Solving Non-deterministic Planning Problems with Pattern Database Heuristics

Pascal Bercher[1,⋆] and Robert Mattmüller[2,⋆⋆]

[1] Institut für Künstliche Intelligenz, Universität Ulm, Germany,
pascal.bercher@uni-ulm.de
[2] Institut für Informatik, Albert-Ludwigs-Universität Freiburg, Germany,
mattmuel@informatik.uni-freiburg.de

Abstract. Non-determinism arises naturally in many real-world applications of action planning. Strong plans for this type of problems can be found using AO* search guided by an appropriate heuristic function. Most domain-independent heuristics considered in this context so far are based on the idea of ignoring delete lists and do not properly take the non-determinism into account. Therefore, we investigate the applicability of pattern database (PDB) heuristics to non-deterministic planning. PDB heuristics have emerged as rather informative in a deterministic context. Our empirical results suggest that PDB heuristics can also perform reasonably well in non-deterministic planning. Additionally, we present a generalization of the pattern additivity criterion known from classical planning to the non-deterministic setting.

Key words: Heuristic search, non-deterministic planning, PDB heuristics

1 Introduction

Non-deterministic planning problems arise naturally as soon as the agent seeking a goal is confronted with an environment that may have an unpredictable influence on action outcomes. Specifically, in this work, we are concerned with finding strong plans [1] for non-deterministic planning tasks in fully observable and static environments. In general, approaches to tackle non-determinism include planning as model checking [1, 2], QBF-based approaches, and heuristically guided explicit state techniques [3–6]. Here, we follow the latter approach and compute strong plans by explicitly constructing the relevant portion of the AND/OR graph encoding the dynamics of the world, and returning the plan

⋆ This work was partly supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center "Eine Companion-Technologie für kognitive technische Systeme" (SFB/TRR 62).

⋆⋆ This work was partly supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS). See www.avacs.org for more information.

corresponding to a solution subgraph. The construction of the graph follows the AO* algorithm [7] and is guided by a pattern database (PDB) heuristic that estimates the cost of the solution subgraph rooted at a given node [8]. Given an informative node evaluation function, more promising parts of the graph are likely to be expanded before the less promising ones, resulting in a relatively low number of node expansions before a solution has been found.

As has been shown in earlier work [3–5], using the AO* algorithm in conjunction with an informative heuristic can be an efficient way to find plans of high quality. Since the evaluation functions employed so far rarely take non-determinism into account properly, in this work we investigate the use of PDB heuristics, since the abstractions underlying the PDBs can be built in a way that preserves the non-determinism of the original problem.

PDB heuristics have been studied before extensively, both for deterministic and non-deterministic problems, and both with problem-specific patterns and with problem-independent pattern selection techniques. In this work, we describe how to use given PDBs in a generic non-deterministic planner. Problem-independent ways to come up with patterns are left for future work. Our main practical contribution is the development of a non-deterministic planner for static and fully-observable problems based on AO* search guided by PDB heuristics that is domain-independent except for the lack of an automated pattern selection. On the theoretical side, we present a generalization of the additivity criterion for sets of patterns known from classical planning [9] to non-deterministic planning.

2 Non-deterministic Planning

We consider non-deterministic planning problems under full observability. In contrast to classical planning, the actions can have several outcomes, only one of which takes effect non-deterministically.

