Partial Symbolic Pattern Databases for Optimal Sequential Planning*

Stefan Edelkamp and Peter Kissmann

Faculty of Computer Science
TU Dortmund, Germany

Abstract. This paper investigates symbolic heuristic search with BDDs for solving domain-independent action planning problems cost-optimally. By distributing the impact of operators that take part in several abstractions, multiple partial symbolic pattern databases are added for an admissible heuristic, even if the selected patterns are not disjoint. As a trade-off between symbolic bidirectional and heuristic search with BDDs on rather small pattern sets, partial symbolic pattern databases are applied.

1 Introduction

Optimal sequential planning for minimizing the sum of action costs is a natural search concept for many applications. For the space-efficient construction of planning heuristics, symbolic pattern databases [7] (a BDD-based compressed representation of explicit-state pattern databases) are promising. Their space requirements are often much smaller than explicit-state pattern databases. They correlate with the size of the pattern state space, but the request of a parameter that respects the RAM limit for all benchmark domains and problem instances makes it harder to apply symbolic pattern databases.

This paper combines symbolic pattern database heuristic search planning with perimeter search. Perimeters and pattern databases are very similar in their approach to speeding up search. Both techniques use backward search to construct a memory-based heuristic. The memory resources determine the abstraction level for pattern databases as they must completely fit into memory. Perimeters are built without any abstraction; the perimeter stops being expanded when a memory limit is reached. The good results for the hybrid of the two techniques align with recent findings in domain-dependent single-agent search: partial pattern databases [2] and perimeter pattern databases [11]. Besides general applicability, our approach covers different cost models; it applies to step-optimal propositional planning and planning with additive action costs. Moreover, the approach can operate with any pattern selection strategy. Instead of predicting the abstract search space, we set a time limit to terminate the pattern database construction.

The paper is structured as follows. First, we recall symbolic shortest-path and heuristic search. Then, we introduce planning pattern databases and address the addition of non-disjoint symbolic pattern database heuristics by an automated distribution of operator costs. Partial symbolic pattern databases, their symbolic construction, and their inclusion in domain-dependent heuristic search are discussed next. Finally, we provide some experiments and give concluding remarks.

* Thanks to DFG for support in the projects ED 74/3 and 74/2.

2 Symbolic Shortest Path Search

We assume a given planning problem $\mathcal{P} = (\mathcal{S}, \mathcal{A}, \mathcal{I}, \mathcal{G}, \mathcal{C})$ of finding a sequence of actions $a_1, \ldots, a_k \in \mathcal{A}$ from $\mathcal{I}$ to $\mathcal{G}$ with minimal $\sum_{i=1}^{k} \mathcal{C}(a_i)$. For symbolic search with BDDs we additionally assume a binary 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 $\bigvee_{a \in \mathcal{A}} (\exists x \, (\mathit{Trans}_a(x, x') \wedge S(x)))$.

For positive action costs, the first plan reported by the single-source shortest-paths search algorithm of Dijkstra [5] is optimal. For implicit graphs, we need two data structures, one to access nodes in the search frontier and one to detect duplicates.

Algorithm 1 Symbolic-Shortest-Path.

Input: Problem $\mathcal{P} = (\mathcal{S}, \mathcal{A}, \mathcal{I}, \mathcal{G}, \mathcal{C})$ 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 $f = 0, \ldots, f_{\max}$
  for all $l = 1, 2, \ldots, g$ **do** $\mathit{Open}[g] \leftarrow \mathit{Open}[g] \setminus \mathit{Open}[g - l]$
  $\mathit{Min}(x) \leftarrow \mathit{Open}[f](x)$
  if $(\mathit{Min}(x) \wedge \mathcal{G}(x) \neq \mathit{false})$ **return** $\mathit{Construct}(\mathit{Min}(x) \wedge \mathcal{G}(x))$
  for all $i = 1, \ldots, C$ **do**
    $\mathit{Succ}_i(x') \leftarrow \bigvee_{a \in \mathcal{A}, \mathcal{C}(a) = i} (\exists x (\mathit{Min}(x) \wedge \mathit{Trans}_a(x, x')))$
    $\mathit{Succ}_i(x) \leftarrow \exists x' (\mathit{Succ}_i(x') \wedge x = x')$
    $\mathit{Open}[f + i](x) \leftarrow \mathit{Open}[f + i](x) \vee \mathit{Succ}_i(x)$

