a_1 \ldots, a_k)$ with $a_i \in \mathcal{A}$ that leads from the (set of) initial state(s) $\mathcal{I} \subseteq \mathcal{S}$ to the planning goal $\mathcal{G} \subseteq \mathcal{S}$.

The objective is to minimize the sum of the costs of all actions in the plan π. The following cost models have been proposed:

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 [17] full metric planning in cost model C4 [18], and preference constraints [19], which penalize plans that go through certain states in cost models C3 and C4. According to a current proposal, tackling cost model C2 is one central aim for the deterministic part of the next international planning competition.

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

Symbolic planning is often based on analyzing planning graphs [3] or by checking the satisfiability of formulas [30, 2]. Here, we refer to symbolic exploration only in the context of using BDDs [5]. While invented in model checking, BDDs contribute to many successful AI planning systems [6, 26, 27, 13]. The idea is to lessen the costs associated with the exponential memory requirements for the state sets involved as problem sizes get bigger.

This paper contributes symbolic 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. The exploration is extended to construct symbolic shortest-path pattern databases, and their use in A* search.

The paper is structured as follows. First we recall symbolic search algorithms that have been applied so far and draw some initial experiments. We then discuss the symbolic design and implementation of Dijkstra's single-source shortest-paths algorithm and its complexity. Next we address symbolic shortest-path pattern databases for planning and their construction for cost model C2. In order 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 weighted symbolic symbolic pattern databases into one, and discuss greedy partition of patterns into disjoint sets. In the experiments we provide promising results for cost-optimizing variants of existing planning competition benchmarks.

2 Symbolic Planning

With symbolic planning we denote implicit search algorithms that represent sets of planning states in form of Boolean functions. All symbolic algorithms assume a binary encoding of a planning problem.

Representing fixed-sized state vectors (of finite-domain) in binary is uncomplicated. For example, the Lloyd's 15-Puzzle [36] can be easily encoded in $16 \times 4 = 64$ bits, with 4 bits encoding the label of the tile. A more concise description is the binary code for the ordinal number of the permutation associated with the puzzle state yielding $\lceil \log 16! \rceil = 45$ bits. For the PSPACE-hard Sokoban [7] problem we also have different options: either we encode the position of the balls individually or we encode their layout on the board. For propositional action planning we can encode the atoms that are valid in a given planning state individually by using the binary representation of their ordinal number, or via the bit vector of atoms being true and false. More generally, a state vector can be represented by encoding the domains of the vector individually, or – assuming a perfect hash function of a state vector – using the binary representation of the hash address.

Given a fixed-length binary encoding for the state vector of a search problem, characteristic functions represent state sets. Such function evaluates to true for the binary representation of a given state vector if and only if the state is a member of that set. As the mapping is 1-to-1, the characteristic function can be identified with the state set itself.

Compared to the space requirements of explicit-state search algorithms, symbolic search algorithms save space by exploiting a shared representation of state sets, which is often considerably smaller than its full enumeration. This has a drastic impact on the design of available algorithms, as not all algorithms adapt to the exploration of state sets. As one feature, all symbolic search algorithm operate on sets of initial states, reporting if there is a plan leading from one initial state to one of the goal states.

Explicit-State Concept	Symbolic Concept
State Set S	Characteristic Function $S(x)$
Search Frontier *Open* Expanded States *Closed*	Characteristic Function *Open*(x) Characteristic Function *Closed*(x)
Initial State(s) $\mathcal{I}$ Goal $\mathcal{G}$	Characteristic Function $\mathcal{I}(x)$ Characteristic Function $\mathcal{G}(x)$
Action a Action Costs $c(a)$	Transition Relation *Trans*$_a(x, x')$ Transition Relation *Trans*$_a(c, x, x')$
Action Set $\mathcal{A}$	Transition Relation *Trans*(x, x')
succ$(u) = \{v \in \mathcal{S} \mid \exists a \in \mathcal{A} : a(u) = v\}$ Heuristic h	Characteristic Function *Succ*(x) Heuristic Relation *Heur*$(value, x)$

Table 1: Comparison of concepts in explicit-state and symbolic search.

