Proceedings of the Twentieth International Conference on Automated Planning and Scheduling (ICAPS 2010)

Pattern Database Heuristics for Fully Observable Nondeterministic Planning

Robert Mattmüller and **Manuela Ortlieb** and **Malte Helmert**
University of Freiburg, Germany, {mattmuel,ortlieb,helmert}@informatik.uni-freiburg.de

Pascal Bercher
University of Ulm, Germany, pascal.bercher@uni-ulm.de

Abstract

When planning in an uncertain environment, one is often interested in finding a contingent plan that prescribes appropriate actions for all possible states that may be encountered during the execution of the plan. We consider the problem of finding strong cyclic plans for fully observable nondeterministic (FOND) planning problems. The algorithm we choose is LAO*, an informed explicit state search algorithm. We investigate the use of pattern database (PDB) heuristics to guide LAO* towards goal states. To obtain a fully domain-independent planning system, we use an automatic pattern selection procedure that performs local search in the space of pattern collections. The evaluation of our system on the FOND benchmarks of the Uncertainty Part of the International Planning Competition 2008 shows that our approach is competitive with symbolic regression search in terms of problem coverage, speed, and plan quality.

Introduction

For an agent planning in an uncertain environment it is not always sufficient to simply assume that all its actions will succeed and to replan upon failure. Rather, it can be advantageous to compute a contingent plan that prescribes actions for all possible states resulting from nondeterministic action outcomes completely ahead of execution. Specifically, in this work, we are concerned with finding strong cyclic plans (Cimatti et al. 2003) for fully observable nondeterministic (FOND) planning problems. Whereas strong plans, i.e., plans guaranteed to lead to a goal state in a finite number of steps, can be found using AO* search (Martelli and Montanari 1973), LAO* search (Hansen and Zilberstein 2001) is better suited to find strong cyclic plans, i.e., plans that may loop indefinitely as long as they do not contain dead ends and there is always a chance of making progress towards the goal.

As shown in earlier work (Hoffmann and Brafman 2005; Bryce, Kambhampati, and Smith 2006; Bercher and Mattmüller 2008), using AO* (or LAO*) in conjunction with an informative heuristic can be an efficient way to find contingent plans of high quality. Most heuristic search planners for nondeterministic problems use a delete relaxation heuristic to guide the search. Since it is not straightforward to make delete relaxations take the nondeterminism into account properly, in this work we investigate the use of pattern database (PDB) heuristics (Culberson and Schaeffer 1998; Edelkamp 2001). The abstractions underlying the PDBs can be designed to preserve the original nondeterminism. We use local search in the space of pattern collections (Haslum et al. 2007) to obtain suitable PDBs.

The contribution of this paper consists of the application of PDB heuristics to domain-independent strong cyclic planning, the definition of a suitable abstraction mapping, definitions of abstract costs and their computation, as well as an implementation and empirical evaluation of the resulting planning algorithm in comparison to Gamer (Kissmann and Edelkamp 2009), a planner that performs BDD-based planning as model checking (Cimatti et al. 2003). Minor contributions are the adaptations of a finite-domain representation of planning problems and of a pattern selection algorithm from the deterministic to the nondeterministic context.

Preliminaries

FOND SAS$^+$ Planning Tasks

We assume that the planning problem is given in a finite-domain representation that can be obtained from Probabilistic PDDL (PPDDL) using an adaptation of Helmert's PDDL-to-SAS$^+$ translation algorithm (2009).

A *fully observable nondeterministic SAS$^+$* (Bäckström and Nebel 1995) planning task is a tuple $\Pi = \langle \mathcal{V}, s_0, s_\star, \mathcal{O} \rangle$ consisting of the following components: $\mathcal{V}$ is a finite set of *state variables* v, each with a finite *domain* $\mathcal{D}_v$ and an *extended domain* $\mathcal{D}_v^+ = \mathcal{D}_v \uplus \{\bot\}$, where $\bot$ denotes the *undefined* or *don't-care* value. A *partial state* is a function s with $s(v) \in \mathcal{D}_v^+$ for all $v \in \mathcal{V}$. We say that s is *defined* for $v \in \mathcal{V}$ if $s(v) \neq \bot$. A *state* is a partial state s that is defined for all $v \in \mathcal{V}$. The set of all states s over $\mathcal{V}$ is denoted as $\mathcal{S}$. Depending on the context, a partial state s_p can be interpreted either as a *condition*, which is *satisfied* by a state s iff s agrees with s_p on all variables for which s_p is defined, or as an *update* on a state s, resulting in a new state s' that agrees with s_p on all variables for which s_p is defined, and with s on all other variables. The *initial state* s_0 of a problem is a state, and the *goal description* $s_\star$ is a partial state. $\mathcal{O}$ is a finite set of *actions* of the form $a = \langle Pre, Eff \rangle$, where the *precondition* Pre is a partial state, and the *effect* Eff

is a finite set of partial states eff, the *nondeterministic outcomes* of a. The *application* of a nondeterministic outcome eff to a state s is the state $\mathit{app}(\mathit{eff}, s)$ that results from updating s with eff. The application of an effect Eff to s is the set of states $\mathit{app}(\mathit{Eff}, s) = \{\mathit{app}(\mathit{eff}, s) \mid \mathit{eff} \in \mathit{Eff}\}$ that might be reached by applying a nondeterministic outcome from Eff to s. An action is *applicable* in s iff its precondition is satisfied in s. The application of a to s is $\mathit{app}(a, s) = \mathit{app}(\mathit{Eff}, s)$ if a is applicable in s, and undefined otherwise. All actions have unit cost.

Strong and Strong Cyclic Planning

The semantics of a planning task $\Pi = \langle \mathcal{V}, s_0, s_\star, \mathcal{O} \rangle$ can be defined via AND/OR graphs over the states $\mathcal{S}$. An *AND/OR graph* $\mathcal{G} = \langle N, C \rangle$ consists of a set of *nodes* N and a set of *connectors* C, where a connector is a pair $\langle n, M \rangle$, connecting the *parent* node $n \in N$ to a nonempty set of *children* $M \subseteq N$. A (directed) *path* in $\mathcal{G}$ is a sequence of nodes successively linked by connectors. Often, an AND/OR graph contains a distinguished *initial node* $n_0 \in N$ and a set of *goal nodes* $N_\star \subseteq N$.