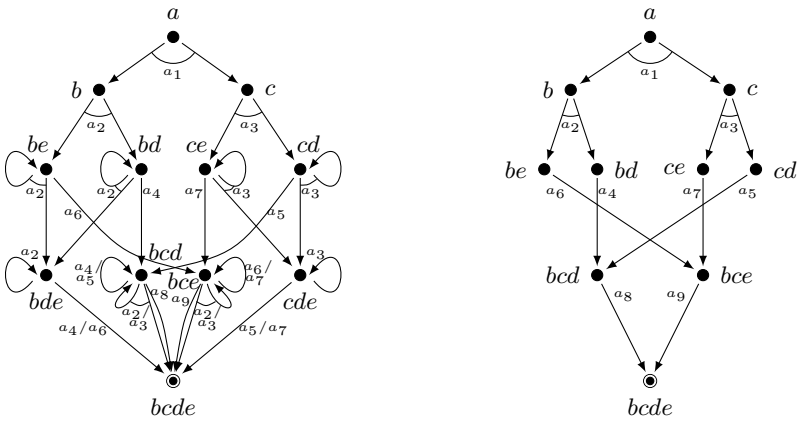
Formally, a *non-deterministic planning problem* $\mathcal{P}$ consists of a finite set *Var* of *state variables*, a finite set A of *actions*, an *initial state* s_0 and a *goal description* $G \subseteq \mathit{Var}$. The set of states $S = 2^{\mathit{Var}}$ is the set of all valuations of the state variables, and a state s is a *goal state* iff $G \subseteq s$. Each action $a \in A$ is a pair consisting of a set of *preconditions* $\mathit{pre}(a) \subseteq \mathit{Var}$ and a set $\mathit{eff}(a)$ of non-deterministic *effects* $\langle \mathit{add}_i, \mathit{del}_i \rangle$, $i = 1, \ldots, n$, each consisting of *add and delete lists* $\mathit{add}_i, \mathit{del}_i \subseteq \mathit{Var}$. We call the set of all variables mentioned in the add and delete lists of an action a its *effect variables* and denote them as $\mathit{effvar}(a) := \bigcup_{i=1}^{n} (\mathit{add}_i \cup \mathit{del}_i)$. An action a is *applicable* in a state s if $\mathit{pre}(a) \subseteq s$ and its application leads to the *successor states* $\mathit{app}(s, a) := \{\ (s \setminus \mathit{del}) \cup \mathit{add} \mid \langle \mathit{add}, \mathit{del} \rangle \in \mathit{eff}(a)\ \}$.

We want to find a strategy that is guaranteed to transform the initial state into an arbitrary goal state within a finite number of steps no matter what the outcomes of the non-deterministic actions are. Such a strategy is a partial function mapping states to applicable actions. Cimatti et al. [1] call such a strategy a *strong plan*. We formalize strong plans by means of solution graphs as follows. A planning problem $\mathcal{P}$ induces an *AND/OR graph* $\mathcal{G} = \langle V, C \rangle$, where

$V = S$ is the set of *nodes* and C is the set of *connectors*. For each non-goal node $v \in V$ and $a \in A$ with $pre(a) \subseteq v$, there is a connector $c = \langle v, app(v, a)\rangle \in C$. We call $pred(c) := v$ the *predecessor* of c and $succ(c) := app(v, a)$ the *successors* of c. The AND/OR graph of $\mathcal{P}$ is defined as the connected component of $\mathcal{G}$ which contains s_0. A *solution graph* $\mathcal{G} = \langle V, C\rangle$ is a connected acyclic subgraph of the AND/OR graph of $\mathcal{P}$ which contains s_0, where for all $v \in V$, either v is a goal state or there is exactly one $c \in C$ such that $pred(c) = v$, and where for all $c \in C$, $pred(c) \in V$ and $succ(c) \subseteq V$.

We use solution graphs to define the cost value of states. Since we prefer strong plans with a low worst-case number of action applications along the way to a goal state, we define the cost of a state $s \in S$ as $cost^*(s) := \min_{\mathcal{G}} depth(\mathcal{G})$, where $\mathcal{G}$ ranges over all solution graphs rooted at s.

Example 1. As an example, consider the following problem with variables a, b, c, d, and e, actions $a_1, \ldots, a_9$, $s_0 = \{a\}$, and $G = \{b, c, d, e\}$, where $a_1 = \langle a, b \wedge \neg a \,|\, c \wedge \neg a\rangle$, $a_2 = \langle b, e \,|\, d\rangle$, $a_3 = \langle c, e \,|\, d\rangle$, $a_4 = \langle b \wedge d, c\rangle$, $a_5 = \langle c \wedge d, b\rangle$, $a_6 = \langle b \wedge e, c\rangle$, $a_7 = \langle c \wedge e, b\rangle$, $a_8 = \langle b \wedge c \wedge d, e\rangle$, and $a_9 = \langle b \wedge c \wedge e, d\rangle$. Preconditions and effects are written in logical notation and different non-deterministic effects are separated by vertical bars. Fig. 1a shows the AND/OR graph of $\mathcal{P}$. There exists only one subgraph encoding a strong plan, shown in Fig. 1b.