Symbolic search executes a functional exploration of the problem graph. This functional representations of states and actions then allow us to compute the functional representation of a set of successors, or the *image*, in a specialized operation. As a byproduct, the functional representation of the set of predecessors, or the *preimage*, can also be efficiently determined.

Table 1 relates the concepts needed for explicit state search to their symbolic counter-parts. Individual transition relations *Trans*$_a(x, x')$ allow to keep *Trans*(x, x') in a partitioned form. Extended transition relations *Trans*(c, x, x') include the costs of actions encoded in binary. Heuristic relations *Heur*$(value, x)$ partition the state space according the heuristic values encoded in *value*. As a feature, all the algorithms in this chapter work for *initial state sets*, reporting a path from one member to the goal. For the sake of coherence, we nonetheless stick to singletons.

As said, symbolic state space search algorithms use Boolean functions to represent sets of states. According to the space requirements of ordinary search algorithms, they save space mainly by sharing parts of the state vector. Different to other compressed dictionary data structures sharing is realized by exploiting a functional representation of the state set. For example, the set of all states in the $(n^2 - 1)$-Puzzle with the blank located on the second or fourth position is represented by the (characteristic) function $\phi(t_0, \ldots, t_{n^2-1}) := (t_1 = 0) \vee (t_3 = 0)$. The characteristic function of a state set can be much smaller than the number of states it represents. The main advantage of symbolic search algorithms that they operate on the functional representation of both state and actions.

We refer to the implicit representation of state and action sets in a data structure as their *symbolic representation*. We select BDDs as the appropriate data structure for characteristic functions. BDDs are directed, acyclic, and labeled graphs. Roughly speaking, these graphs are interpreted deterministic finite-state

automata, accepting the state vectors (encoded in binary) that are contained in the underlying set. In a scan of a state vector starting at the start node of the BDD at each intermediate BDD node, a state variable is processed, following a fixed variable ordering. The scan either terminates at a (non-accepting) leaf labeled *false* (or 0), which means that a state is not contained in the set, or at a(n accepting) leaf labeled *true* (or 1), which means that the state is not contained in the set. Compared to a host of ambiguous representations of Boolean formulas, the BDD representation is unique. As in usual implementations of BDD libraries, different BDDs share their structures. Such libraries have efficient operations of combining BDDs and subsequently support the computation of images. Moreover, BDD packages often support arithmetic operations on variables of finite domains with BDDs. To avoid notational conflicts in this chapter, we denote nodes of the problem graph as *states* and vertices of the BDDs as *nodes*.

Compared to the space requirements of explicit-state search algorithms, symbolic search algorithms save space by exploiting a shared representation of state sets, which is often considerably smaller than its full enumeration. This has a drastic impact on the design of available algorithms, as not all algorithms adapt to the exploration of state sets. As one feature, all symbolic search algorithm operate on sets of initial states, reporting if there is a plan leading from one initial state to one of the goal states.

Transitions are also 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. In other words what we are really interested in, is image of a state set S with respect to a transition relation *Trans*, which is equal to applying the following operation

$$\mathit{Image}(x') = \exists x \ (\mathit{Trans}(x, x') \wedge S(x)),$$

where $S(x)$ denotes the characteristic function of set S. The result is a characteristic function of all states reachable from the state in S in one step. Iterating the process (starting with the representation of the initial state(s)) yields a symbolic implementation of breadth-first search (BFS).

Fortunately, by keeping sub-relations Trans_a separated and 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

$$\mathit{Image}(x') = \bigvee_{a \in \mathcal{A}} \left(\exists x \ (\mathit{Trans}_a(x, x') \ \wedge \ S(x))\right).$$

3 Step- and Cost-Optimal Symbolic Planning

So far, uni- and bidirectional BFS as well as different implementations of A* [16, 20, 28, 34], and memory-limited branch-and-bound by [27] have been applied to solve propositional planning problems with symbolic search step-optimally. In the context of introducing preference constraints in PDDL3 [19], cost-optimal symbolic BFS for linear, non-monotone cost functions generates the entire search space, incrementally improving the solution quality with increasing depth [15].