If we assume that the largest action cost is bounded by some constant C, a symbolic shortest path search procedure is implemented in Algorithm 1. The algorithm works as follows. The BDD $\mathit{Open}[0]$ 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 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 all successor states that can be reached with cost i. In order to attach new f-values to this set, we add the result in bucket $f + i$.

An advanced implementation for the priority queue is a one-level bucket [4], namely an array of size $C + 1$, each of which entries stores a BDD for the elements. For large values of C, multi-layered bucket and radix-heap data structures are appropriate [1].

Let us briefly consider possible implementations for *Construct*. If all previous layers remain in main memory, sequential solution reconstruction is sufficient. If layers are eliminated as in frontier search [19] or breadth-first heuristic search [21], 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 [8].

The above algorithm traverses the search tree expansion of the problem graph. It is sound as it finds an optimal solution if one exists. It is complete and necessarily terminates if there is no solution. By employing backward search, therefore, it can be used to generate symbolic pattern databases. For the search, we apply a symbolic variant of A* [9, 12, 17, 20], which, for consistent heuristics, can be deemed as a variant of the symbolic implementation of Dijkstra's algorithm.

3 Additive Symbolic Pattern Databases

Applying abstractions simplifies a problem: exact distances in these relaxed problems serve as lower bound estimates. Moreover, the combination of heuristics for different abstractions often leads to a better search guidance. Pattern databases [3] completely evaluate an abstract search space $\mathcal{P}' = (\mathcal{S}', \mathcal{A}', \mathcal{I}', \mathcal{G}', \mathcal{C}')$ prior to the concrete, base-level search in $\mathcal{P}$ in form of a dictionary containing the abstract goal distance from each state in $\mathcal{S}'$. Abstractions have to be cost-preserving, which implies that each concrete path maps to an abstract one with smaller or equal cost. The construction process is backwards and the size of a pattern database is the number of states it contains.

One natural concept for planning abstraction domains is to select a pattern variable support set $\mathit{varset}(R) \subseteq \mathcal{V}$ and project each planning state and each action to $\mathit{varset}(R)$. In domain-dependent planning, the abstraction functions are selected by the user. For domain-independent planning, the system has to infer the abstractions automatically [13, 16, 14].

Symbolic pattern databases [7] are pattern databases that are constructed symbolically for later use in symbolic or explicit-state heuristic search. They exploit that the representation of the actions is provided in form of a relation, which allows to perform backward search by computing the predecessors (the *PreImage*) of a given state set efficiently. Each state set with a distinct goal distance is represented as a BDD. Different to the posterior compression of the state set [10], the construction itself works on a compressed representation, allowing the generation of much larger databases.

Additive symbolic pattern databases allow to add abstract goal distances of different abstractions without affecting the admissibility of the heuristic estimate, i.e., the sum of the abstract goal distances is still a lower bound for the original space. Cost-preserving abstract problem tasks $\mathcal{P}_1, \ldots, \mathcal{P}_k$ wrt. state space homomorphisms $\phi_1, \ldots, \phi_k$ of a planning task $\mathcal{P}$ are trivially additive, if no original action contributes to two different abstractions. For disjoint patterns in single-agent challenges like the $(n^2 - 1)$-Puzzle, this assumption is immediate, as only one tile can move at a time. For general domains, however, the restriction limits pattern database design. Therefore, we introduce the weighted combination of different abstract tasks as follows. First, we determine the number of times an original action $a \in \mathcal{A}$ is valid in the abstractions.