The *strong preimage* of a set of nodes $X \subseteq N$ is the set of connectors that are *guaranteed* to lead to a node in X, and the *weak preimage* of X is the set of connectors that *may* lead to a node in X. Formally,

$$\mathit{spre}(X) = \{\langle n, M \rangle \in C \mid M \subseteq X\} \quad \text{and}$$
$$\mathit{wpre}(X) = \{\langle n, M \rangle \in C \mid M \cap X \neq \emptyset\}.$$

We write $N(C') = \{n \in N \mid \exists M \subseteq N \colon \langle n, M \rangle \in C'\}$ to denote the set of nodes with an outgoing connector in $C' \subseteq C$.

A *subgraph* of an AND/OR graph $\mathcal{G}$ is an AND/OR graph $\mathcal{G}' = \langle N', C' \rangle$ such that $N' \subseteq N$ and C' only contains connectors from C whose involved nodes (parent and children) are contained in N'. We usually require that the initial node n_0 of $\mathcal{G}$ is contained in N' as the initial node n'_0 of $\mathcal{G}'$, and that the set of goal nodes is $N'_\star = N_\star \cap N'$. The following properties of a subgraph $\mathcal{G}'$ of $\mathcal{G}$ are of interest:

Acyclicity: $\mathcal{G}'$ contains no path from n to n for any $n \in N'$.

Closedness: For all nongoal nodes $n \in N' \setminus N'_\star$, there is exactly one outgoing connector $c = \langle n, M \rangle$ in C'.

Properness: For all nongoal nodes $n \in N' \setminus N'_\star$, there is a finite path in $\mathcal{G}'$ starting at n and ending in a goal node $n_\star \in N'_\star$.

The *AND/OR graph induced by* Π is the graph $\mathcal{G} = \langle N, C \rangle$, where N is the set of states $\mathcal{S}$ of the planning task, and where there is one connector $\langle s, \mathit{app}(a, s) \rangle \in C$ for each state s and action a applicable in s leading from s to the states that might result from different nondeterministic outcomes of a. The initial node is $n_0 = s_0$, and a node n is a goal node in $N_\star$ iff it satisfies the goal description $s_\star$. A subgraph of the AND/OR graph induced by Π is called a *strong cyclic plan* if it is closed and proper, and a strong cyclic plan is called a *strong plan* if it is acyclic. A subgraph is a *weak plan* if it contains a path from n_0 to a goal node. Strong cyclic plans ("trial-and-error strategies") are a compromise between the overly optimistic view of weak plans and the strict requirement of strong plans that an action that is supposed to be part of a plan may never completely fail or even repulse the agent away from the goal. Strong cyclic plans allow such actions, as long as there is some chance to move closer to the goal and no danger of ending up in a dead end.

During the construction of a plan, an *explicit graph* $\mathcal{G}' = \langle N', C' \rangle$ is maintained that is a connected subgraph of $\mathcal{G}$ with the property that for all nongoal nodes n, either all outgoing connectors from n in $\mathcal{G}$ are contained in C' (n is *expanded*) or none of them is contained in C' (n is *unexpanded*). A *partial strong cyclic plan* (*partial strong plan*) is a subgraph of the explicit graph that satisfies the properties of a strong cyclic plan (strong plan), with the exception that closedness is relaxed such that outgoing connectors are only required for *expanded* nongoal nodes and that properness is not required.

AO* and LAO* Search

The problems of finding a strong plan or a strong cyclic plan, given a planning task Π, can be solved by AO* and LAO* graph search, respectively. AO* search (Martelli and Montanari 1973) is an algorithm that gradually builds an explicit graph until a strong solution has been found or the graph has been completely generated. Starting from n_0, in each step, it first extracts a partial solution by tracing down the most promising connectors, expands one or more of the unexpanded nongoal nodes encountered, and updates the information about which outgoing connectors are deemed most promising given the information obtained from the last expansion. The quality f of unexpanded nongoal nodes is estimated using a heuristic h, and in interior nodes, it is an aggregate of the qualities of the successor nodes.

Whereas AO* is sufficient to find strong plans, strong cyclic plans can be found using an extension to AO* called LAO* (Hansen and Zilberstein 2001). Unlike AO*, which uses backward induction, LAO* uses a dynamic programming algorithm like policy iteration or value iteration to update node estimates, thus allowing it to find solutions with loops as well, while still following a heuristic guidance. Pseudocode of (a variant of) LAO* is given in Algorithm 1. In the pseudocode, in which we assume that a solution exists, $\mathcal{G}$ is the implicit graph, $\mathcal{G}'$ the explicit graph, and $\textsc{Trace}(\mathcal{G}')$ traces down marked connectors in $\mathcal{G}'$ and returns the corresponding subgraph, which is considered incomplete if it still contains unexpanded nongoal nodes. These nodes are returned by UNEXPANDEDNONGOAL and then expanded simultaneously (EXPANDALL), i.e., their successor nodes and corresponding connectors are incorporated into $\mathcal{G}'$. After initializing the cost estimates f of all new nodes, the subgraph $\mathcal{Z}$ of nodes to be updated is chosen as the portion of $\mathcal{G}'$ weakly BACKWARDREACHable from the freshly expanded nodes. While VALUEITERATION is performed on $\mathcal{Z}$, the algorithm maintains the invariant that for each expanded nongoal node, exactly one outgoing connector minimizing f is marked.