(a) The AND/OR graph of $\mathcal{P}$. (b) Solution graph for $\mathcal{P}$.

Fig. 1: The AND/OR graph and a solution graph for $\mathcal{P}$.

3 Search Algorithm

We use AO* [7] graph search to traverse the AND/OR graph induced by a planning problem. AO* starts with the empty graph and successively expands it until a solution graph has been found or the AND/OR graph of $\mathcal{P}$ has been completely generated. The performance of AO* heavily relies on the quality of the heuristic function applied to the fringe nodes.

4 Pattern Database Heuristics

Pattern database heuristics are a special case of abstraction heuristics. The basic idea is to obtain heuristic values by optimally solving abstractions of the planning problem and using abstract costs as heuristic values. The abstractions are precomputed before the actual search is performed. During the search, no costly calculations are necessary. The heuristic values are merely retrieved from the pattern database, in which the cost values of the abstract states have been stored during the preprocessing stage. Each abstraction can be regarded as a simplification of the planning problem obtained by restraining it to a given pattern, i.e., a subset of the state variables.

Formally, an abstraction $\mathcal{P}^i$ of $\mathcal{P}$ with respect to a pattern $P_i \subseteq Var$ is the planning problem with variables $Var^i = P_i$ and states $S^i = 2^{Var^i}$, where all conditions and effects are restricted to P_i. More precisely, for $var \subseteq Var$, let $var^i := var \cap P_i$. Then $\mathcal{P}^i$ contains an action a^i for each $a \in A$, where $pre(a^i) = pre(a)^i$, and $\mathit{eff}(a^i)$ contains pairs $\langle add^i, del^i \rangle$ of add and delete lists for all pairs $\langle add, del \rangle$ in $\mathit{eff}(a)$. Finally, $s_0^i = s_0 \cap P_i$ and $G^i = G \cap P_i$. Given a pattern P_i, we define the heuristic function h^i by $h^i(s) := cost_i^*(s) := cost_i^*(s^i)$, where the abstract costs are defined analogously to the concrete costs of a state, i.e., as the depth of a depth-minimizing (abstract) solution graph rooted at s^i.

We calculate the heuristic values in a preprocessing step by complete exhaustive search. Since the size of the abstract state space grows exponentially in the size of the pattern, reasonable patterns should not be too large.

Given a pattern collection P consisting of patterns P_i, $i = 1, \ldots, k$, such that each h^i is admissible, i.e., never overestimates the true cost of a state, we can define the heuristic function $h^P(s) := \max_{P_i \in P} h^i(s)$ without violating admissibility. Since we want to maintain as informative heuristic values as possible, however, maximization is often not sufficient.

We call a pattern collection P consisting of patterns $P_1, \ldots, P_k$ *additive* if $\sum_{i=1}^{k} h^i(s) \leq cost^*(s)$ for all states $s \in S$. Given a set $\mathcal{M}$ of additive pattern collections P, we can define the heuristic function $h^{\mathcal{M}}(s) := \max_{P \in \mathcal{M}} \sum_{P_i \in P} h^i(s)$. While $h^{\mathcal{M}}$ is still admissible, it is in general more informative than any of the heuristics h^P. If admissible heuristics are used in combination with an appropriate search algorithm like LAO* [6], one can guarantee to find an optimal plan (if one exists). Admissible heuristics using a set $\mathcal{M}$ of additive pattern collections have another benefit: If the choice of $\mathcal{M}$ is sufficiently good, the resulting heuristic values can be even more appropriate than those of non-admissible heuristics. In classical planning, Edelkamp [9] provides a general criterion for testing whether a pattern collection is additive. It is easy to see that the analogous criterion also holds for the non-deterministic setting, and in particular, that every single h^i is admissible.