3.1 Bidirectional Breadth-First Search

In a symbolic variant of BFS we determine the set of states S_i reachable from the initial state s in i steps. The search is initialized with start state $\mathcal{I}$. The following equation determines S_i given both S_{i-1} and the transition relation:

$$S_i(x') = \exists x \ (S_{i-1}(x) \wedge \mathit{Trans}(x, x')).$$

The formula calculating the successor function is a relational product. Informally, a state x' belongs to S_i if it has a predecessor x in the set S_{i-1} and there exists an operator which transforms x into x'. Note that the right hand side of the equation depends on x compared to x' on the left hand side. Thus, it is necessary to substitute x' with x in for the next iteration. In case of an interleaved representation there is no needs to reorder or reduce, and the substitution can be achieved by a textual replacement of the node labels in the BDD.

In order to terminate the search we test whether or not a state is represented in the intersection of the set S_i and the set of goal states G. Since we enumerated $S_0, \ldots, S_{i-1}$ the iteration index i is known to be the optimal solution length.

As a byproduct for symbolic search for the construction of symbolic pattern databases we have already seen the advantage of the transition relation *Trans* to perform backward search. Recall that for state sets S_i we successively determine the preimages of the goal set by computing

$$S_i(x) = \exists x' \ (S_{i+1}(x') \wedge \mathit{Trans}(x, x'))$$

for a decreasing index i. As the search is symbolic large goal sets do not impose a burden to the search process.

In bidirectional breadth-first search, forward and backward search are carried out concurrently. On the one hand we have the symbolic forward search frontier F_f with $F_0 = \mathcal{I}$ and on the other hand the backward search frontier B_b with $B_0 = \mathcal{G}$. When the two search frontiers meet ($\phi_{F_f} \wedge \phi_{B_b} \not\equiv \bot$) we have found an optimal solution of length $f + b$. With the two horizons *Open*$^+$ and *Open*$^-$ the algorithm is implemented in pseudo code in Algorithm 1.

In a graph with uniform weights, the number of iterations remains equal to the optimal solution length f^*. Solution reconstruction now proceeds from the established intersection to the respective starting states.

Algorithm 1 Symbolic-Bidirectional-BFS

Input: State space problem with *Trans*, $\mathcal{G}$ and $\mathcal{I}$
Output: Optimal solution path

$Open^+(x') \leftarrow \mathcal{I}(x')$; $Open^-(x') \leftarrow G(x')$
while $(Open^+(x') \wedge Open^-(x') \equiv \bot)$
  if (*forward*)
    $Open^+(x) \leftarrow \exists x' \ ((x = x') \ \wedge \ Open^+(x'))$
    $Succ(x') \leftarrow \exists x \ (Open^+(x) \wedge Trans(x, x'))$
    $Open^+(x') \leftarrow Succ(x')$
  else
    $Pred(x) \leftarrow \exists x \ (Open^-(x') \wedge Trans(x, x'))$
    $Open^-(x) \leftarrow Pred(x)$
    $Open^-(x') \leftarrow \exists x \ ((x = x') \ \wedge \ Open^-(x'))$
return $Construct(Open^+(x') \wedge Open^-(x'))$

The choice of the search direction (function call `forward`) is crucial for a successful exploration. There are three simple criteria: BDD size, the number of represented states, and smaller exploration time. Since the former two are not well suitable to predict the computational efforts of the next iteration the third criterion should be preferred.

In our planner MIPS-BDD we implemented the above methods, choosing a binary encoding of a minimized state description inferred by [22]. We compare bidirectional symbolic BFS with other optimal planners in domains that enforce minimal-step plans. (We selected the top-performing competitors from the last international planning competition (IPC-5) to compare with, instead of choosing other step-optimal planners [38, 23].) The results are obtained with matching CPU and memory limits are presented in Figure 1. For this domain, we see a clear advantage of applying BDD technology.

