Structural-Pattern Databases

Michael Katz and **Carmel Domshlak***
Faculty of Industrial Engineering & Management
Technion, Israel

Abstract

Explicit abstraction heuristics, notably pattern-database and merge-and-shrink heuristics, are employed by some state-of-the-art optimal heuristic-search planners. The major limitation of these abstraction heuristics is that the size of the abstract space has to be bounded by a (large) constant. Targeting this issue, Katz and Domshlak (2008b) introduced structural, and in particular fork-decomposition, abstractions, in which the planning task is abstracted by an instance of a tractable fragment of optimal planning. At first view, however, the lunch was not free. Some of the power of the explicit abstraction heuristics comes from pre-computing the heuristic function offline, and then determine $h(s)$ for each evaluated state s by a very fast lookup in a "database". In contrast, fork-decomposition offer a poly-time, yet far from being fast, computation.

In this contribution, we show that the time-per-node complexity bottleneck of the fork-decomposition heuristics can be successfully overcome. Specifically, we show that an equivalent of the explicit abstractions' notion of "database" exists for the fork-decomposition abstractions as well, and this despite of their exponential-size abstract spaces. Experimentally, we show that heuristic search with such "databased" fork-decomposition heuristics favorably competes with the state-of-the-art of optimal planning.

Introduction

Heuristic search is nowadays one of the leading approaches to cost-optimal planning. The difference between various cost-optimizing heuristic-search planners is mainly in the employed admissible heuristic functions. In state-space search, such a heuristic estimates the cost of achieving the goal from a given state, and being admissible, it guarantees not to overestimate that cost. During the last decade, numerous computational ideas evolved into new admissible heuristics for domain-independent planning; this includes delete-relaxing max heuristic $h_{\max}$ (Bonet and Geffner 2001), critical path heuristics h^m (Haslum and Geffner 2000), landmark heuristics h^L and h^{LA} (Karpas and Domshlak 2009), and abstraction heuristics such as pattern-database (Edelkamp 2001), merge-and-shrink (Helmert, Haslum, and Hoffmann 2007), and structural-pattern (Katz and Domshlak 2008b) heuristics. In addition, these heuristics have been combined via additive ensembles of admissible heuristics (see, e.g., Edelkamp (2001), Haslum et al. (2005), Katz and Domshlak (2008a), Karpas and Domshlak (2009)).

Apart from being as informative as possible, the heuristic should also be computable in polynomial time. In fact, the latter is a necessary, yet, in practice, insufficient condition. If any reasonable time cap is put on the planner, the heuristic computation per search node has to be really fast. For many problems, this has to be the case even if the heuristic is perfect, and this because of a large number of states evaluated along the optimal path. This well-known issue can be exemplified by the results of the cost-optimal planning track of the IPC-2008 competition in which even the best-performing heuristic search planner HSP^*_{F} (Haslum 2008) solved less problems within the time limit of 30 minutes than the baseline breadth-first search. With this "heuristic selection" criterion of the fast per-search-node computation in mind, here we focus on abstraction heuristics, and most closely, on structural-pattern heuristics.

Probably the most well-known abstraction heuristics are pattern database (PDB) heuristics that are based on projecting the planning task onto a subset of its state variables and then explicitly searching for optimal plans in the abstract space. The limitation of PDB heuristics in practice is that both the size of the abstract space and its dimensionality have to be fixed.[1] The more recent, merge-and-shrink abstractions, generalize PDB heuristics to overcome the latter limitation; instead of perfectly reflecting just a few state variables, merge-and-shrink abstractions allow for imperfectly reflecting all variables. The abstract space of merge-and-shrink is still searched explicitly, and thus it still has to be of fixed size.

Targeting *both* limitations of the PDB abstractions, Katz and Domshlak (2008b) introduced the framework of structural patterns in which the planning task is abstracted to a tractable fragment of optimal planning. In the extreme case where the problem itself belongs to that fragment, nothing will be abstracted at all, and the perfect heuristic will

*The work of both authors is partly supported by Israel Science Foundation grant 670/07.

[1] This does not necessarily apply to *symbolic* PDBs that, on some tasks, may provide exponential reduction of the PDB's representation (Edelkamp 2002).

be computed in polynomial time. Katz and Domshlak introduced a concrete instance of the structural patterns idea, called *fork decomposition*, in which the problem is projected (in a proper meaning of this term) onto fork and inverted fork subgraphs of the problem's causal graph. The promise of fork-decomposition has been shown via a formal asymptotic performance analysis on selected domains, but the empirical relevance has yet to be shown.

Examining the empirical effectiveness of the fork-decomposition heuristics, we have implemented and evaluated them using A^* on numerous IPC domains. On many domains, the quality of the fork-decomposition heuristics appeared to be very encouraging, to say the least. However, in all domains we pretty quickly bumped into the wall of the per-node computation complexity. While the planner was able to solve many challenging problems within a standard memory cap, it was typically running out of reasonable time limits. Surprising it was not. While pattern database and merge-and-shrink heuristics are also computationally prohibitive, they allow for pre-computing the heuristic values for all abstract states simultaneously, storing them in a lookup table, and then reusing that pre-search computation between search states. In contrast, the abstract spaces induced by structural patterns can be as large as the original state space, and thus they cannot be pre-solved and pre-stored exhaustively.

Overall, while the empirical (in)effectiveness of fork-decomposition was not surprising, given the good quality of its induced estimates, disappointing it was. Here, however, we show that the time-per-node complexity bottleneck of the fork-decomposition heuristics can be successfully overcome. Specifically, we show that an equivalent of the PDB's and merge-and-shrink's notion of "database" exists for the fork-decomposition abstractions as well, and this *despite of their exponential-size abstract spaces*. In fact, the same effect could possibly be achieved also for other (to be developed in the future) structural-pattern heuristics.

Unlike for PDBs and merge-and-shrink abstractions, *structural-pattern databases* (and in particular these for the fork-decomposition's fork and inverted fork abstractions) do not (and cannot) result in purely lookup computations of $h(s)$. In contrast, structural-pattern databases are based on a proper partition of heuristic computation (*h-partition*) into parts that can be shared between search states, and parts that shall be *computed* per individual state. We prove existence of such effective h-partitions for both fork and inverted fork abstractions. We then formally and empirically show that these h-partitions lead to fast pre-search and per-node computations that allow for exploiting the informativeness of the fork-decomposition heuristics in practice. Experimentally, we show that A^* equipped with the "databased" fork-decomposition heuristics favorably competes with the state-of-the-art of cost-optimal planning.