Theorem 1. *A pattern collection P is additive if for all actions $a \in A$ and for all patterns $P_i \in P$, if $P_i \cap \mathit{effvar}(a) \neq \emptyset$, then $P_j \cap \mathit{effvar}(a) = \emptyset$ for all $j \neq i$.*

Proof. The proof is by induction on the true cost $cost^*(s)$ of s. The base case for $cost^*(s) = 0$ is trivial (since in this case, all abstractions of s are abstract

goal states and we only sum up costs of zero for all abstractions). For the inductive case, consider a concrete solution graph $\mathcal{G}$ minimizing $cost^*(s)$. Let c be the root connector of $\mathcal{G}$, a an action inducing c, and s' a successor of s in $\mathcal{G}$, $s' \in succ(c)$, along a cost-maximizing path. In the following, for any concrete state $\hat{s}$ and pattern P_i, $cost_i^*(\hat{s})$ denotes the abstract cost of $\hat{s}^i$, i.e., the depth of a depth-minimizing abstract solution graph rooted at $\hat{s}^i$. Without loss of generality, assume that there is a pattern, say P_1, in P such that $P_1 \cap \mathit{effvar}(a) \neq \emptyset$. Then, by assumption, $P_j \cap \mathit{effvar}(a) = \emptyset$ for $j = 2, \ldots, k$. Now let s'' be a successor of s, $s'' \in succ(c)$, such that $(s'')^1$ maximizes the $cost_1^*$ among the abstract successors of s^1. For all patterns P_i other than P_1, we have $(s'')^i = s^i$. Applying the induction hypothesis to s'', we obtain

$$\sum_{i=1}^{k} cost_i^*(s'') \leq cost^*(s'') \ , \tag{1}$$

whereas by assumption and by the definition of $cost^*$, we get

$$cost^*(s'') + 1 \leq cost^*(s') + 1 = cost^*(s) \ . \tag{2}$$

On the other hand,

$$\sum_{i=1}^{k} cost_i^*(s) \leq cost_1^*(s'') + 1 + \sum_{i=2}^{k} cost_i^*(s'') = \sum_{i=1}^{k} cost_i^*(s'') + 1 \ . \tag{3}$$

Taking all this together, by (3), (1), and (2) we obtain that

$$\sum_{i=1}^{k} cost_i^*(s) \leq \sum_{i=1}^{k} cost_i^*(s'') + 1 \leq cost^*(s'') + 1 \leq cost^*(s') + 1 = cost^*(s) \ .$$

Since the heuristic values $h^i(s)$ are defined as $cost_i^*(s)$, this is the same as $\sum_{i=1}^{k} h^i(s) \leq cost^*(s)$. □

Corollary 1. *The heuristic h^i is admissible for any pattern P_i.* □

Finding an appropriate pattern collection $P_1, \ldots, P_k$ is in itself a challenging problem. Different approaches include clustering variables with strong interaction into one pattern, or to perform local search in the space of pattern collections [10, 11], starting from singleton patterns for all goal variables and extending the collection until an evaluation function estimating the number of node expansions in a search using the current pattern collection reaches a local optimum. In the following, however, we will focus on how to use a given pattern collection during search, not on how to obtain it in the first place.

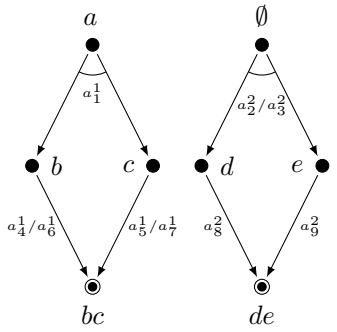
Example 2. Consider the problem from Example 1 and its two abstractions $\mathcal{P}^1$ and $\mathcal{P}^2$ with respect to the patterns $P_1 = \{a, b, c\}$ and $P_2 = \{d, e\}$. Then $s_0^1 = \{a\}$, $s_0^2 = \emptyset$, $G^1 = \{b, c\}$, $G^2 = \{d, e\}$, and the abstract actions are $a_1^1 = \langle a, b \wedge \neg a \,|\, c \wedge \neg a \rangle$, $a_2^1 = \langle b, \top \rangle$, $a_3^1 = \langle c, \top \rangle$, $a_4^1 = \langle b, c \rangle$, $a_5^1 = \langle c, b \rangle$,

$a_6^1 = \langle b, c\rangle$, $a_7^1 = \langle c, b\rangle$, $a_8^1 = \langle b \wedge c, \top\rangle$, and $a_9^1 = \langle b \wedge c, \top\rangle$ for pattern P_1, and $a_1^2 = \langle \top, \top\rangle$, $a_2^2 = \langle \top, e \,|\, d\rangle$, $a_3^2 = \langle \top, e \,|\, d\rangle$, $a_4^2 = \langle d, \top\rangle$, $a_5^2 = \langle d, \top\rangle$, $a_6^2 = \langle e, \top\rangle$, $a_7^2 = \langle e, \top\rangle$, $a_8^2 = \langle d, e\rangle$, and $a_9^2 = \langle e, d\rangle$ for pattern P_2, respectively.