Often, the search is equipped with a *solve-labeling procedure* that can be used to decide when a solution has been found and as a means of pruning the search space, since out-

Algorithm 1 LAO*($\mathcal{G}$)

$\mathcal{G}' \leftarrow \langle n_0, \emptyset \rangle$
while n_0 unsolved **do**
  $E \leftarrow$ UNEXPANDEDNONGOAL(TRACE($\mathcal{G}'$))
  if $E = \emptyset$ **then** $E \leftarrow$ UNEXPANDEDNONGOAL($\mathcal{G}'$)
  $N_{\text{new}} \leftarrow$ EXPANDALL(E)
  $f(n') \leftarrow \begin{cases} 0 & \text{if } n' \in N_\star \\ h(n') & \text{otherwise} \end{cases}$ for all $n' \in N_{\text{new}}$
  $\mathcal{Z} \leftarrow$ BACKWARDREACH(E)
  SOLVELABELING($\mathcal{G}'$)
  VALUEITERATION($\mathcal{Z}$)
return TRACE($\mathcal{G}'$)

going connectors from solved nodes do not need to be traced down any more. In strong cyclic planning, a node can be marked as *solved* if it is a goal node or if there is an applicable action that has a chance of leading to a solved node and that is guaranteed not to lead to potential dead-end nodes. Technically, the solve-labeling procedure for strong cyclic planning is a nested fixed point computation, outlined in Algorithm 2. Note that the outer loop computes a greatest fixed point, whereas within that loop, first the least fixed point of the set of all connectors weakly backward-reachable from the goal nodes along solved connectors in C_s is computed, followed by the computation of the greatest fixed point of the set of all connectors in C'_s guaranteed not to lead outside of C'_s.

Algorithm 2 SOLVELABELING($\mathcal{G}$)

$C_\text{s} \leftarrow C$
while C_s has not reached a fixed point **do**
  $C'_\text{s} \leftarrow \{\langle n, M\rangle \in C \mid M \cap N_\star \neq \emptyset\}$
  while C'_s has not reached a fixed point **do**
    $C'_\text{s} \leftarrow C'_\text{s} \cup (\mathit{wpre}(N(C'_\text{s})) \cap C_\text{s})$
  while C'_s has not reached a fixed point **do**
    $C'_\text{s} \leftarrow C'_\text{s} \cap \mathit{spre}(N(C'_\text{s}) \cup N_\star)$
  $C_\text{s} \leftarrow C'_\text{s}$
return $N(C_\text{s})$

Given an informative heuristic, more promising parts of the graph are likely to be expanded before the less promising ones by AO* and LAO*, and irrelevant portions of the search space may never be explored before a solution is found.

Heuristic Function

The performance of AO* and LAO* heavily depends on an appropriate heuristic estimator for unexpanded nongoal nodes. Our primary aim is to find *some* plan *fast* rather than to find an *optimal* plan, and hence the heuristic estimator we use does not necessarily have to accurately reflect remaining plan costs, but it *should* reflect the expected remaining search effort.

If we assume tree search for a strong plan, a constant branching factor b of the connectors, and a constant goal depth d below the current node, then the size of any nondegenerate solution tree rooted at the current node and even more the search effort, will be exponential in d. Therefore, minimizing d helps minimizing the search effort. If the distance from the current node to the necessary goal nodes is not constant, then because of the tree shape, for complete nondegenerate trees it is the *maximal* goal distance that matters. Thus, for strong planning, a reasonable measure for the remaining search effort below a node n is the depth of a depth-minimizing solution for the subproblem corresponding to n. We cannot compute this depth directly without solving the whole subproblem, but using an estimator that returns the depth of a depth-minimizing plan for a simplified problem can be expected to provide useful guidance. However, there are still two questions to be answered:

- What kind of simplification should be used?
- How can this estimate be adapted from strong to strong cyclic plans?

To answer the first question, note that in classical planning, the most important classes of simplifications are delete relaxations and projections. In nondeterministic planning, we additionally have the choice of whether or not to relax the nondeterminism and allow the agent not only to choose the action, but also its outcome among the possible nondeterministic outcomes.

It is easy to come up with examples showing that relaxing nondeterminism can lead to heuristic estimates that guide the search in the wrong direction, whereas with the nondeterminism represented in the heuristic, the search is guided in the right direction. Therefore, we would like to retain nondeterminism in the relaxation. Since, unlike in combination with delete relaxation, this is straightforward with projections, for the rest of this work we answer the question what kind of simplification to be used with *projection to a subset of the state variables*.

To answer the second question, first notice that in strong cyclic planning, the depth of a depth-minimizing plan is no longer well-defined, since there may be infinitely long paths in a plan. We can, however, make the simplifying assumption that all nondeterministic outcomes of the actions are equally likely, and replace the depth of a plan (the *maximal* number of steps to a goal) by the *expected* number of steps to a goal in order to obtain a well-defined heuristic.

We can express both variants of the heuristic (with maximum and expected value) in a Bellman formulation (Bellman 1957) as

$$h(n) = \begin{cases} 0 & \text{if } n \text{ is a goal,} \\ 1 + \min\limits_{\langle n,M\rangle \in C} \max\limits_{n' \in M} h(n') & \text{otherwise,} \end{cases} \tag{1}$$

in the case of maximization, and as

$$h(n) = \begin{cases} 0 & \text{if } n \text{ is a goal,} \\ 1 + \min\limits_{\langle n,M\rangle \in C} \frac{1}{|M|} \sum\limits_{n' \in M} h(n') & \text{otherwise,} \end{cases} \tag{2}$$

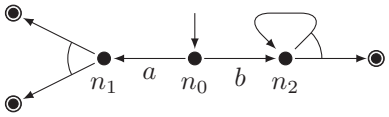
for the expected value.

It is easy to see that with maximization, h is guaranteed to give finite values for a node n iff there is a strong plan

starting in n. With expected values, h is guaranteed to have finite values for a node n iff there is a strong cyclic plan starting in n (i.e., if n can be marked as solved by the solve-labeling procedure for strong cyclic planning). The reason is that $h(n)$ is the expected number of steps to a goal node using a strong cyclic plan minimizing that expected number, which must be finite, since a random walk of sufficient finite length in the current strongly connected component (SCC) of the abstract state space always has a nonzero chance of making irreversible progress towards an SCC strictly closer to the goal, and there are only finitely many SCCs.

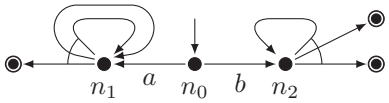
The following examples show how the heuristic trades off plan lengths against chances of success and how states admitting "less cyclic" plans are preferred to states only admitting "more cyclic" plans. In both examples, n_0 is the current (abstract) node, we use Eq. 2 to compute h-values, and we want to know which (abstract) action applicable in n_0 is the most promising. Initial nodes are depicted with an incoming edge without source node, goal nodes with circles, and edges belonging to the same connector with a joining arc.