Preliminaries

We consider planning problems captured by the SAS$^+$ formalism (Bäckström and Nebel 1995) with non-negative action costs. Such a problem is given by a quintuple $\Pi = \langle V, A, I, G, c\rangle$, where:

- V is a set of *state variables*, each $v \in V$ is associated with a finite domain $\mathcal{D}(v)$; the value provided to a variable v by a (possibly partial) assignment p to V is denoted by $p[v]$. Each complete assignment to V is called a *state*, and $S = \mathcal{D}(V)$ is the *state space* of Π. I is an *initial state*. The *goal* G is a partial assignment to V; a state s is a *goal state* iff $G \subset s$.
- A is a finite set of *actions*. Each action a is a pair $\langle \mathsf{pre}(a), \mathsf{eff}(a)\rangle$ of partial assignments to V called *preconditions* and *effects*, respectively. By $A_v \subseteq A$ we denote the actions affecting the value of v. $c : A \to \mathbb{R}_0^+$ is a real-valued, non-negative *action cost* function.

An action a is applicable in a state s iff $s[v] = \mathsf{pre}(a)[v]$ whenever $\mathsf{pre}(a)[v]$ is specified. Applying a changes the value of v to $\mathsf{eff}(a)[v]$ if $\mathsf{eff}(a)[v]$ is specified. The resulting state is denoted by $s[\![a]\!]$; by $s[\![\langle a_1, \ldots, a_k\rangle]\!]$ we denote the state obtained from sequential application of the (respectively applicable) actions $a_1, \ldots, a_k$ starting at state s. Such an action sequence is an *s-plan* if $G \subseteq s[\![\langle a_1, \ldots, a_k\rangle]\!]$, and it is a *cost-optimal* (or, in what follows, *optimal*) s-plan if the sum of its action costs is minimal among all s-plans. The purpose of (optimal) planning is finding an (optimal) I-plan.

For a pair of states $s_1, s_2 \in S$, by $c(s_1, s_2)$ we refer to the cost of a cheapest action sequence taking us from s_1 to s_2 in the transition system induced by Π; $h^*(s) = \min_{s' \supseteq G} c(s, s')$ is the custom notation for the cost of the optimal s-plan in Π. We also use two well-known graphical structures induced by a planning task Π with variables V.

- The *causal graph* $CG(\Pi)$ of Π is a digraph over nodes V. An arc (v, v') is in $CG(\Pi)$ iff $v \neq v'$ and there exists an action $a \in A$ such that both $\mathsf{eff}(a)[v']$ and either $\mathsf{pre}(a)[v]$ or $\mathsf{eff}(a)[v]$ are specified.
- The *domain transition graph* $DTG(v, \Pi)$ of $v \in V$ is an arc-labeled digraph over the nodes $\mathcal{D}(v)$ such that an arc (ϑ, ϑ') labeled with $a \in A$ belongs to $DTG(v, \Pi)$ iff both $\mathsf{eff}(a)[v] = \vartheta'$ and either $\mathsf{pre}(a)[v] = \vartheta$ or $\mathsf{pre}(a)[v]$ is unspecified.

Our focus here is on *state-dependent, admissible abstraction heuristics*. A heuristic is *state-dependent* if its estimate for a search node depends only on the problem state associated with that node, that is, $h : S \to \mathbb{R}_0^+ \cup \{\infty\}$. Most heuristics in use these days are state-dependent (though see, e.g., Richter et al. (2008) for a different case). A state-dependent heuristic h is *admissible* if $h(s) \leq h^*(s)$ for all states s. Finally, an *abstraction* heuristic is based on mapping Π's transition system over states S to an *abstract transition system* over states S^α. The mapping is defined by an abstraction function $\alpha : S \to S^\alpha$ that guarantees $c_\alpha(\alpha(s), \alpha(s')) \leq c(s, s')$ for all states $s, s' \in S$. The (admissible) abstraction heuristic $h^\alpha(s)$ is then the distance from $\alpha(s)$ to the closest abstract goal state.

Heuristic Complexity

Accuracy and low time complexity are both desired yet competing properties of heuristic functions. For many powerful heuristics, and in particular abstraction heuristics, computing $h(s)$ for each state s in isolation is infeasible—while

computing $h(s)$ is polynomial in the description size of Π, it is often not efficient enough to be performed at each search node. Fortunately, for some heuristics this obstacle can be overcome to a large extent by sharing most of the computation between the evaluations of h on different states. If that is possible, the shared parts of computing $h(s)$ for *all* problem states s are pre-computed and memorized before the search, and then reused during the search by the evaluations of h on different states. This mixed offline/online heuristic computation, called henceforth h-*partition*, is adopted by various optimal planners using backward critical-path heuristics h^m for $m > 1$ (Haslum, Bonet, and Geffner 2005), landmark heuristics h^L and h^{LA} (Karpas and Domshlak 2009), and abstraction pattern-database and merge-and-shrink heuristics (Edelkamp 2001; Helmert, Haslum, and Hoffmann 2007). Without an effective h-partition, optimal search with these heuristics would not scale up well, while with such an h-partition these heuristics constitute the state of the art of heuristic search planning.

We define the time complexity of an h-partition as the complexity of computing h for a *set* of states. Given a subset of k problem states $S' \subseteq S$, the h-partition's time complexity of computing $\{h(s) \mid s \in S'\}$ is expressed as $O(X + kY)$, where $O(X)$ and $O(Y)$ are, respectively, the complexity of the (offline) pre-search and (online) per-node parts of computing $h(s)$ for a state $s \in S'$.

Abstraction Heuristics and h-Partitions

As we briefly mentioned above, an abstraction heuristic for a problem Π maps Π's transition system over states S to an abstract transition system over states S^α. The mapping is defined by an abstraction function $\alpha : S \to S^\alpha$ that guarantees $c_\alpha(\alpha(s), \alpha(s')) \leq c(s, s')$ for all states $s, s' \in S$. Such "distance conservancy" is in particular guaranteed by homomorphism abstractions, obtained (only) by systematically contracting groups of states into abstract states. The abstraction heuristic $h^\alpha(s)$ is the distance from $\alpha(s)$ to the closest abstract goal state, and admissibility of h^α is implied by the distance conservancy of α.