3.2 Cost-Optimal Search

Symbolic BFS finds the optimal solution in the number of solution steps. BDDs are also capable of optimizing a cost functions f over the problem space space-efficiently. In this section, we do not make any specific assumption about f (such

	MIPS-BDD		SAT-PLAN		MAX-PLAN		CPT2	
Problem	Steps	Time	Steps	Time	Steps	Time	Steps	Time
1	23	1.02s	-	-	23	979s	-	-
2	23	0.98s	-	-	23	1,353s	-	-
3	23	1.00s	-	-	23	1,148s	-	-
4	23	0.99s	-	-	23	841s	-	-
5	23	1.00s	-	-	23	1,438s	-	-
6	45	3.33s	-	-	-	-	-	-
7	46	3.44s	-	-	-	-	-	-
8	87	1,132s	-	-	-	-	-	-
9	87	544s	-	-	-	-	-	-

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

as to be monotone or being composed of g or h), except that f operates on variables of finite domains. The problem has become prominent in the area of (over-subscribed) action planning, where a cost function encodes and accumulates the desire for the satisfaction of soft constraints on planning goals, which has to be maximized. As an example, consider that additionally to an ordinary goal description, we prefer certain blocks in Blocksworld to be placed on the table. For the sake of simplicity, we restrict ourselves to minimization problems. This implies that we want to find the path to a goal state that has the smallest f-value.

To compute a BDD $F(\mathit{value}, x)$ for the cost function $f(x)$ over a set finite domain state variables $x = (x_1, \ldots, x_k)$ with $x_i \in [\min_{x_i}, \max_{x_i}]$, we first compute the minimum and maximum values that f can take. This defines the range $[\min_f, \max_f]$ that has to be encoded in binary. For example if f is a linear function $\sum_{i=1}^{k} a_i x_i$ with $a_i \geq 0$, $i \in \{1, \ldots, k\}$ then $\min_f = \sum_{i=1}^{k} a_i \min_{x_i}$ and $\max_f = \sum_{i=1}^{k} a_i \max_{x_i}$.

To construct $F(\mathit{value}, x)$ we build a sub-BDDs $\mathit{Partial}(\mathit{value}, x)$ with *value* representing $a_i x_i$, $i \in \{1, \ldots, k\}$, and combine the intermediate results to the relation $F(\mathit{value}, x)$ using the relation Add. As the a_i are finite the relation $\mathit{Partial}(\mathit{value}, x)_i$ can be computed using $\mathit{value} = x_i + \ldots + x_i$ (a_i times) or adapt the ternary relation *Mult* (to be constructed similar to *Add*). This shows that all operations to construct F can be realized using finite-domain arithmetics on BDDs. Actually, there is an option of constructing the BDD for a linear function directly from looking at the coefficients in $O(\sum_{i=0}^{n} |a_i|)$ time and space.

Algorithm 2 displays the pseudo-code for symbolic BFS incrementally improving an upper bound U on the solution cost. The algorithm applies symbolic BFS until the entire search space has been traversed and stores the currently optimal solution. As before state sets are represented in form of BDDs. Additionally,

Algorithm 2 Cost-Optimal-Symbolic-BFS

Input: State space problem with transition relation *Trans* and cost relation F
Output: Cost-optimal solution path