- Strong vs. strong cyclic plans:



  Here, $h(n_1) = 1$ and $h(n_2) = 1 + \frac{1}{2}(0 + h(n_2))$, i.e., $h(n_2) = 2$. Therefore, the action a leading to a state admitting a strong subplan is preferred to the action b leading to a state only admitting a strong cyclic subplan.

- Cyclicity of strong cyclic plans:



  Here, $h(n_1) = 1+\frac{1}{3}(0+2\cdot h(n_1))$ and $h(n_2) = 1+\frac{1}{3}(2\cdot 0 + h(n_2))$, i.e., $h(n_1) = 3$ and $h(n_2) = \frac{3}{2}$, so the "less cyclic" subplan is preferred. Note that this example is simplified for presentation, since the outgoing connector from n_1 collapses to a connector $\langle n_1, M\rangle$ with $|M| = 2$ because of the set representation of successors.

Abstractions

Syntactic Projections

Pattern database heuristics are based on computing cost or search effort estimates in an abstract version of the original planning problem. In this section, we discuss syntactic projections for fully observable nondeterministic SAS$^+$ planning tasks.

A *pattern* is a subset $P \subseteq \mathcal{V}$ of the variables, and a *pattern collection* $\mathcal{P} \subseteq 2^{\mathcal{V}}$ is a set of patterns. The *projection* of a planning task Π to P is the task that results from Π if all variables outside of P are ignored, states are merged into one abstract state if they agree on all variables in P, and conditions and effects are restricted to P. Formally, the abstract task only contains the variables $\mathcal{V}\downarrow_P = \mathcal{V} \cap P$, the projection $s\downarrow_P$ of a (partial) state s is s restricted to P, i.e., $s\downarrow_P(v) = s(v)$ for all $v \in P$, and projections of effects, actions, action sets and tasks are defined element-wise and component-wise: $\mathit{Eff}\downarrow_P = \{\mathit{eff}\downarrow_P \mid \mathit{eff} \in \mathit{Eff}\}$, $\langle \mathit{Pre}, \mathit{Eff}\rangle\downarrow_P = \langle \mathit{Pre}\downarrow_P, \mathit{Eff}\downarrow_P\rangle$, $\mathcal{O}\downarrow_P = \{a\downarrow_P \mid a \in \mathcal{O}\}$, and $\langle \mathcal{V}, s_0, s_\star, \mathcal{O}\rangle\downarrow_P = \langle \mathcal{V}\downarrow_P, s_0\downarrow_P, s_\star\downarrow_P, \mathcal{O}\downarrow_P\rangle$. We refer to states, connectors etc. from the state space of the original problem as *concrete* states, connectors etc., and to those from the projection as *abstract* states, connectors etc.

By definition, the projection of a planning task is again a planning task. Its induced AND/OR graph is in general at most as large as the induced AND/OR graph of the original planning task. The syntactic projection has the property that it preserves action applicability, effect applications, and solvedness of states. More precisely, let $s \in \mathcal{S}$ be a concrete state, $a = \langle \mathit{Pre}, \{\mathit{eff}_1, \ldots, \mathit{eff}_n\}\rangle$ a concrete action, solved the set of concrete states admitting a strong cyclic plan, and solved_P the set of abstract states admitting a strong cyclic plan. Then (1) if a is applicable in s, then $a\downarrow_P$ is applicable in $s\downarrow_P$, (2) $\mathit{app}(\mathit{eff}, s)\downarrow_P = \mathit{app}(\mathit{eff}\downarrow_P, s\downarrow_P)$ and $\mathit{app}(\mathit{Eff}, s)\downarrow_P = \mathit{app}(\mathit{Eff}\downarrow_P, s\downarrow_P)$, and (3) $s \in \mathit{solved}$ implies $s\downarrow_P \in \mathit{solved}_P$. Claims (1) and (2) immediately follow from the definition of projections. For (3), consider the solve-labeling procedure described in Algorithm 2. We call nodes in $N(C_\mathrm{s})$ *solved*, nodes in $N(C'_\mathrm{s})$ after the first and before the second inner loop *connected*, and nodes in $N(C'_\mathrm{s})$ after the second inner loop *safe*. Initially, all concrete and abstract nodes are solved, so the induction base holds. In the i-th iteration of the outer loop, a concrete node n is marked as connected (in the first inner loop) iff there is a finite path along nodes marked as solved in the previous iteration starting at n and ending in a goal node. Since, by (1) and (2), actions remain applicable in the abstraction and concrete successors are represented by abstract successors, and since, by induction hypothesis, solved concrete nodes are projected to solved abstract nodes, this sequence has an abstract counterpart showing that $n\downarrow_P$ is connected as well. In the safe-labeling of the current iteration (the second inner loop), initially, all connected nodes are marked as safe, and a node is only removed from the safe nodes if all its outgoing connectors may potentially lead to an unsafe nongoal node. An abstract node $n\downarrow_P$ is only removed from the abstract safe nodes if there is no abstract connector that guarantees staying in the abstract safe nodes. But then all corresponding concrete nodes n would be removed from the concrete safe nodes as well, for if there were a corresponding concrete node n and an outgoing connector c of n only leading to safe nodes, the projection of c to P would only lead to safe abstract nodes by induction hypothesis of the inner loop. Since the solved nodes in the next iteration are the remaining safe nodes, the induction step holds as well.

Pattern Database Heuristics

The abstractions and abstract costs or search effort estimates 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 values of the abstract states have been stored during the preprocessing stage. Each ab-

straction is a projection of the original planning problem to a pattern P and defines a heuristic function h^P.

Since the size of the abstract state space grows exponentially in the size of the pattern P, reasonable patterns should not be too large. So, typically, one pattern will include only a small fraction of $\mathcal{V}$, and hence the corresponding abstraction will completely ignore the contribution of many variables to the hardness of solving the problem that could be captured if the variables were included in P. Instead of using arbitrarily large patterns, one usually resorts to using several smaller patterns and aggregating the abstract costs of a state in the abstractions corresponding to the patterns.