The most well-studied abstraction heuristics are *pattern database (PDB)* heuristics. Given a planning task Π over state variables V, such a heuristic is based on projecting Π onto a subset of its variables $V^\alpha \subseteq V$. Such a homomorphism abstraction α maps two states $s_1, s_2 \in S$ into the same abstract state iff $s_1[V^\alpha] = s_2[V^\alpha]$. Inspired by the (similarly named) domain-specific heuristics for search problems such as (k^2-1)-puzzles, Rubik's Cube, etc. (Culberson and Schaeffer 1998; Hernadvölgyi and Holte 1999; Felner, Korf, and Hanan 2004), PDB heuristics have been successfully exploited in domain-independent planning (Edelkamp 2001; 2002; Haslum et al. 2007).

Apart from the need to automatically select good projections, the two limitations of PDB heuristics are the size of the abstract space and its dimensionality. First, the number of abstract states should be small enough to allow reachability analysis in S^α by exhaustive search, and thus we must have $|S^\alpha| = O(1)$. The bound on $|S^\alpha|$ is typically set explicitly given the time and memory limitations of the system. Second, since PDB abstractions are projections, the explicit constraint on $|S^\alpha|$ implies a fixed-dimensionality constraint on the abstract space, that is, $|V^\alpha| = O(1)$. In problems with, informally, many alternative resources, this limitation is a pitfall. For instance, suppose $\{\Pi_i\}$ is a sequence of Logistics problems of growing size with $|V_i| = i$. If each package in Π_i can be transported by some $\Theta(i)$ vehicles, then starting from some j, h^α will not account *at all* for movements of vehicles essential for solving Π_j (Helmert and Mattmüller 2007).

Note that, while the first limitation of projection abstractions can be overcome to a certain extent with additive ensembles of numerous projections, the dimensionality limitation remains an issue in additive PDBs as well. Within these limitations, however, a very attractive side of PDB abstractions is the complexity of their natural h^α-partition. Instead of computing $h^\alpha(s) = h^*(\alpha(s))$ from scratch for each evaluated state s (which would not be practical for all but tiny projections), the practice is to pre-compute and store $h^*(\alpha(s))$ for *all* abstract states $\alpha(s) \in S^\alpha$, and then the per-node computation of $h^\alpha(s)$ boils down to hash-table lookup with a perfect hash function. In our terms, the time and space complexity of that PDB h^α-partition for a set of k states is $O\left(|S^\alpha|\left(\log\left(|S^\alpha|\right) + |A|\right) + k\right)$ and $O(|S^\alpha|)$, respectively. This is precisely what makes PDB heuristics so attractive in practice.

Aiming at preserving the attractiveness of the PDB h^α-partition while eliminating the bottleneck of fixed dimensionality, Helmert, Haslum, and Hoffmann (2007) generalize the methodology of Dräger et al. (2006) and introduce what we call *merge-and-shrink (MS)* planning abstractions. MS abstractions are homomorphisms that generalize PDB abstractions by allowing for more flexibility in selection of pairs of state to be contracted. The problem's state space is viewed as the synchronized product of its projections onto the single state variables. This product can be computed by iteratively composing two abstract spaces, replacing them with their product. While in a PDB the size of the abstract space S^α is controlled by limiting the number of product compositions, in MS abstractions it is controlled by interleaving the iterative composition of projections with abstraction of the partial composites.

The accuracy of the MS heuristics crucially depends on the order in which composites are formed and the choice of abstract states to contract. This flexibility in the MS abstraction strategy also adds variability to the complexity of its natural h^α-partition. The time and space complexity for the linear abstraction strategy of Helmert et al. are respectively $O\left(|V||S^\alpha|\left(\log\left(|S^\alpha|\right) + |A|\right) + k \cdot |V|\right)$ and $O(|S^\alpha|)$. Similarly to PDB abstractions, the per-node computation of $h^\alpha(s)$ with an MS abstraction α is just a lookup in a data structure storing $h^*(\alpha(s))$ for all abstract states $\alpha(s) \in S^\alpha$. Hence, while the pre-search computation with MS abstractions can be more costly than with PDBs, the online part of computing heuristic values is still extremely efficient. This per-node efficiency allows Helmert et al. to demonstrate impressive practical effectiveness of MS abstractions on several IPC domains (Helmert, Haslum, and Hoffmann 2007).

Merge-and-shrink abstractions escape the fixed-

dimensionality constraint of PDBs, but not the constraint on the abstract space to be of a fixed size. In attempt to eliminate *both* these constraints of "explicit" abstractions, Katz and Domshlak (2008b) introduce a framework of *structural-pattern* abstractions. A structural-pattern abstraction α maps the problem in hand to an abstract problem from a tractable fragment of optimal planning. This way, the dimensionality of S^α is not limited even by $|V|$, and the size of the abstract space is not limited even by the size of the original space.[2] In particular, Katz and Domshlak (2008b) introduce a concrete family of structural-pattern abstraction, *fork-decomposition*, corresponding to two classes of tractable optimal planning. A digraph forms *fork/inverted-fork* if it is connected, loop-less, and have all its arcs outgoing/incoming from/to a single node (called the root node of the graph). Propositions 3–4 of Katz and Domshlak (2008b) state that an optimal plan for a planning task Π can be found in polynomial time if

1. the causal graph of Π forms a fork, and the root variable is binary-valued, or
2. the causal graph of Π forms an inverted fork[3], and the domain of the root variable is fixed.

Now, given a general planning task Π, for each variable $v_i \in V$, let $V_i^{\mathrm{f}} \subseteq V$ contain v_i and all its immediate successors in $CG(\Pi)$, and $V_i^{\mathrm{i}} \subseteq V$ contain v_i and all its immediate predecessors in $CG(\Pi)$. At a high-level summary, the fork-decomposition of Π with $|V| = n$ is obtained by

(1) schematically constructing a set of projection abstractions $\{\Pi_i^{\mathrm{f}}, \Pi_i^{\mathrm{i}}\}_{i=1}^n$,

(2) reformulating the actions of $\{\Pi_i^{\mathrm{f}}, \Pi_i^{\mathrm{i}}\}_{i=1}^n$ to single-effect actions so that the causal graphs of Π_i^{f} and Π_i^{i} become respectively forks and inverted forks rooted in v_i, and

(3) within each Π_i^{f}, abstracting the domain of v_i to $\{0, 1\}$, and within each Π_i^{i}, abstracting the domain of v_i to $\{0, 1, \ldots, b\}$ with $b = O(1)$.