The two graphs to the right show the optimal solution graphs for both abstract problems (left: $\mathcal{P}^1$, right: $\mathcal{P}^2$). We get a more accurate heuristic value by summing over all patterns, since the pattern collection $\{P_1, P_2\}$ is additive. E.g.,

$$\begin{aligned} h(\{a\}) &= cost_1^*(\{a\}^1) + cost_2^*(\{a\}^2) \\ &= cost_1^*(\{a\}) + cost_2^*(\emptyset) \\ &= 2 + 2 = 4 = cost^*(\{a\}). \end{aligned}$$

Hence, in this case, the heuristic even computes the true cost value.



5 Experimental Results

We have encoded several instances of non-deterministic planning problems and solved them with our AO*-based planner and two different heuristics (FF heuristic [12, 5] and PDB heuristics), as well as with Gamer [2], the winner of the fully observable non-deterministic (FOND) track of the uncertainty part of the International Planning Competition 2008 (IPPC'08). For the comparison, we did not use the domains from the IPPC'08, since those problems only allow for strong cyclic plans which our planner cannot find. The results, which were obtained on a machine with an AMD Turion 64 X2 processor (1600 MHz per core), and 1500 MB memory, are summarized in Tables 1 and 2.

In the first domain, Chain of Rooms [13], a number of rooms are sequentially connected by doors. A robot starting in the leftmost room has to visit each room at least once. It can move between rooms if the connecting door is open. Before a door can be passed or opened, the robot has to observe whether the door is open or closed. This observation action is modeled as turning on a light which changes the state of the door from undefined to either open or closed non-deterministically.

In the second domain, Coin Flip, there are n coins, which are initially contained in a bag. The coins have to be tossed exactly once each, in an arbitrary order. Tossing results in the coin showing heads or tails non-deterministically. After a coin has been tossed, it can be turned from heads to tails or vice versa, depending on which side is currently up. The goal is to have all coins on the table showing heads.

For the PDB heuristics and the Chain of Rooms domain, we used one pattern collection for each instance, where each pattern holds all state variables that belong to four neighboring rooms (24 Boolean variables). We omitted one room between each group of four rooms to meet the condition of Theorem 1, thus

dividing a problem with n rooms into $\lceil n/5 \rceil$ subproblems. In the Coin Flip domain, we represented each of n coins by a corresponding pattern (giving us n patterns containing 3 Boolean variables each). Since those subproblems are completely independent, we achieve perfect heuristic values and no unnecessary nodes are expanded.

Table 1: Experimental results from the Chain of Rooms domain. The numbers given in the columns *pre*, *search*, and *sum* are preprocessing, search, and overall times in seconds, *mem* denotes RAM in MB (for PDB plus concrete search space), |PDB| the overall number of PDB entries, *nodes* the number of generated nodes in the AND/OR graph, and |BDD| is the number of BDD nodes. Dashes indicate that the time-out of 30 minutes or the memory bound of 1500 MB was exceeded.

#rooms	AO* Planner, FF			AO* Planner, PDB						Gamer	
	search	mem	nodes	pre	search	sum	mem	|PDB|	nodes	search	|BDD|
20	2	3	236	4	1	5	3	315	274	6	16699
40	15	13	873	7	2	9	17	679	1138	23	130657
60	69	33	1909	16	6	22	46	1043	2602	—	—
80	250	69	3346	26	15	41	100	1407	4666	—	—
100	655	133	5183	41	33	74	190	1771	7330	—	—
120	1497	214	7419	61	73	134	319	2135	10594	—	—
140	—	—	—	91	117	208	495	2499	14458	—	—
160	—	—	—	127	194	321	736	2863	18922	—	—
180	—	—	—	177	314	491	1050	3227	23986	—	—

Table 2: Experimental results from the Coin Flip domain.