We define $\mathit{valid}(a) = |\{i \in \{1,\ldots,k\} \mid \phi_i(a) \neq \mathit{noop}\}|$, and $t = \mathit{lcm}_{a\in\mathcal{A}}\mathit{valid}(a)$, where *lcm* denotes the least common multiple, and *noop* a void action. Then, we scale the original problem $\mathcal{P}$ to $\mathcal{P}^*$, by setting $\mathcal{C}^*(a) = t \cdot \mathcal{C}(a)$. All plans π now have costs $\mathcal{C}^*(\pi) = \sum_{a\in\pi} t{\cdot}Cost(a) = t{\cdot}\mathcal{C}(\pi)$, such that an optimal plan in $\mathcal{P}$ induces an optimal plan in $\mathcal{P}^*$ and vice versa. Next, we set all action costs $\mathcal{C}_i^*(a)$ in the abstract problem $\mathcal{P}_i^*$ to $\mathcal{C}_i^*(a) = t \cdot \mathcal{C}(a)/\mathit{valid}(a)$, if $\phi_i(a) \neq \mathit{noop}$, and to zero, otherwise. As t is the least common multiple, the cost values in the abstract domains remain integral.

The following result shows that the sum of the heuristic values according to the abstractions $\mathcal{P}_i$, $i \in \{1,\ldots,k\}$, is a lower bound.

Theorem 1. *The weighted combination of different abstract tasks $\mathcal{P}_1,\ldots,\mathcal{P}_k$ of $\mathcal{P}$ wrt. cost-preserving state space homomorphims $\phi_1,\ldots,\phi_k$ always results in an admissible heuristic for the scaled problem $\mathcal{P}^*$.*

Proof. Let $\pi = (a_1,\ldots,a_l)$ be any plan solving $\mathcal{P}^*$ and $\pi_i = (\phi_i(a_1),\ldots,\phi_i(a_l))$ be the corresponding solution in abstraction i, $i \in \{1,\ldots,k\}$. Some $\phi_i(a_j)$ are trivial and correspond to cost 0. The sum of costs of all π_i is equal to

$$\begin{aligned}
\sum_{i=1}^{k}\mathcal{C}_i^*(\pi_i) = \sum_{i=1}^{k}\sum_{j=1}^{l}\mathcal{C}_i^*(a_j) &= \sum_{i=1}^{k}\sum_{j=1}^{l} t\cdot\mathcal{C}(a_j)[\phi_i(a)\neq\mathit{noop}]/\mathit{valid}(a_j)\\
&= \sum_{j=1}^{l}\sum_{i=1}^{k} t\cdot\mathcal{C}(a_j)[\phi_i(a)\neq\mathit{noop}]/\mathit{valid}(a_j)\\
&\leq \sum_{j=1}^{l}(t\cdot\mathcal{C}(a_j)/\mathit{valid}(a_j))\sum_{i=1}^{k}[\phi_i(a_j)\neq\mathit{noop}]\\
&= \sum_{j=1}^{l}(t\cdot\mathcal{C}(a_j)/\mathit{valid}(a_j))\mathit{valid}(a_j)\\
&= \sum_{j=1}^{l} t\cdot\mathcal{C}(a_j) = \sum_{a\in\pi} t\cdot\mathcal{C}(a) = \mathcal{C}^*(\pi).
\end{aligned}$$

As π is chosen arbitrarily, we conclude that $\min_\pi \sum_{i=1}^{k}\mathcal{C}_i^*(\pi_i) \leq \min_\pi \mathcal{C}^*(\pi)$.

Additive symbolic databases for non-disjoint patterns are novel, but similar techniques for explicit search have been introduced (though not empirically evaluated) [18].