Given a finite collection $\mathcal{P}$ of patterns, one could define the heuristic function $h^{\mathcal{P}}(n) = \max_{P \in \mathcal{P}} h^P(n)$. Since we want to maintain heuristic values as informative as possible, however, maximization is often not sufficient and replacing maximization by summation could produce more informative heuristic values. On the other hand, adding h-values does not provide an additional advantage if the added heuristics reflect the contribution of the same or at least similar sets of actions, whereas other actions are still disregarded. Therefore, similarly to classical planning (Edelkamp 2001), we say that patterns $P_1, \ldots, P_k$ are *additive* if there is no action $a = \langle Pre, Eff \rangle \in \mathcal{O}$ that affects variables from more than one of the patterns, i.e., no action for which $effvars(Eff) \cap P_i \neq \emptyset$ for more than one $i = 1, \ldots, k$, where $effvars(Eff) = \{v \in \mathcal{V} \mid \exists eff \in Eff\colon eff(v) \neq \bot\}$. If the patterns in $\mathcal{P}$ are additive, we can define $h^{\mathcal{P}}$ as $h^{\mathcal{P}}(n) = \sum_{P \in \mathcal{P}} h^P(n)$ instead of via maximization. Finally, given a set $\mathcal{M}$ of additive pattern collections, we can define the heuristic function $h^{\mathcal{M}}(n) = \max_{\mathcal{P} \in \mathcal{M}} \sum_{P \in \mathcal{P}} h^P(n)$, which in general dominates all of the heuristics $h^{\mathcal{P}}$.

Note that, unlike in optimal classical planning, where additivity is required to ensure admissibility of the resulting heuristic, we do not *necessarily* require additivity here, but rather use it as a means of getting an *informative*, not an *admissible*, heuristic, since additive pattern collections tend to cover action costs more accurately than collections of dependent or even overlapping patterns.

Pattern Selection as Local Search

To determine the pattern collections $\mathcal{P}$ to be used, we use the local search algorithm by Haslum et al. (2007). Even though the evaluation function for pattern collections in their algorithm is based on the expected number of node expansions during IDA* search, not AO* or LAO* search, with the pattern collection under consideration, we believe that the patterns that are obtained that way are still useful in nondeterministic planning. The reasoning behind this is that the question whether variables are related closely enough to be grouped into one pattern is largely orthogonal to the question whether the actions in the planning task are deterministic.

Let $\mathcal{P}$ be a pattern collection. Then the *canonical heuristic function* of $\mathcal{P}$ is the function $h^{\mathcal{P}}_{\mathrm{C}} = \max_{\mathcal{P}' \in \mathcal{M}} \sum_{P \in \mathcal{P}'} h^P$, where $\mathcal{M}$ is the set of all maximal additive subsets of $\mathcal{P}$. The pattern selection algorithm considers a pattern collection $\mathcal{P}$ to be better than a pattern collection $\mathcal{P}'$ if the expected number of node expansions of an IDA* search with $h^{\mathcal{P}}_{\mathrm{C}}$ is lower than the expected number of node expansions of an IDA* search with $h^{\mathcal{P}'}_{\mathrm{C}}$.

Algorithm 3 PATTERNSELECTION(Π, $bound$)

```
P ← INITIALCOLLECTION(Π)
while ‖P‖ < bound do
  S̄ ← CHOOSESAMPLES(Π)
  N ← NEIGHBORHOOD(P, Π)
  for all P' ∈ N do
    δ(P') ← IMPROVEMENT(P', P, S̄, Π)
  if δ(P') = 0 for all P' ∈ N then break
  P ← argmax_{P'∈N} δ(P')
return P
```

The pattern selection presented as pseudocode in Algorithm 3 performs a hill-climbing search in the space of pattern collections, starting with the initial collection INITIALCOLLECTION(Π) that contains one singleton pattern for each variable occurring in the goal. The search continues until either there is no more improvement or the cumulative sizes of the pattern databases contained in the current pattern collection $\mathcal{P}$ exceed a given $bound$. The size of the pattern collection is $\|\mathcal{P}\| = \sum_{P \in \mathcal{P}} \prod_{v \in P} |\mathcal{D}_v|$. The NEIGHBORHOOD($\mathcal{P}, \Pi$) of the pattern collection $\mathcal{P}$ with respect to planning task Π contains a pattern collection $\mathcal{P}'$ if there is a $P \in \mathcal{P}$ and a variable $v \notin P$ that is causally relevant to P, such that $\mathcal{P}' = \mathcal{P} \uplus \{P'\}$ for $P' = P \cup \{v\}$. The IMPROVEMENT($\mathcal{P}', \mathcal{P}, \overline{\mathcal{S}}, \Pi$) of $\mathcal{P}'$ over $\mathcal{P}$ with respect to the sample states in $\overline{\mathcal{S}}$ is the estimated number of states $s \in \overline{\mathcal{S}}$ for which $h^{\mathcal{P}'}_{\mathrm{C}}$ returns a higher value than $h^{\mathcal{P}}_{\mathrm{C}}$. This is precisely the number of states $s \in \overline{\mathcal{S}}$ for which $h^{P'}(s) > h^{\mathcal{P}}(s) - \sum_{P \in S \setminus \{P'\}} h^P(s)$ for some additive subset $S \subseteq \mathcal{P}'$ that includes P'. The algorithm terminates eventually, since $\mathcal{P}$ only increases in size and there are only finitely many different patterns.

In each iteration, a call to CHOOSESAMPLES(Π) is used to draw n sample states (non-uniformly) from the state space that are used to compute an estimate of how much guidance $h^{\mathcal{P}}_{\mathrm{C}}$ will provide. A sample is the last state on a random walk through the state space, where successor states are selected uniformly at random among all possible successors. The length of the random walk is chosen uniformly between 0 and a fixed upper bound $d = 2 \cdot h^{\mathcal{P}}_{\mathrm{C}}(s_0)$.

Implementation Details

Abstract Cost Computation

Let $P \subseteq \mathcal{V}$ be a pattern and $\mathcal{G} = \langle N, C \rangle$ the AND/OR graph induced by the projection of the given planning task to P. Then the abstract costs are defined as the result of value iteration on the relevant part of $\mathcal{G}$ (Algorithm 4).