#coins	AO* Planner, FF			AO* Planner, PDB						Gamer	
	search	mem	nodes	pre	search	sum	mem	|PDB|	nodes	search	|BDD|
20	—	—	—	2	1	3	4	60	1125	—	—
40	—	—	—	3	4	7	27	120	4645	—	—
60	—	—	—	4	19	23	85	180	10565	—	—
80	—	—	—	7	60	67	180	240	18885	—	—
100	—	—	—	11	217	328	366	300	29605	—	—
120	—	—	—	18	306	324	597	360	42725	—	—
140	—	—	—	23	533	556	905	420	58245	—	—
160	—	—	—	34	908	942	1307	480	76165	—	—

It is worth mentioning that Gamer will always find optimal solutions, whereas our AO*-based planner in general only finds suboptimal solutions.

6 Conclusion and Future Work

We have presented and evaluated a planner for fully-observable non-deterministic planning problems based on AO* search guided by PDB heuristics. Additionally, we have shown a generalization of the pattern additivity criterion known from classical planning, which allows for more informative heuristics while maintaining admissibility. The experimental results show that PDB heuristics are a

promising tool to guide heuristic search algorithms for non-deterministic planning problems.

So far, the patterns are still selected manually. Obviously, a reasonable automated pattern selection technique is necessary to obtain a truly domain-independent planner. In classical planning, local search in the space of pattern collections [10, 11] is often used to automatically select patterns. We believe that this approach will result in good patterns for the non-deterministic case, too. Besides pattern selection, future work includes the adaptation of our implementation to multi-valued state variables and the generalization of the algorithm from strong planning to strong cyclic planning [1].

Acknowledgments. We want to thank Peter Kissmann for providing us with the Gamer planning system and his help in using it.

References

1. Cimatti, A., Pistore, M., Roveri, M., Traverso, P.: Weak, strong, and strong cyclic planning via symbolic model checking. Artificial Intelligence **147**(1–2) (2003) 35–84
2. Edelkamp, S., Kissmann, P.: Solving fully-observable non-deterministic planning problems via translation into a general game. In: Proc. 32nd German Annual Conference on Artificial Intelligence (KI'09). (2009)
3. Bryce, D., Kambhampati, S., Smith, D.E.: Planning graph heuristics for belief space search. Journal of Artificial Intelligence Research **26** (2006) 35–99
4. Hoffmann, J., Brafman, R.I.: Contingent planning via heuristic forward search with implicit belief states. In: Proc. 15th International Conference on Automated Planning and Scheduling (ICAPS'05). (2005) 71–80
5. Bercher, P., Mattmüller, R.: A planning graph heuristic for forward-chaining adversarial planning. In: Proc. 18th European Conference on Artificial Intelligence (ECAI'08). (2008) 921–922
6. Hansen, E.A., Zilberstein, S.: LAO*: A heuristic search algorithm that finds solutions with loops. Artificial Intelligence **129**(1–2) (2001) 35–62
7. Nilsson, N.J.: Principles of Artificial Intelligence. Springer (1980)
8. Bercher, P.: Anwendung von Pattern-Database-Heuristiken zum Lösen nichtdeterministischer Planungsprobleme. Diplomarbeit, Albert-Ludwigs-Universität Freiburg im Breisgau (2009)
9. Edelkamp, S.: Planning with pattern databases. In: Proc. 6th European Conference on Planning (ECP'01). (2001) 13–24
10. Edelkamp, S.: Automated creation of pattern database search heuristics. In: Proc. 4th Workshop on Model Checking and Artificial Intelligence (MoChArt'06). (2006) 35–50
11. Haslum, P., Botea, A., Bonet, B., Helmert, M., Koenig, S.: Domain-independent construction of pattern database heuristics for cost-optimal planning. In: Proc. 22nd AAAI Conference on Artificial Intelligence (AAAI'07). (2007) 1007–1012
12. Hoffmann, J., Nebel, B.: The FF planning system: Fast plan generation through heuristic search. Journal of Artificial Intelligence Research **14** (2001) 253–302
13. Rintanen, J.: Constructing conditional plans by a theorem-prover. Journal of Artificial Intelligence Research **10** (1999) 323–352