4 Partial Symbolic Pattern Databases

Perimeter search [6] tries to reap the benefits of front-to-front-evaluations in bidirectional search, while avoiding the computational efforts involved in re-targeting the heuristics towards a continuously changing search frontier. It conducts a cost-bounded best-first search starting from the goal nodes; the nodes on the final search frontier, called the perimeter, are stored in a dictionary. Then, a forward search, starting from $\mathcal{I}$,

employs front-to-front evaluation with respect to these nodes. Alternatively, in front-to-goal perimeter search all nodes outside the perimeter are assigned to the maximum of the minimum of the goal distances of all expanded nodes in the perimeter and an additionally available heuristic estimate. Although larger perimeters provide better heuristics, they take increasingly longer to compute. The memory requirements for storing the perimeter are considerable and, more crucially, the effort for multiple heuristic computations can become large.

In essence, a partial pattern database is a perimeter in the abstract space (with the interior stored). Any node in the original space has a heuristic estimate to the goal: if it is in the partial pattern database, the recorded goal distance value is returned; if not, the radius d of the perimeter is returned. This heuristic is both admissible and consistent. Building a partial pattern database starts from the abstract goal and stores heuristic values. When a memory or time limit is reached it terminates and the heuristic values are used for the forward search. The value d is the minimum cost of all abstract nodes outside the partial pattern database. On one extreme, a partial pattern database with no abstraction reverts to exactly a perimeter. On the other extreme, a partial pattern database with a coarse-grained abstraction will cover the entire space, and performs exactly like an ordinary pattern database. Bidirectional BFS is an interleaved partial pattern database search without abstraction.

An ordinary pattern database represents the entire space; every state visited during the forward search has a corresponding heuristic value in the database. The pattern database abstraction level is determined by the amount of available memory. Fine-grained abstraction levels are not possible, because the memory requirements increase exponentially with finer abstractions. Explicit-state partial pattern databases, however, generally do not cover the entire space.

A partial symbolic pattern database is the outcome of a partial symbolic backward shortest paths exploration in abstract space. It consists of a set of abstract states $\mathcal{S}'_{<d}$ and their goal distance, where $\mathcal{S}'_{<d}$ contains all states in $\mathcal{S}'$ with cost-to-goal less than d, and where d is a lower bound on the cost of any abstract state not contained in $\mathcal{S}'_{<d}$. The state sets are kept as BDDs $H[i]$ representing $\mathcal{S}'_{<i+1} \setminus \mathcal{S}'_{<i}$, $i \in \{0, ..., d-1\}$. The BDD $H[d]$ represents the state set $\mathcal{S}' \setminus \mathcal{S}'_{<d}$, such that a partial pattern database partitions the abstract search space. If $C_{\max}$ is the cost of the most expensive action, the construction works as shown in Algorithm 2.[1]

Multiple pattern databases can be *merged* by either taking the maximum, the sum, or the weighted combination of individual pattern database entries. Additionally we apply iterative abstraction to refine the quality of a database. If pattern database construction is partial because of the time cut-off, an abstraction ϕ can be re-abstracted to ϕ' (e.g., by dropping another variable). The idea is illustrated in Fig. 1. Algorithm 3 shows how remaining states outside the previous perimeter are classified wrt. ϕ'. The process is iterated until all states are classified.

[1] For large costs, pattern databases can become sparse, so that in practice we compress the BDD vector $H[0..\max]$ to a list containing $(i, H[i])$ for every i with $H[i] \neq \mathit{false}$.

Algorithm 2 Construct-Partial-Symbolic-PDB

Input: Abstract planning problem $\mathcal{P}' = (\mathcal{S}', \mathcal{A}', \mathcal{I}', \mathcal{G}', \mathcal{C}')$
Output: Partial or full symbolic pattern database

Reached $\leftarrow H[0] \leftarrow \mathcal{G}'$
for all $d = 0, 1, 2, \ldots$ **do**
  $H[d] \leftarrow H[d] \wedge \neg$*Reached*
  if ($H[d..d + C_{\max} - 1] \equiv$ *false*) **then return** (*full*, $H[0..d-1]$)
  if (*time-out*) **then return** (*partial*, ($H[0..d]$, $\neg$*Reached*))
  Reached $\leftarrow$ *Reached* $\vee H[d]$
  for all $i = 1, \ldots, C_{\max}$ **do**
    Succ$_i \leftarrow \bigvee_{a \in \mathcal{A}', C'(a)=i}$ *PreImage*$(H[d]$, *Trans*$'_a)$
    $H[d+i] \leftarrow H[d+i] \vee$ *Succ*$_i$