$U \leftarrow \max_f$
loop
  $\mathit{Closed}(x) \leftarrow \mathit{Open}(x) \leftarrow \mathcal{I}(x)$
  $\mathit{Intersection}(x) \leftarrow \mathcal{I}(x) \wedge \mathcal{G}(x)$
  $\mathit{Bound}(\mathit{value}, x) \leftarrow F(\mathit{value}, x) \wedge \bigvee_{i=\min_f}^{U}(\mathit{value} = i)$
  $\mathit{Eval}(\mathit{value}, x) \leftarrow \mathit{Intersection}(x) \wedge \mathit{Bound}(\mathit{value}, x)$
  while $(\mathit{Eval}(\mathit{value}, x) \neq \bot)$
    if $(\mathit{Open}(x) = \bot)$ **return** "Exploration completed"
    $\mathit{Succ}(x) \leftarrow \exists x\, (\mathit{Trans}(x, x') \wedge \mathit{Open}(x))$
    $\mathit{Succ}(x) \leftarrow \exists x'(\mathit{Succ}_i(x') \wedge x = x')$
    $\mathit{Open}(x) \leftarrow \mathit{Succ}(x) \wedge \neg \mathit{Closed}(x)$
    $\mathit{Closed}(x) \leftarrow \mathit{Closed}(x) \vee \mathit{Succ}(x)$
    $\mathit{Intersection}(x) \leftarrow \mathit{Open}(x) \wedge \mathcal{G}(x)$
    $\mathit{Eval}(\mathit{value}, x) \leftarrow \mathit{Intersection}(x) \wedge \mathit{Bound}(\mathit{value}, x)$
    if $(\mathit{Eval}(\mathit{value}, x) \neq \bot)$
      for each $i \in \{\min_f, \ldots, U\}$
        if $(F(\mathit{value}, x) \wedge (\mathit{value} = i) \wedge \mathit{Eval}(\mathit{value}, x) \neq \bot)$
          $U \leftarrow i - 1$
          $\mathit{sol} \leftarrow \mathit{Construct}(\mathit{Eval}(\mathit{value}, x))$
          break
return *sol*

the search frontier is reduced to those states that have a cost value of at most U. In case an intersection with the goal is found, the breadth-first exploration is suspended to construct solution with the smallest f-value for states in the intersection. The cost gives a new upper bound U denoting the quality of the currently best solution minus 1. After the minimal-cost solution has be found, the breadth-first exploration is resumed.

Theorem 1 *(Optimality of Cost-Optimal Symbolic BFS) The plan constructed by* cost-optimal symbolic BFS *has minimum cost. The number of images is bounded by the radius (maximum BFS-level) of the underlying problem graph.*

Proof: The algorithm applies duplicate detection and traverses the entire state space. It generates each possible state exactly once. Eventually, the state of minimum f-value will be encountered. Only those goal states are abandoned from

the cost evaluation that have an f-value larger than or equal to the current best solution value. The exploration terminates if all BFS-Layers have been generated.

In Figure 2 we validated that our planner BDD-MIPS is capable of computing optimal solutions in a planning domains with preferences and that it can produce 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 the optimality was proven after 4,607s.

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

Figure 2: Cost-optimal symbolic search in *Storage*, *Qualitative Preferences* (IPC-5).

4 Symbolic Shortest Paths Planning

The single-source shortest-paths search algorithm of Dijkstra finds a plan with minimized total cost [10]. For positive action costs, the first plan reported is provably optimal. For implicit graphs an implementation of Dijkstra's algorithm requires two data structures, one to access nodes in the search frontier and one to detect duplicates. For symbolic search, the second aspect is less dominating, but will be addressed later.

As BDDs allow sets of states to be represented efficiently, the priority queue of a search problem with integer-valued cost function can be partitioned to a list of buckets $\mathit{Open}[0], \ldots, \mathit{Open}[f_{\max}]$. We assume that the largest action cost (inducing the difference between the largest and smallest key) is bounded by some constant C. The pseudo-code is shown in Algorithm 7.

The algorithm works as follows. The BDD *Open* is set to the representation of the start state(s) with f-value 0. Unless at least one goal state is reached, in one iteration we first choose the next f-value 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 $\mathit{Trans}_a(x, x')$ is applied to determine the BDD for the subset of

Algorithm 3 Symbolic-Shortest-Path (Cost Model 2).

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 $\mathit{Trans}_a(x, x')$
Output: Optimal solution path

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

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 is a one-level bucket [9]. This priority queue implementation consists of an array of size $C + 1$, each of which is the link to a BDD for the elements.