The problems $\{\Pi_i^{\mathrm{f}}, \Pi_i^{\mathrm{i}}\}_{i=1}^n$ correspond to distance-conservative (though, after step (2), not necessarily homomorphic) abstractions of Π, and, similarly to PDB and MS abstractions, can be used either separately or as parts of heuristic ensembles.

To see that the fork-decomposition escapes both limitations of the projection abstractions, note that both $|V_i^{\mathrm{f}}|$ and $|V_i^{\mathrm{i}}|$ can be $\Theta(|V|)$, and thus each induced $|S^\alpha|$ can be $\Theta(|S|)$. The latter is a blessing, but a mixed one, and both sides of the story are demonstrated in Table 1, showing the performance of four different planners on the planning tasks from the Logistics domain of IPC-2000. The first planner, $h^{\mathcal{F}}$, is A^* equipped with the additive fork-decomposition heuristic "all forks" under uniform action cost partitioning (Katz and Domshlak 2008b). It is compared to (1) *MS*-10^5—Helmert et al.'s A^* with a merge-and-shrink abstraction constrained by $|S^\alpha| \leq 10^5$, (2) $\mathrm{HSP}_{\mathrm{F}}^*$ heuristic-search planner, and (3) Gamer planner. *MS*-10^5 represents the state-of-the-art in optimal planning with abstraction heuristics, while Gamer and $\mathrm{HSP}_{\mathrm{F}}^*$ were the two best-performing cost-optimal planners at IPC-2008.

		$h^{\mathcal{F}}$		*MS*-10^5		$\mathrm{HSP}_{\mathrm{F}}^*$		Gamer
task	h^*	*nodes*	*time*	*nodes*	*time*	*nodes*	*time*	*time*
04-0	20	21	0	21	0	21	0.3	0.3
04-1	19	20	0	20	0	20	0.4	0.3
04-2	15	16	0	16	0	16	0.4	0.3
05-0	27	28	0	28	0.4	28	0.6	0.4
05-1	17	18	0	18	0.4	18	0.7	0
05-2	8	9	0	9	0.4	9	0.8	0.3
06-0	25	26	0	26	1.2	26	1.0	0.4
06-1	14	15	0	15	1.3	15	1.2	0.3
06-2	25	26	0	26	1.3	26	1.0	0.4
06-9	24	25	0	25	1.2	25	1.0	0.4
07-0	36	37	3.9	37	4.9	24317	35.5	6.4
07-1	44	1689	93.7	49	4.9	362179	453.1	16.9
08-0	31	32	3.4	32	6.9	14890	33.5	3.9
08-1	44	45	5.4	45	7.2	114155	198.8	10.0
09-0	36	37	3.9	37	9.5	32017	83.2	5.8
09-1	30	31	3.7	31	9.4	6720	26.5	2.9
10-0	45	46	13.2	668834	29.7	599645	3376	610.0
10-1	42	43	12.2	1457130	43			478.7
11-0	48	697	214.3	701106	37.4			847.2
11-1	60	21959	5561.0					
12-0	42	43	20.3	775996	43.6			444.5
12-1	68	106534	25241.5	2222340	87.5			

Table 1: The results for A^* on Logistics-2000 domain with the additive fork-decomposition heuristic "all forks" $h^{\mathcal{F}}$ under uniform action cost partitioning. Comparison with A^* and merge-and-shrink abstraction with $|S^\alpha| \leq 10^5$, $\mathrm{HSP}_{\mathrm{F}}^*$ heuristic-search planner, and Gamer planner. h^*, *nodes*, and *time* are the length (= cost) of the optimal plan, number of expanded nodes and total time in seconds; for Gamer the notion of expanded nodes is irrelevant.

As it appears from Table 1, at least on the Logistics domain, the abstraction-based heuristic search favorably competes with the leading optimal-search planners: Both $h^{\mathcal{F}}$ and *MS*-10^5 outperform $\mathrm{HSP}_{\mathrm{F}}^*$ and Gamer on these planning tasks. The comparison between $h^{\mathcal{F}}$ and *MS*-10^5, however, is more important to us here. On the one hand, $h^{\mathcal{F}}$ almost consistently expands less search nodes than *MS*-10^5, with the difference hitting four orders of magnitude. On the other hand, the time complexity of $h^{\mathcal{F}}$ per search node is substantially higher than this of *MS*-10^5, with the two expanding (approximately) 4 and 100000 nodes per second, respectively. The outcome is simple: while with no time limits (and only memory limit of 1.5 GB) $h^{\mathcal{F}}$ solves more tasks than *MS*-10^5, the inverse holds with a time limit of one hour (see tasks 11-1 and 12-1).

Structural-Pattern Databases

Empirical (in)effectiveness of the fork-decomposition heuristics was not surprising, yet it was disappointing to see an informative heuristic being out of reach of practical planning. Fortunately, this is not the end of the story. We now show that the time-per-node complexity bottleneck of fork-decomposition heuristics can be successfully overcome. Specifically, we show that an equivalent of PDB's and merge-and-shrink notion of "database" exists for fork-decomposition abstractions as well, despite of

[2] While dimensionality larger than $|V|$, and abstract-space size larger than $|S|$ may appear excessive, see (Katz and Domshlak 2008a) for concrete examples of why this can be useful.

[3] Proposition 4 of Katz and Domshlak (2008b) also requires the problem to be what is called 1-dependent, but this requirement was later found unnecessary.

their exponential-size abstract spaces. Of course, unlike with PDB and merge-and-shrink abstractions, *structural-pattern databases* (and in particular these for the fork-decomposition's fork and inverted fork abstractions) do not (and cannot) result in purely lookup computations of $h^\alpha(s)$. The online part of the h^α-partition has to be non-trivial in the sense that its complexity cannot be $O(1)$.