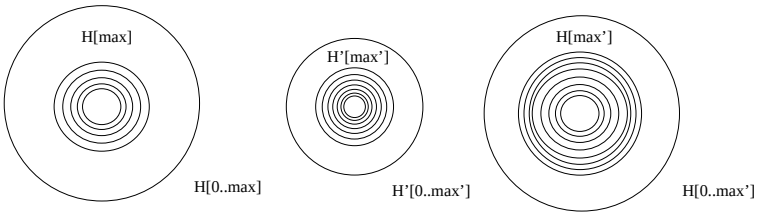

Fig. 1. Partial pattern database refinement: left 2 pattern databases are merged into the right one.

In the extension to weighted graphs, all successors of the set of states with minimum f-value are determined wrt. the current cost value g and action cost value i. To determine their h-values, the merged symbolic pattern database is scanned.[2]

5 Experiments

Table 1 compares symbolic search with partial symbolic pattern databases (PSPDB, construction time included) with symbolic bidirectional breadth-first search (SBBFS), PDB [14] and LFPA [15] on various competition problems on equivalent computational resources. For automated pattern selection (at least compared to [14]) we chose a rather simple policy. Abstractions are variable projections, combined via greedy bin-packing. PSPDB shows benefits to PDB, especially considering that automated pattern selection is less elaborated. Wrt. LFPA, the advantage is less impressive.

For TPP-9 (from IPC-5), a problem that according to [15] cannot be solved by any other heuristic search planner so far. It also can not be solved with bidirectional search. Two partial pattern databases were constructed. As some actions remain applicable in both abstractions, their costs are 1. The costs of all other actions in the abstract space

[2] In our implementation, we avoid merging prior to the search and merge the results of the lookups. However, we observed that the advantages of such dynamic lookups are small.

Algorithm 3 Refinement

Input: Planning problem $\mathcal{P} = (\mathcal{S}, \mathcal{A}, \mathcal{I}, \mathcal{G})$, multiple pattern database abstraction set Φ
Output: Symbolic pattern database set

for all $\phi \in \Phi$ **do**
  $(r, H[0..\max]) \leftarrow$ *Construct-Partial-Symbolic-PDB*(P_ϕ)
  while $(r =$ *partial*$)$ **do**
    $\phi' \leftarrow$ *Abstract*(ϕ); *Rest* $\leftarrow$ *false*
    $(r, H'[0..\max']) \leftarrow$ *Construct-Partial-Symbolic-PDB*$(P_{\phi'})$
    for all $i = 1, \ldots, \max - 1$ **do** *Rest* $\leftarrow (H'[i] \wedge H[\max]) \vee$ *Rest*
    for all $i = \max + 1, \ldots, \max'$ **do** $H[i] \leftarrow H'[i] \wedge H[\max]$
    $H[\max] \leftarrow H[\max] \wedge (H'[\max] \vee$ *Rest*$)$
    $\max \leftarrow \max'$
  $\mathcal{H} \leftarrow$ *Merge*$(\mathcal{H}, H[0..\max])$
return $\mathcal{H}$

Problem	L*	PSPDB	SBBFS	PDB	LFPA
log-10-0	45	137s	577s	497s	117s
log-10-1	42	174s	546s	405s	129s
log-11-0	48	68s	577s	377s	129s
log-11-1	60	65s	-	-	284s
log-12-0	42	47s	473s	545s	185s
log-12-1	68	861s	-	-	221s
sat-4	17	5s	2s	6s	11s
sat-5	15	41s	37s	110s	47s
sat-6	20	38s	27s	21s	634s
sat-7	21	567s	-	-	-
psr-48	37	10s	128s	787s	36s
psr-49	47	57s	-	-	67s

Table 1. Comparison between optimal planning approaches (on a 2.6GHz CPU, 2 GB RAM).

and the costs of all actions in the original state space are multiplied with two. The exploration took 354 iterations to find a plan of 48 steps in about 2h. Between 32 and 64 million BDDs were used for the entire exploration, corresponding to about 1.5 GB.