Let us briefly consider possible implementations for *Construct*. If all previous layers remain in main memory, sequential solution reconstruction is sufficient. If buckets are eliminated as in frontier search [33] or breadth-first heuristic search [38], 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 [14]. 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 [1].

Theorem 2 *(Optimality and Complexity of Algorithm 3) 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 most $O(C \cdot f^*)$ full and $O(C \cdot |\mathcal{A}| \cdot f^*)$ partitioned images, where f^* is the optimal solution cost.*

Proof: The algorithm mimics the execution of the algorithm of Dijkstra in a one-level bucket structure. 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 in form of delayed duplicate detection in the next two sections.

5 Symbolic Pattern Databases

Abstraction is the key to the automated design of search heuristics. Applying abstractions simplifies a problem, and exact distances in theses 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 [8] 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 [12] 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 databases can be maintained by set of BDDs $\mathit{PDB}[0], \ldots, \mathit{PDB}[h_{\max}]$.

For the construction of a symbolic shortest-path pattern database, the symbolic implementation of Dijkstra's algorithm is adapted as follows. For a given problem abstraction, the symbolic pattern database BDD $\mathit{PDB}[0] \ldots, \mathit{PDB}[h_{\max}]$ is produced. 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 3, 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 $PDB[0] \vee \ldots \vee PDB[i-1]$ from the current layer $PDB[i]$. If memory is sparse, a strategy to reduce the duplicate detection scope becomes crucial.

6 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 [38]. In the following we generalize the concept from unweighted to weighted search graphs.

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 given as*

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

In unweighted graphs, we have $c(u, v) = 1$ for all $u, v \in \mathcal{S}$. Moreover, in undirected graphs $\delta(\mathcal{I}, u)$ and $\delta(\mathcal{I}, v)$ differ by at most 1 so that the locality is 2. (Due to more general setting, our definition for unweighted graphs is off by 1 compared to the definition of [38], where locality does not include the edge cost $\max\{\max_{u \in \mathcal{S}, v \in succ(u)}\{\delta(\mathcal{I}, u) - \delta(\mathcal{I}, v)\}, 0\}$).

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 3 *(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 $v \in succ(u)$. 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. The following theorem proves the existence of such a bound.

Theorem 4 *(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 5 *(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

$$\begin{aligned} L &\leq \max_{u \in \mathcal{S}, v \in \mathit{succ}(u)} \{\delta(v, u)\} + C = \max_{u \in \mathcal{S}, v \in \mathit{succ}(u)} \{\delta(u, v)\} + C \\ &= \max_{u \in \mathcal{S}, v \in \mathit{succ}(u)} \{c(u, v)\} + C = 2C. \end{aligned}$$

7 Automated Pattern Selection

In domain-dependent planning, the selection of abstraction functions is provided by the user. For domain-independent planning the system has to infer the abstractions automatically. Unfortunately, there is a huge number of feasible planning abstractions to choose from [25].

Nonetheless, first progress in computing abstractions automatically has been made [21]. 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 4 Symbolic-SSSP-PDB-Construction

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

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

A pattern database *PDB* is the outcome of a symbolic backward shortest paths exploration in abstract space. The pseudo-code of the algorithm of Dijkstra for shortest-path symbolic pattern database construction is shown in Algorithm 4. We see that up to L many previous layers are subtracted before a bucket is expanded.

More than one 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 disjoint) [32].

There are different approaches to select a disjoint pattern partition automatically. One possible approach, suggested by [11] and [21], 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

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

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).

8 Heuristic Symbolic Planning

BDDA* [16] can be casted as a variant of the BDD-based implementation of Dijkstra's algorithms with consistent heuristics. BDDA* was invented by [16] in the context of solving the single-agent challenges. ADDA* developed by [20] is an alternative implementation of BDDA* with ADDs, while SetA* by [28] introduces branching partitioning. Symbolic branch-and-bound search has been proposed by [27]. In an experimental study [35] 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 [14].