In the remainder of this section we prove existence of such effective h-partitions for both fork and inverted fork abstractions. In the next section we empirically show that these h-partitions lead to fast pre-search and per-node computations that allow for successfully exploiting the informativeness of the fork-decomposition heuristics in practice.

Theorem 1 *Let $\Pi = \langle V, A, I, G, c\rangle$ be a planning task with a fork causal graph rooted in a binary-valued variable r. There exists an h^*-partition for Π such that, for any set of k states, the time and space complexity of that h^*-partition is, respectively, $O(d^3|V| + |A_r| + k \cdot d|V|)$ and $O(d^2|V|)$, where $d = \max_v \mathcal{D}(v)$.*

The proof of Theorem 1 is based on a modification of the poly-time algorithm of Katz and Domshlak (2008b) for computing $h^*(s)$ for state s of such a task Π. Let $\mathcal{D}(r) = \{0, 1\}$, where $s[r] = 0$; for $\vartheta \in \mathcal{D}(r)$, let $\neg\vartheta$ denote the opposite value $1 - \vartheta$. Let $\sigma(r)$ be a 0/1 sequence of length $1 + d$, such that, for $1 \le i \le |\sigma(r)|$, $\sigma(r)[i] = 0/1$ if i is odd/even. Let $\trianglerighteq^*[\sigma(r)]$ be the set of all non-empty prefixes of $\sigma(r)$ if $G[r]$ is unspecified, and the set of all non-empty prefixes of $\sigma(r)$ ending with $G[r]$, otherwise. For each $\sigma \in \trianglerighteq^*[\sigma(r)]$, $c(\sigma)$ denotes the (straightforwardly computable) cost of achieving the respective sequence of value changes of r.

For each $v \in V \setminus \{r\}$, let DTG_v^0 and DTG_v^1 be the subgraphs of $DTG(v, \Pi)$ obtained by removing from the latter all the arcs labeled with 1 and 0, respectively.

1. For each $v \in V \setminus \{r\}$, and each $\vartheta, \vartheta' \in \mathcal{D}(v)$, compute the shortest (i.e., cost-minimal) paths from ϑ to ϑ' in DTG_v^0 and DTG_v^1.
2. For each $\sigma \in \trianglerighteq^*[\sigma(r)]$, and each $v \in V \setminus \{r\}$, build a layered digraph $\mathcal{L}_v(\sigma)$ with $|\sigma| + 1$ layers $L_0, \ldots, L_{|\sigma|}$, where L_0 consists of only $s[v]$, and for $1 \le i \le |\sigma|$, L_i consists of all nodes reachable from the nodes L_{i-1} in DTG_v^0/DTG_v^1 if i is odd/even. For each $\vartheta \in L_{i-1}, \vartheta' \in L_i$, $\mathcal{L}_v(\sigma)$ contains an arc (ϑ, ϑ') weighted with the distance from ϑ to ϑ' in DTG_v^0/DTG_v^1 if i is odd/even.
3. Set $\mathcal{R} = \emptyset$. For each $\sigma \in \trianglerighteq^*[\sigma(r)]$, a plan ρ_σ for Π is constructed as follows.
   i For each $v \in V \setminus \{r\}$, find a shortest path from $s[v]$ to $G[v]$ in $\mathcal{L}_v(\sigma)$. If no such path exists, go to the next $\sigma \in \trianglerighteq^*[\sigma(r)]$. The i-th edge on this path (say, from $\vartheta \in L_{i-1}$ to $\vartheta' \in L_i$) corresponds to the shortest path from ϑ to ϑ' in either DTG_v^0 or DTG_v^1, and we denote this path by S^i_ϑ.
   ii Set $\mathcal{R} = \mathcal{R} \cup \{\rho_\sigma\}$, $\rho_\sigma = S^1 \cdot a_{\sigma[2]} \cdot S^2 \cdot \ldots \cdot a_{\sigma[m]} \cdot S^m$, where $m = |\sigma|$, S^i is obtained by an arbitrary merge of $\{S^i_v\}_{v \in V \setminus \{r\}}$, and a_x is the action that changes the value of r to x.
4. If $\mathcal{R} = \emptyset$, then "unsolvable". Otherwise, return $\rho = \mathrm{argmin}_{\rho_\sigma \in \mathcal{R}}\, c(\rho_\sigma)$.

This algorithm can be formulated as follows.

(1) For each $\vartheta_r \in \mathcal{D}(r)$, $v \in V \setminus \{r\}$, and $\vartheta, \vartheta' \in \mathcal{D}(v)$, let $p_{\vartheta,\vartheta';\vartheta_r}$ be the cost of the cheapest sequence of actions ϑ to ϑ' *provided* $r = \vartheta_r$. The whole set $\{p_{\vartheta,\vartheta';\vartheta_r}\}$ can be computed by a straightforward variant of the all-pairs-shortest-paths, Floyd-Warshall algorithm in time $O(d^3|V|)$.

(2) For each $v \in V \setminus \{r\}$, $1 \le i \le d + 1$, and $\vartheta \in \mathcal{D}(v)$, let $g_{\vartheta;i}$ be the cost of the cheapest sequence of actions changing $s[v]$ to ϑ *provided* a sequence $\sigma \in \trianglerighteq^*[\sigma(r)]$, $|\sigma| = i$, of value changes of r. Given $\{p_{\vartheta,\vartheta';\vartheta_r}\}$, the set $\{g_{\vartheta;i}\}$ is given by the solution of the recursive equation

$$g_{\vartheta;i} = \begin{cases} p_{s[v],\vartheta;s[r]}, & i = 1 \\ \min\limits_{\vartheta'} g_{\vartheta';i-1} + p_{\vartheta',\vartheta;s[r]}, & 1 < i \le \delta_\vartheta,\ i \text{ is odd} \\ \min\limits_{\vartheta'} g_{\vartheta';i-1} + p_{\vartheta',\vartheta;\neg s[r]}, & 1 < i \le \delta_\vartheta,\ i \text{ is even} \\ g_{\vartheta;i-1}, & \delta_\vartheta < i \le d + 1 \end{cases},$$

where $\delta_\vartheta = |\mathcal{D}(v)| + 1$ if $\vartheta \in \mathcal{D}(v)$. Given that,

$$h^*(s) = \min_{\sigma \in \trianglerighteq^*[\sigma(r)]} [\, c(\sigma) + \sum_{v \in V \setminus \{r\}} g_{G[v];|\sigma|} \,].$$