Here, FORWARDREACH($\mathcal{G}$) builds the portion of $\mathcal{G}$ that is forward reachable from n_0, SOLVELABELING($\mathcal{G}$) labels solved nodes in $\mathcal{G}$, and RESTRICTTOSOLVED($\mathcal{G}$, $solved$) restricts $\mathcal{G}$ to solved nodes. We have already shown that all forward reachable solved concrete states are represented by an abstract state that will be marked as $solved$. Therefore,

Algorithm 4 ABSTRACTCOSTCOMPUTATION($\mathcal{G}$)

$\mathcal{G} \leftarrow$ FORWARDREACH($\mathcal{G}$)
$solved \leftarrow$ SOLVELABELING($\mathcal{G}$)
$\mathcal{G} \leftarrow$ RESTRICTTOSOLVED($\mathcal{G}, solved$)
$h'(n) \leftarrow \infty$ for all $n \in N \setminus solved$
$h'(n) \leftarrow 0$ for all $n \in solved$
repeat
  $h \leftarrow h'$
  for all $n \in solved \setminus N_\star$ **do**
    $h'(n) \leftarrow 1 + \min_{\langle n, M \rangle \in C} \frac{1}{|M|} \sum_{n' \in M} h(n')$
until $\max_{n \in N} |h'(n) - h(n)| < \varepsilon$
return h'

if we assign the heuristic value ∞ to all abstract states outside *solved* and prune the concrete search as soon as we encounter a concrete state s that maps to an abstract state $s\downarrow_P$ with $h(s\downarrow_P) = \infty$, the search remains complete.

On the remaining solved abstract nodes, the algorithm performs value iteration until the error falls below ε, and eventually returns the cost function (including the mapping of unsolved abstract states to ∞).

The value iteration is guaranteed to converge, since restricting the set of nodes on which value iteration is performed to those that are marked as solved guarantees that all cost values are bounded. Together with the fact that, starting with $h \equiv 0$, the Bellman update is a monotonically increasing function, this is sufficient to ensure convergence and hence termination.

Experiments

Setting and Benchmarks

We evaluated our planner on the benchmark problems of the IPC 2008 FOND track. The instances belong to the domains *blocksworld*, *faults*, *first-responders*, and *forest*.

We ran our planner with PDB heuristic, a variant of the FF heuristic (combining nondeterminism relaxation with delete relaxation) and the heuristic assigning 0 to goal nodes and 1 to all other nodes. We compared it to the winner of the FOND track 2008, Gamer (Kissmann and Edelkamp 2009), based on the criteria *coverage* (number of problems for which a strong cyclic plan was found), *speed* (time needed for each problem), *plan size* (number of state-action table entries), *plan quality* (expected number of steps to the goal under random selection of action outcomes), and *guidance* (number of nodes created by LAO*). The guidance criterion was only used in comparisons between different configurations of our planner, not when comparing against Gamer, which performs a symbolic search for which the guidance criterion is not meaningful.

Since all problems solved by either planner only admit strong cyclic plans and no strong plans, we only report times needed to find strong cyclic plans. In a setting where strong plans are the preferred solutions and a strong cyclic plan is only acceptable if no strong plan exists, our planner, like Gamer, could be configured to search for a strong plan first, using the same search framework as described for strong cyclic planning (with LAO* replaced by AO* and simpler solve-labeling and abstract cost computation procedures) and only turn to strong cyclic planning after determining that no strong plan exists.

Domain (probs)	PDB	DR	0/1	Gamer
blocksworld (30)	10	10	10	10
faults (55)	55	54	33	34
first-responders (100)	23	24	19	19
forest (90)	6	6	3	6
overall (275)	**94**	**94**	**65**	**69**

Table 1: Coverage of LAO* with PDB, delete relaxation, and 0/1 heuristic, compared to Gamer.

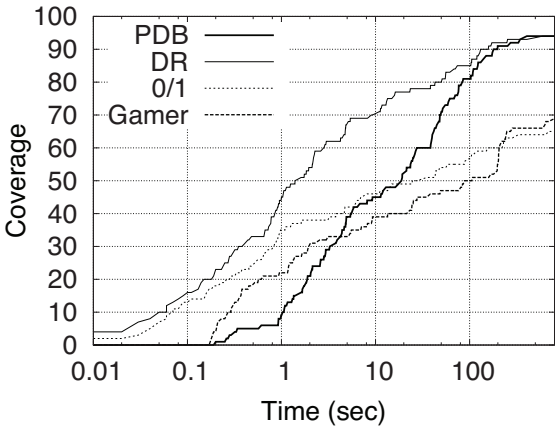


Figure 1: Coverage over time.

The experiments were conducted on a 2 GHz Intel Pentium E2180 CPU with 1 GB memory limit. In order to obtain comparable search times and coverage values for the different configurations of our planner, two time limits were set independently for preprocessing including PDB computation (5 minutes) and search (15 minutes minus preprocessing time). Gamer was run with an overall time limit of 15 minutes.

Results

We report the results obtained by our planner using 1000 samples in the pattern selection and convergence threshold $\varepsilon = 10^{-4}$ in the value iteration (both in the evaluation of abstract states and in the value iteration subroutine of LAO*).

Table 1 shows problem coverage per domain and overall after 15 minutes. Problems detected to be unsolvable by either planner are treated as if they were unsolved. Figure 1 shows the development of the coverage over time, with the PDB approach only gaining more covered instances after a longer preprocessing time and eventually catching up with the delete-relaxation approach.