In the extension of BDDA* to weighted graphs shown in Algorithm 5, 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. A joint distribution constructed 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 6 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 6 *(Optimality and Complexity of Algorithm 5) For transition weights $w \in \{1, \ldots, C\}$, the symbolic version of algorithm A* on a two-level bucket*

Algorithm 5 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 $\mathit{Trans}_a(x, x')$, shortest-path locality L
Output: Optimal solution path

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

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 [38], a recursive reconstruction based based on relay layers can be preferable, and as said, relay layers can be avoided by using disk space.

9 Zero-Cost Actions

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.

Algorithm 6 Evaluate

Input: State set *States*(x), value h
Global: Disjoint pattern databases $\mathit{PDB}_1, \ldots, \mathit{PDB}_k$
Output: *Result*(x) for subset of *States*(x) with $h = \mathit{PDB}_1(x) + \ldots + \mathit{PDB}_k(x)$

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*$(x) \wedge \mathit{PDB}_1[i_1](x) \wedge \ldots \wedge \mathit{PDB}_k[i_k](x)$)
return *Result*(x)

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 to be 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.

10 Results

We ran experiments on a Linux 64-bit AMD computer with 2.4 GHz. 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. 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 $L = 10$.

There are several step-optimal planners, e.g HSP by Geffner and Haslum, MIPS-XXL by Edelkamp, Petrify by Hickmott et al., UMOP by Jensen, and BFHSP by Zhou and Hansen, etc. Unfortunately, none of the planners includes monotone action costs[1]. 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 planners CPT by Vidal & Geffner and TP4 by Haslum. Both planners compute 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[2]. We imposed a time limit of 1 hour and a memory limit of 2 GB[3].

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

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

Table 2 shows the results we obtained in the IPC-3 domain *ZenoTravel*. The symbolic implementation of Dijkstra's algorithms solved the first 7 problems cost-optimal. 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 (including pattern database construction) in about 20min. The comparison with CPT shows that the sequential duration raises from factor 1 to more than 2 compared the parallel duration. Finding these plans took more time, and larger problems could not be solved within the time or space

[1]We are aware of two current implementation efforts for sequential optimal planners, one by Menkes van den Briel and one by Malte Helmert.

[2]For relaxed plans it is known that finding the sequential optimal plan is NP-hard, while finding the parallel optimal plan is polynomial [24].

[3]The reference computer for CPT/TP4 is has a 2.8GHz CPU equipped with 1 GB RAM [37].

limits (CPT solved instance 10 and 11, too). In such cases we provide lower bounds.

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

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

Table 3 shows the results we obtained in the IPC-3 domain *DriverLog*. Here we experimented with Dijkstra's algorithm only. For this case the 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 plan costs for the first 7 problems were 303 (1.01s), 407 (7.63s), 215 (1.93s), 503 (59.71s), 182 (18.44s), 336 (129s), and 380 (1,715s).

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

Table 4: Results in *Depots* (IPC-3).

Table 4 shows the results we obtained in the IPC-3 domain *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[4].

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

Table 5: Results in *Sattelite*, *SimpleTime* (IPC-3).

Table 5 shows the results for the IPC-3 domain *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.

Table 6 shows the results we obtained in the IPC-5 domain *Storage*. The planner we compare with is CPT2[5]. 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.

11 Generalization of the Cost Model

In a PDDL problem domain for cost model 3, costs are provided by a *state formula*, where *state-formula* is an expression over the action and the state's fluents

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

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

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

Table 6: Results in *Storage*, *Time* (IPC-5).

(corresponding to PDDL 2, Level 2 planning) or indicator variables (assuming 1 for *true* and 0 for *false*, corresponding to PDDL 3 propositional planning).

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 adaption of Dijkstra's algorithm to this scenario is shown in Algorithm 7. In order to attach new f-values to this set we compute $f = f' + c$ by using BDD arithmetics on finite domains.

The construction of shortest path symbolic pattern databases and the application of BDDA* for such a setting are extended analogously.