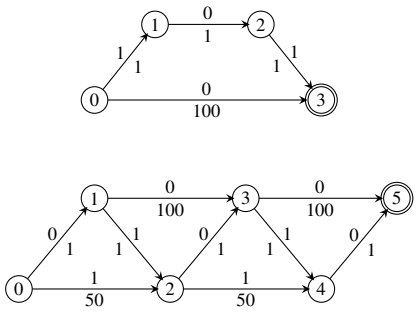
Notice that step (1) is already state-independent, but the heavy step (2) is not. However, the state dependence of step (2) can mostly be overcome. For each $v \in V \setminus \{r\}$, $\vartheta \in \mathcal{D}(v)$, $1 \le i \le d + 1$, and $\vartheta_r \in \mathcal{D}(r)$, let $g_{\vartheta;i}(\vartheta_r)$ be the cost of the cheapest sequence of actions changing ϑ to $G[v]$ *provided* the value changes of r induce a $0/1$ sequence of length i starting with ϑ_r. Very similarly to before, the set $\{g_{\vartheta;i}(\vartheta_r)\}$ is given by the solution of the recursive equation

$$g_{\vartheta;i}(\vartheta_r) = \begin{cases} p_{\vartheta,G[v];\vartheta_r}, & i = 1 \\ \min\limits_{\vartheta'} g_{\vartheta';i-1}(\neg\vartheta_r) + p_{\vartheta,\vartheta';\vartheta_r}, & 1 < i \le \delta_\vartheta \\ g_{\vartheta;i-1}(\vartheta_r), & \delta_\vartheta < i \le d + 1 \end{cases}, \quad (1)$$

which can be solved in time $O(d^3|V|)$. Note that this equation is now independent of the evaluated state s, and yet $\{g_{\vartheta;i}(\vartheta_r)\}$ allow for computing $h^*(s)$ for a given state s via

$$h^*(s) = \min_{\sigma \in \trianglerighteq^*[\sigma(r)]} [\, c(\sigma) + \sum_{v \in V \setminus \{r\}} g_{s[v];|\sigma|}(s[r]) \,] \quad (2)$$

With the new formulation, the only computation that has to be performed per search node is this of the final minimization over $\trianglerighteq^*[\sigma(r)]$, which is the lightest part of the whole algorithm anyway. The major computation, notably this of $\{p_{\vartheta,\vartheta';\vartheta_r}\}$ and $\{g_{\vartheta;i}(\vartheta_r)\}$, can now be performed offline, and shared between the evaluated states. The space required to store this information is $O(d^2|V|)$ as it contains only a fixed amount of information per each pair of values of a same variable. The time complexity of the offline computation is $O(d^3|V| + |A_r|)$; the $|A_r|$ component stems from



(a)

r	$\lvert\sigma\rvert$	$c(\sigma)$	$v=0$	$v=1$	$v=2$	$v=3$	$u=0$	$u=1$	$u=2$	$u=3$	$u=4$	$u=5$
0	1	0	100	∞	∞	0	201	200	101	100	1	0
	2	24	100	2	1	0	201	200	101	100	1	0
	3	48	100	2	1	0	53	102	3	2	1	0
	4	72	100	2	1	0	53	102	3	2	1	0
	5	96	3	2	1	0	5	4	3	2	1	0
	6	120	3	2	1	0	5	4	3	2	1	0
	7	144	3	2	1	0	5	4	3	2	1	0
1	1	0	∞	∞	1	0	∞	∞	∞	∞	∞	0
	2	24	100	∞	1	0	101	52	51	2	1	0
	3	48	3	2	1	0	101	52	51	2	1	0
	4	72	3	2	1	0	53	4	3	2	1	0
	5	96	3	2	1	0	53	4	3	2	1	0
	6	120	3	2	1	0	5	4	3	2	1	0
	7	144	3	2	1	0	5	4	3	2	1	0

(b)

Figure 1: Structural-pattern database for a fork-structured problem with a binary-valued root variable r and two children v and u, and $G[r] = 0$, $G[v] = 3$, and $G[u] = 5$. (a) depicts the domain transition graphs of v (top) and u (bottom); the numbers above and below each edge are the precondition on r and the cost of the respective action. (b) depicts the database created by the algorithm. For instance, the entry in row $r=0 \wedge |\sigma|=5$ and column $v = 0$ captures the value of $g_{v=0;5}(r=0)$ computed as in Eq. 1. The shaded entries are these examined at the online computation of $h^*(r=0, v=0, u=0)$.

pre-computing the pair of values w_0 and w_1. The time complexity of the online computation per state is $O(d|V|)$; $|V|$ comes from the internal summation and d comes from the size of $\trianglerighteq^*[\sigma(r)]$.

Figure 1(b) shows the structural-pattern database created for a fork-structured problem with a binary-valued root r, two children v and u, and $G[r] = 0$, $G[v] = 3$, and $G[u] = 5$; the domain transition graphs of v and u are depicted in Figure 1(a). Online computation of $h^*(s)$ as in Eq. 2 for $s = (r = 0, v = 0, u = 0)$ sums over the shaded entries of each of the four rows having such entries, and minimizes over the resulting four sums, with the minimum being obtained in the row $r=0 \wedge |\sigma|=5$.

Theorem 2 *Let $\Pi = \langle V, A, I, G, c\rangle$ be a planning task with an inverted fork causal graph rooted in r with $|\mathcal{D}(r)| = b = O(1)$. There exists an h^*-partition for Π such that, for any set of k states, the time and space complexity of that h^*-partition is $O(b|V||A_r|^{b-1} + d^3|V| + k \cdot |V||A_r|^{b-1})$ and $O(|V||A_r|^{b-1} + d^2|V|)$, respectively, where $d = \max_v \mathcal{D}(v)$.*

Similarly to the proof of Theorem 1, the proof of Theorem 2 is also based on a modification of the poly-time algorithm of Katz and Domshlak (2008b) for computing $h^*(s)$ for state s of a task Π as in the theorem. The latter algorithm finds $h^*(s)$ as follows.