Table 2 shows times in seconds (t), LAO* node creations (n), and plan sizes (s), of our planner with PDB heuristic, our planner with delete relaxation heuristic, and with the 0/1 heuristic, as well as the times and plan sizes of Gamer, on

Problem	PDBs				Delete relaxation			0/1 heuristic			Gamer				
	t	(t_p,	t_s)	s	n	t	s	n	t	s	n	t	(t_p,	t_s)	s
bw-1	25.10	(25.05,	0.05)	11	**43**	0.20	**10**	50	**0.16**	13	296	220.73	(172.17,	48.56)	**10**
bw-2	3.91	(3.83,	0.08)	**9**	293	0.28	**9**	293	**0.07**	**9**	**92**	211.27	(170.28,	40.99)	16
bw-3	4.23	(4.09,	0.14)	**17**	**931**	**0.48**	**17**	**931**	0.85	19	2335	206.07	(164.33,	41.74)	21
bw-4	4.90	(3.80,	1.10)	**17**	**8515**	**4.84**	32	23154	6.39	18	24406	203.46	(156.60,	46.86)	26
bw-5	4.88	(4.27,	0.61)	16	4899	1.39	16	5968	**0.92**	22	**3476**	202.66	(160.05,	42.61)	**13**
bw-6	4.39	(3.98,	0.41)	**11**	2960	0.87	**11**	2960	**0.72**	**11**	**2710**	196.37	(158.49,	37.88)	19
bw-7	5.43	(3.93,	1.50)	25	10277	1.95	25	8549	**1.28**	**24**	**5373**	198.17	(156.15,	42.02)	28
bw-8	5.90	(4.15,	1.75)	**14**	14515	**1.84**	16	**8718**	3.22	**14**	15754	197.07	(155.70,	41.37)	19
bw-9	4.42	(4.41,	0.01)	**8**	**34**	0.35	**8**	626	**0.29**	**8**	534	203.51	(158.31,	45.20)	10
bw-10	4.66	(4.25,	0.41)	**13**	**1988**	0.95	**13**	3080	**0.89**	**13**	3904	205.38	(161.01,	44.37)	**13**
faults-5-5	26.14	(26.09,	0.05)	65	**329**	**0.73**	65	509	43.75	**39**	6138	168.35	(51.92,	116.43)	65
faults-6-4	19.26	(18.42,	0.84)	124	**5987**	**1.75**	135	7072	35.03	**29**	13157	88.39	(29.52,	58.87)	138
faults-7-4	51.92	(42.01,	9.91)	280	46964	**3.00**	256	**15152**	157.93	32	39895	25.34	(13.75,	11.59)	**31**
faults-8-3	23.54	(19.55,	3.99)	275	26311	**4.76**	276	**26304**	58.39	**29**	32351	107.69	(38.45,	69.24)	262
faults-9-2	48.17	(20.66,	27.51)	135	19533	**0.78**	**33**	**4049**	14.24	88	10996	24.48	(10.31,	14.17)	135
faults-9-3	42.62	(39.61,	3.01)	429	**23836**	**9.03**	407	49068	240.82	**49**	82410	285.15	(111.77,	173.38)	371
faults-10-2	27.54	(27.19,	0.35)	167	**882**	**2.13**	122	15012	48.02	**119**	20358	84.25	(28.48,	55.77)	165
fr-1-6	2.83	(0.34,	2.49)	**8**	9776	2.27	**8**	9776	119.56	**8**	**7414**	**1.23**	(0.75,	0.48)	**8**
fr-2-4	**1.74**	(1.15,	0.59)	**8**	**1191**	2.17	**8**	7780	8.13	**8**	6400	38.31	(35.61,	2.70)	11
fr-4-3	2.61	(1.02,	1.59)	**10**	**8060**	**2.19**	**10**	**8060**	24.66	**10**	20928	631.95	(527.59,	104.36)	—
fr-4-6	50.15	(1.15,	49.00)	**13**	139964	55.59	**13**	153742	**43.28**	14	**48384**	249.51	(215.16,	34.35)	—
forest-2-5	21.31	(16.14,	5.17)	59	6378	13.43	**56**	**4138**	229.22	59	6841	**2.03**	(1.52,	0.51)	**56**
forest-2-8	26.71	(21.55,	5.16)	59	6378	13.09	**56**	**4138**	230.13	59	6843	**1.90**	(1.41,	0.49)	**56**

Table 2: Times, plan sizes, and nodes created for LAO* with PDB, delete relaxation, and 0/1 heuristic, compared to Gamer. Dashes in the s column for Gamer indicate plan files too large to process.

a subset of the benchmark problems solved by all four approaches, more specifically, the hard problems for which at least one algorithm needed more than 30 seconds to solve. In the case of PDB heuristics, the times for preprocessing (translation from PPDDL to FOND SAS$^+$ and PDB computation, t_p) and search (t_s) are reported separately, with their sum shown in the overall times column (t). Translation times were always below 0.2 seconds. For Gamer, the times for preprocessing including reachability analysis (t_p) and search (t_s), excluding plan output time, are reported. The plan sizes reported for Gamer are not the numbers of policy entries in the plans Gamer produces, but rather the number of entries that remain after restricting the policy to the states reachable following the policy. This postprocessing decreases policy sizes by several orders of magnitude (e.g., for blocksworld instance #1 the number of policy entries drops from 428527 to 10).

In the blocksworld domain, problems 1 to 10 are sufficiently simple to be solved by all planners, whereas the remaining 20 problems are too hard for all of them. On the solved problems, our planner outperforms Gamer in terms of search time, whereas plan sizes are similar. In the faults domain, the pure search time of our planner is mostly lower than that of Gamer. On the harder instances, even overall times of our planner, including PDB computation, are lower than the times needed by Gamer, and our coverage is noticeably higher. Again, plan sizes are similar. In the first-responders domain, coverages and plan sizes are similar, and running times are very mixed. In the forest domain, all approaches except LAO*+0/1 solve the same six problems with similar plan sizes, with Gamer needing less time.