12 Conclusion and Discussion

In this paper we have shown how to extend BDD-based planning to compute *shortest* plans. Symbolic weighted pattern database were constructed based on a bucket implementations of Dijkstra's single-source shortest-paths algorithm. The distributions of different pattern databases are merged on-the-fly and deliver heuristic estimates for finding solutions to the concrete problems with symbolic A* search. The shortest-path locality limits the duplicate detection scope.

Is the approach scalable? If all costs are multiplied with a factor, then the only change is that intermediate empty buckets have to be scanned. On the other hand,

Algorithm 7 Shortest-Path-BDD-Dijkstra.

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 $\mathit{Trans}_a(c, x, x')$, shortest-path locality L

Output: Optimal solution path or $\bot$ if no such plan exists

$\mathit{Open}(f, x) \leftarrow\ (f = 0) \wedge \mathcal{I}(x)$
loop
  $f' \leftarrow \min\{f \mid f \wedge \mathit{Open}(f, x) \neq \bot\}$
  $\mathit{Min}(x) \leftarrow \exists f(\mathit{Open}(f, x)\ \wedge\ f = f')$
  if $(\mathit{Min}(x) \wedge\ \mathcal{G}(x) = \bot)$
    return $\mathit{Construct}(\mathit{Min}(x) \wedge\ \mathrm{G}(x))$
  $\mathit{Rest}(f, x) \leftarrow \mathit{Open} \wedge \neg\, \mathit{Min}$
  $\mathit{Succ}(f, x') \leftarrow \exists x, f', c$
    $(\mathit{Min}(x') \wedge \bigvee_{a \in \mathcal{A}} \mathit{Trans}_a(c, x, x') \wedge f' + c = f)$
  $\mathit{Succ}(f, x) \leftarrow \exists x(\mathit{Succ}(f, x') \wedge x = x')$
  $\mathit{Open}(f, x) \leftarrow \mathit{Rest}(f, x) \vee \mathit{Succ}(f, x)$

finding the next non-trivial bucket is fast compared to computing images of non-empty ones. Only on very sparse graph or very large action costs the difference may becomes dominant. For these cases, more advance bucket implementations apply.

If the number of images increases, this does not necessarily mean that the symbolic algorithm is less efficient. On the other hand, if BDDs represent very small sets of states, then the gain of the symbolic representation becomes obsolete. Partitioning along the action and action is a fair compromise between the number and hardness of image computations.

The experimental comparison with CPT has not produced a clear-cut winner. Nonetheless, the weighted BDD planning approach indicates better performance, for the cost model we have chosen. The core difference of BDD-based compared to SAT- and CSP-based planning is that BDDs images are computed independently wrt. to previous levels, while the complexity of the other two approaches raises with the search depth. This induces that SAT-based planners tend to loose dominance in problems that require a large search depth. So far, we gave evidence that the generic approach of symbolic shortest-path search has the potential to compete in the next international planning competition, and to influence the design of planners in the near future.

References

[1] 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.

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

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

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

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

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

[7] J. C. Culberson. Sokoban is PSPACE-complete. In *FUN*, pages 65–76, 1998.

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

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

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

[11] S. Edelkamp. Planning with pattern databases. In *ECP*, pages 13–24, 2001.

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

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

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

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

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

[17] R. Fikes and N. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

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

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

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

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

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

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

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

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

[26] R. Jensen. *Efficient BDD-based planning for non-deterministic, fault-tolerant, and adversarial domains*. PhD thesis, Carnegie-Mellon University, 2003.

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

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

[29] A. Junghanns. *Pushing the Limits: New Developments in Single-Agent Search*. PhD thesis, University of Alberta, 1999.

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

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

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

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

[34] K. Qian. *Formal Verification using heursitic search and abstraction techniques*. PhD thesis, University of New South Wales, 2006.

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

[36] D. Ratner and M. K. Warmuth. The $(n^2 - 1)$-puzzle and related relocation problems. *Journal of Symbolic Computation*, 10(2):111–137, 1990.

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

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