1. For each $v \in V \setminus \{r\}$, and each $\vartheta, \vartheta' \in \mathcal{D}(v)$, compute shortest (i.e., cost-minimal) paths from ϑ to ϑ' in $DTG(v, \Pi)$.
2. Enumerate all $\Theta(|A_r|^{b-1})$ cycle-free paths from $s[r]$ to $G[r]$ in $DTG(r, \Pi)$. For each such path, construct a plan for Π based on that path for r, and the shortest paths computed in (1). The cheapest such plan is a cost-optimal plan from s in Π, and thus induces $h^*(s)$.

This algorithm can be formulated as follows.

(1) For each $v \in V \setminus \{r\}$, and $\vartheta, \vartheta' \in \mathcal{D}(v)$, let $p_{\vartheta,\vartheta'}$ be the cost of the cheapest sequence of actions changing ϑ to ϑ'. The whole set $\{p_{\vartheta,\vartheta'}\}$ can be computed using the Floyd-Warshall algorithm in time $O(d^3|V|)$.

(2) For each cycle-free path $\pi = a_1 \cdot \ldots \cdot a_m$ from $s[r]$ to $G[r]$ in $DTG(v, \Pi)$, let g_π be the cost of the cheapest plan from s in Π based on π, and the shortest paths computed in (1). Each g_π can be computed as

$$g_\pi = \sum_{i=1}^{m} c(a_i) + \sum_{i=0}^{m} \sum_{v \in V \setminus \{r\}} p_{\mathsf{pre}_i[v], \mathsf{pre}_{i+1}[v]},$$

where $\mathsf{pre}_0 \cdot \ldots \cdot \mathsf{pre}_{m+1}$ are the values needed from the parents of r along the path π. That is, for each $v \in V \setminus \{r\}$, and $0 \le i \le m+1$,

$$\mathsf{pre}_i[v] = \begin{cases} s[v], & i = 0 \\ G[v], & i = m+1, \text{ and } G[v] \text{ is specified} \\ \mathsf{pre}(a_i)[v], & 1 \le i \le m, \text{ and } \mathsf{pre}(a_i)[v] \text{ is specified} \\ \mathsf{pre}_{i-1}[v] & \text{otherwise} \end{cases}$$

From that, we have $h^*(s) = \min_\pi g_\pi$.

Notice that step (1) is state-independent, but step (2) is not so. However, the dependence of step (2) on the evaluated state can be substantially relaxed. As there are only $O(1)$ different values of r, it is possible to consider cycle-free paths to $G[r]$ from *all* values of r. For each such path π, and each parent variable $v \in V \setminus \{r\}$, we know what would be the first value of v required by π. Given that, we can pre-compute the cost-optimal plans induced by each π and *assuming the parents start at their first needed values.* The remainder of the computation of $h^*(s)$ is delegated to online, and the modified step (2) is as follows.

For each $\vartheta_r \in \mathcal{D}(r)$ and each cycle-free path $\pi = a_1 \cdot \ldots \cdot a_m$ from ϑ_r to $G[r]$ in $DTG(v, \Pi)$, let a "proxy" state s_π be

$$s_\pi[v] = \begin{cases} \vartheta_r, & v = r \\ G[v], & \forall 1 \le i \le m : \mathsf{pre}(a_i)[v] \text{ is unspecified} \\ \mathsf{pre}(a_i)[v], & i = \operatorname{argmin}_j \{\mathsf{pre}(a_j)[v] \text{ is specified}\} \end{cases},$$

domain (D)	$S(D)$	$h^{\mathcal{F}}$		$h^{\mathcal{J}}$		$h^{\mathcal{FJ}}$		*MS*-10^4		*MS*-10^5		HSP^*_{F}		Gamer		blind		$h_{\max}$	
		s	$\%S$	s	$\%S$	s	$\%S$	s	$\%S$	s	$\%S$	s	$\%S$	s	$\%S$	s	$\%S$	s	$\%S$
airport-ipc4	20	**20**	100	17	85	17	85	16	80	16	80	15	75	11	55	17	85	**20**	100
blocks-ipc2	30	21	70	18	60	18	60	18	60	20	67	**30**	100	**30**	100	18	60	18	60
depots-ipc3	7	**7**	100	4	57	6	86	**7**	100	4	57	4	57	4	57	4	57	4	57
driverlog-ipc3	12	**12**	100	**12**	100	**12**	100	**12**	100	**12**	100	9	75	11	92	7	58	8	67
freecell-ipc3	5	**5**	100	4	80	4	80	**5**	100	1	20	**5**	100	2	40	4	80	**5**	100
grid-ipc1	2	**2**	100	1	50	1	50	**2**	100	**2**	100	0	0	**2**	100	1	50	**2**	100
gripper-ipc1	20	7	35	7	35	7	35	7	35	7	35	6	30	**20**	100	7	35	7	35
logistics-ipc1	7	6	86	4	57	5	71	4	57	5	71	3	43	6	86	2	29	2	29
logistics-ipc2	22	**22**	100	16	73	16	73	16	73	21	95	16	73	20	91	10	45	10	45
miconic-strips-ipc2	85	51	60	50	59	50	59	54	64	55	65	45	53	**85**	100	50	59	50	59
mprime-ipc1	24	23	96	19	79	21	88	21	88	12	50	8	33	9	38	19	79	**24**	100
mystery-ipc1	20	**20**	100	16	80	**20**	100	16	80	12	60	11	55	8	40	17	85	17	85
openstacks-ipc5	7	**7**	100	**7**	100	**7**	100	**7**	100	**7**	100	**7**	100	**7**	100	**7**	100	**7**	100
pathways-ipc5	4	**4**	100	**4**	100	**4**	100	3	75	**4**	100	**4**	100	**4**	100	**4**	100	**4**	100
pipesworld-notankage-ipc4	21	16	76	15	71	16	76	**20**	95	12	57	13	62	11	52	14	67	17	81
pipesworld-tankage-ipc4	14	10	71	9	64	9	64	**13**	93	7	50	7	50	6	43	10	71	10	71
psr-small-ipc4	50	49	98	49	98	49	98	**50**	100	**50**	100	**50**	100	47	94	48	96	49	98
rovers-ipc5	7	6	86	**7**	100	6	86	6	86	**7**	100	6	86	5	71	5	71	6	86
satellite-ipc4	6	**6**	100	**6**	100	**6**	100	**6**	100	**6**	100	5	83	**6**	100	4	67	5	83
schedule-strips	46	**46**	100	32	70	**46**	100	22	48	1	2	11	24	3	7	29	63	31	67
tpp-ipc5	6	**6**	100	**6**	100	**6**	100	**6**	100	**6**	100	5	83	5	83	5	83	**6**	100
trucks-ipc5	9	6	67	7	78	7	78	6	67	5	56	**9**	100	3	33	5	56	7	78
zenotravel-ipc3	11	**11**	100	**11**	100	**11**	100	**11**	100	**11**	100	8	73	10	91	7	64	8	73
$s(p)$	435	**363**		321		344		328		283		277		315		294		317	
$\hat{s}(p)$		**20.45**		17.96		18.88		18.99		16.65		15.55		16.73		15.6		17.74	
$w(p)$		**14**		7		8		11		9		6		8		2		7	