6 Conclusion

We extended heuristic search planning to work with partial symbolic pattern databases, a recent compromise between bidirectional breadth-first search and a low number of variables in the pattern for a full database. Thereby, we obtained promising results for a selection of planning benchmarks. As a matter of fact, the results are highly selective, and given that both LFPA and PDB take different parameter settings in different domains, we are excited to directly compare the approaches in a competition.

Due to the scaling of action costs to preserve admissibility of the heuristic, we employed a bucket implementation of Dijkstra's single-source shortest paths algorithm for the cost-limited construction of each database. One possible future research avenue is to construct partial pattern databases on-the-fly; similar to bidirectional BFS, where the search continues in the direction that is likely to be the cheapest.

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. K. Anderson, R. Holte, and J. Schaeffer. Partial pattern databases. In *SARA*, pages 20–34, 2007.
3. J. C. Culberson and J. Schaeffer. Pattern databases. *Computational Intelligence*, 14(4):318–334, 1998.
4. R. B. Dial. Shortest-path forest with topological ordering. *Communications of the ACM*, 12(11):632–633, 1969.
5. E. W. Dijkstra. A note on two problems in connection with graphs. *Numerische Mathematik*, 1:269–271, 1959.
6. J. F. Dillenburg and P. C. Nelson. Perimeter search. *Artificial Intelligence*, 65(1):165–178, 1994.
7. S. Edelkamp. Symbolic pattern databases in heuristic search planning. In *AIPS*, pages 274–293, 2002.
8. S. Edelkamp. External symbolic heuristic search with pattern databases. In *ICAPS*, pages 51–60, 2005.
9. S. Edelkamp and F. Reffel. OBDDs in heuristic search. In *KI*, pages 81–92, 1998.
10. A. Felner, Richard E. Korf, R. Meshulam, and R. C. Holte. Compressed pattern databases. *Journal of Artificial Intelligence Research*, 30:213–247, 2006.
11. A. Felner and Nir Ofek. Combining perimeter search and pattern database abstractions. In *SARA*, pages 155–168, 2007.
12. E. A. Hansen, R. Zhou, and Z. Feng. Symbolic heuristic search using decision diagrams. In *SARA*, pages 83–98, 2002.
13. P. Haslum, B. Bonet, and H. Geffner. New admissible heuristics for domain-independent planning. In *AAAI*, pages 1163–1168, 2005.
14. P. Haslum, A. Botea, M. Helmert, B. Bonet, and S. Koenig. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *AAAI*, pages 1007–1012, 2007.
15. M. Helmert, P. Haslum, and J. Hoffmann. Flexible abstraction heuristics for optimal sequential planning. In *ICAPS*, pages 176–183, 2007.
16. J. Hoffmann, A. Sabharwal, and C. Domshlak. Friends or foes? An AI planning perspective on abstraction and search. In *ICAPS*, pages 294–303, 2006.
17. R. M. Jensen, R. E. Bryant, and M. M. Veloso. SetA*: An efficient BDD-based heuristic search algorithm. In *AAAI*, pages 668–673, 2002.
18. M. Katz and C. Domshlak. Combining perimeter search and pattern database abstractions. In *ICAPS-Workshop*, 2007.
19. R. E. Korf, W. Zhang, I. Thayer, and H. Hohwald. Frontier search. *Journal of the ACM*, 52(5):715–748, 2005.
20. K. Qian. *Formal Verification using heursitic search and abstraction techniques*. PhD thesis, University of New South Wales, 2006.
21. R. Zhou and E. Hansen. Breadth-first heuristic search. In *ICAPS*, pages 92–100, 2004.