Problem	PDB	DR	0/1	Gamer
bw-1	20.00	15.00	56.00	**14.00**
bw-2	14.00	14.00	14.00	**13.50**
bw-3	31.50	31.50	21.25	**17.50**
bw-4	29.00	33.75	32.00	**26.00**
bw-5	**21.50**	**21.50**	**21.50**	22.00
bw-6	**21.50**	**21.50**	**21.50**	**21.50**
bw-7	44.00	44.00	46.00	**22.00**
bw-8	**24.00**	28.00	**24.00**	**24.00**
bw-9	**12.50**	**12.50**	**12.50**	14.00
bw-10	**18.00**	**18.00**	**18.00**	**18.00**
faults-5-5	**6.12**	**6.12**	8.00	**6.12**
faults-6-4	9.44	8.94	13.00	**8.87**
faults-7-4	11.97	**11.75**	16.00	16.00
faults-8-3	17.69	17.70	21.00	**17.16**
faults-9-2	**24.01**	25.00	26.12	**24.01**
faults-9-3	20.90	21.92	22.50	**20.09**
faults-10-2	27.01	30.97	30.97	**27.00**
fr-1-6	**12.00**	**12.00**	**12.00**	14.00
fr-2-4	**14.00**	**14.00**	**14.00**	**14.00**
fr-4-3	**15.00**	**15.00**	**15.00**	—
fr-4-6	**20.00**	**20.00**	**20.00**	—
forest-2-5	22.83	24.00	22.83	**22.00**
forest-2-8	22.83	24.00	22.83	**22.00**

Table 3: Plan qualities (expected numbers of steps to the goal under random selection of action outcomes) for LAO* with PDB, delete relaxation, and 0/1 heuristic, compared to Gamer. Note that this definition of plan quality differs from the IPC definition for strong *acyclic* solutions, i.e., worst-case number of steps to the goal. Dashes in the column for Gamer indicate plan files too large to process.

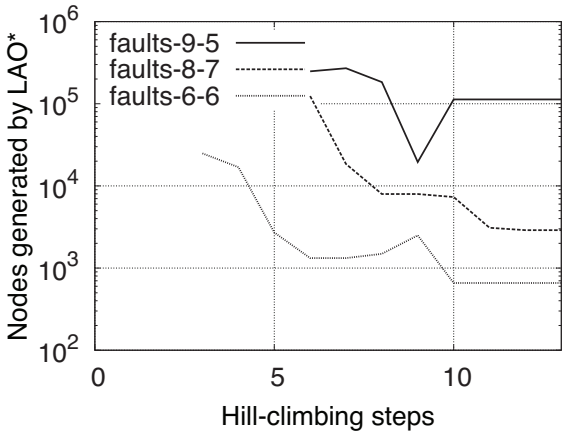


Figure 2: Guidance dependent on number of local search steps in pattern selection procedure.

Table 3 shows the expected numbers of steps to the goal under random selection of action outcomes of the plans found by the different approaches for the instances from Table 2. The results do not allow us to conclude that one of the approaches leads to significantly better plans with respect to this quality measure.

PDB heuristic and delete relaxation heuristic provide similar guidance to the search in all domains, and typically better guidance than the trivial 0/1 heuristic (Table 2 is biased towards problems for which LAO* with 0/1 heuristic accidentally appears well-guided, whereas considering all problems, the 0/1 heuristic provides a clearly worse guidance than PDBs and delete relaxation). In order to determine how different pattern collections guide the search, we interrupted the hill-climbing search in the space of pattern collections after k steps for increasing k and measured the guidance provided to LAO* by the current pattern collection after k steps. Figure 2 shows how the guidance improves with k for selected instances from the *faults* domain. Missing data points for small k indicate that LAO* timed out.

Conclusion

We presented and evaluated a domain-independent planner for fully observable nondeterministic planning problems using LAO* search guided by a PDB heuristic.

Our empirical evaluation suggests that heuristically guided progression search can be competitive with or even outperform uninformed symbolic regression search in terms of speed and coverage if an informative heuristic is used. The plans are of similar size and the expected numbers of steps to the goal when executing them, assuming uniform selection of action outcomes, are comparable as well. The comparison between delete relaxation heuristic and PDB heuristic shows that both heuristics guide the search similarly well, with PDB guidance significantly improving with more time spent on the pattern selection. Also, the sizes and qualities of the plans found are comparable, whereas regarding speed, LAO* search with delete relaxation heuristic is often faster than LAO* search with PDB heuristic, if search and preprocessing time are added. However, because of the simple table look-up during search, pure search times are often lower with a PDB heuristic and the larger the problems become, the more this can compensate for the higher preprocessing times.

Acknowledgments

We thank Peter Kissmann for his assistance with Gamer and the anonymous reviewers for their helpful suggestions.

This work was partly supported by the German Research Foundation (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), and as part of the Transregional Collaborative Research Center SFB/TRR 62 "Companion-Technology for Cognitive Technical Systems".

References

Bäckström, C., and Nebel, B. 1995. Complexity results for SAS^+ planning. *Comput. Intell.* 11(4):625–655.

Bellman, R. 1957. *Dynamic Programming*. Princeton University Press.

Bercher, P., and Mattmüller, R. 2008. A planning graph heuristic for forward-chaining adversarial planning. In *ECAI'08*, 921–922.

Bryce, D.; Kambhampati, S.; and Smith, D. E. 2006. Planning graph heuristics for belief space search. *JAIR* 26:35–99.

Cimatti, A.; Pistore, M.; Roveri, M.; and Traverso, P. 2003. Weak, strong, and strong cyclic planning via symbolic model checking. *Artif. Intell.* 147(1–2):35–84.

Culberson, J. C., and Schaeffer, J. 1998. Pattern databases. *Comput. Intell.* 14(3):318–334.

Edelkamp, S. 2001. Planning with pattern databases. In *ECP'01*, 13–24.

Hansen, E. A., and Zilberstein, S. 2001. LAO*: A heuristic search algorithm that finds solutions with loops. *Artif. Intell.* 129(1–2):35–62.

Haslum, P.; Botea, A.; Helmert, M.; Bonet, B.; and Koenig, S. 2007. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *AAAI'07*, 1007–1012.

Helmert, M. 2009. Concise finite-domain representations for PDDL planning tasks. *Artif. Intell.* 173(5–6):503–535.

Hoffmann, J., and Brafman, R. I. 2005. Contingent planning via heuristic forward search with implicit belief states. In *ICAPS'05*, 71–80.

Kissmann, P., and Edelkamp, S. 2009. Solving fully-observable non-deterministic planning problems via translation into a general game. In *KI'09*, volume 5803 of *LNCS*, 1–8. Springer.

Martelli, A., and Montanari, U. 1973. Additive AND/OR graphs. In *IJCAI'73*, 1–11.