Table 2: A summary of the experimental results. Per domain, S denotes the number of tasks solved by *any* planner. Per planner/domain, the number of tasks solved by that planner is given both by the absolute number (s) and by the percentage from "solved by any" ($\%S$). Boldfaced results indicate the best performance within the corresponding domain. The last three rows summarize the number of solved instances in total (s), the domain-normalized measure of solved instances in total ($\hat{s}$), and the number of domains in which the planners achieved superior performance (w).

that is, the non-trivial part of s_π captures the first values of $V \setminus \{r\}$ needed along π.[4] Given that, let g_π be the cost of the cheapest plan from s_π in Π based on π, and the shortest paths $\{p_{\vartheta,\vartheta'}\}$ computed in (1). Each g_π can be computed as

$$g_\pi = \sum_{i=1}^{m} \left[c(a_i) + \sum_{v \in V \setminus \{r\}} p_{\mathsf{pre}_i[v], \mathsf{pre}_{i+1}[v]} \right],$$

where, for each $v \in V \setminus \{r\}$, and $1 \leq i \leq m+1$,

$$\mathsf{pre}_i[v] = \begin{cases} s_\pi[v], & i = 1 \\ G[v], & i = m+1, \text{ and } G[v] \text{ is specified} \\ \mathsf{pre}(a_i)[v], & 2 \leq i \leq m, \text{ and } \mathsf{pre}(a_i)[v] \text{ is specified} \\ \mathsf{pre}_{i-1}[v] & \text{otherwise} \end{cases}$$

Storing the pairs (g_π, s_π) accomplishes the offline part of the computation. Now, given a search state s, we can compute

$$h^*(s) = \min_{\pi : s_\pi[r] = s[r]} \left[g_\pi + \sum_{v \in V \setminus \{r\}} p_{s[v], s_\pi[v]} \right].$$

The dominant offline computation is computing $\{g_\pi\}$. The number of cycle-free paths to $G[r]$ in $DTG(v, \Pi)$ is $\Theta(|A_r|^{b-1})$, and computing g_π for each such path π can be done in time $O(b|V|)$. Hence, the overall offline time complexity is $O(b|V||A_r|^{b-1})$, and space complexity (including the storage of the proxy states s_π) is $O(|V||A_r|^{b-1})$. The time complexity of the online computation per state is $O(|V||A_r|^{b-1})$; $|V|$ comes from the internal summation and $|A_r|^{b-1}$ from the upper bound on the number of cycle-free paths from $s[r]$ to $G[r]$.

[4] For ease of presentation, we omit here the case where v is needed neither along π, nor by the goal; such variables should be simply ignored when accounting for the cost of π.

Experimental Evaluation

To evaluate the practical attractiveness of structural-pattern heuristics in general, and the corresponding h-partitions in particular, we have conducted an empirical study on a wide sample of planning domains from the international planning competitions 1998-2006. The selection of the domains was aimed at allowing a comparative evaluation with other baseline and state of the art approaches/planners that, at the moment, not all support all PDDL features.

We implemented three additive fork-decomposition heuristics within a standard heuristic forward search framework of the Fast Downward planner (Helmert 2006), using the A^* algorithm with full duplicate elimination. The three heuristics were these outlined by Katz and Domshlak (2008b). The $h^{\mathcal{F}}$ heuristic corresponds to the ensemble of all (not clearly redundant) fork subgraphs of the causal graph, with the domains of the roots being abstracted using the "leave-one-value-out" binary-valued domain decompositions. The $h^{\mathcal{J}}$ heuristic is the same but for the inverted fork subgraphs, with the domains of the roots being abstracted using the "distance-to-goal-value" ternary-valued domain decompositions. The ensemble of the $h^{\mathcal{FJ}}$ heuristic is the union of these for $h^{\mathcal{F}}$ and $h^{\mathcal{J}}$. The action cost partitioning for all three ensembles was set to "uniform".

We compare with two baseline approaches, namely blind search (A^* with a heuristic function which is 0 for goal states and 1 otherwise) and A^* with the $h_{\max}$ heuristic (Bonet and Geffner 2001), as well as with state of the art abstraction heuristics, represented by the merge-and-shrink abstractions of Helmert, Haslum, and Hoffmann (2007). The latter were constructed under the linear, f-preserving abstraction strategy suggested by these authors, and this under two fixed bounds on the size of the abstraction, $|S^\alpha| < 10^4$

and $|S^\alpha| < 10^5$. These four (baseline and MS) heuristics were implemented by Helmert *et al.* (2007) within the same planning system as our fork-decomposition heuristics, allowing for a fairly unbiased comparison. We also compare to the Gamer (Edelkamp and Kissmann 2009) and HSP^*_F (Haslum 2008) planners that were respectively the winner and the runner-up at the sequential optimization track of IPC-2008. On the algorithmic side, Gamer is based on a bidirectional blind search using sophisticated symbolic-search techniques, and HSP^*_F uses A^* with an additive critical-path heuristic.

The summary of our experimental results is shown in Table 2. The experiments were conducted on 3GHz Intel E8400 CPU with 2 GB memory, using a 1.5 GB memory limit and 30 minute timeout. The only exception from that was Gamer for which we used similar machines but with 4 GB memory, and 2 GB memory limit; this was done to provide Gamer with the environment for which it was configured. For each domain D, $S(D)$ is the number of tasks in D that were solved by at least one planner in the suite. For each domain D and planner p, $s(D, p)$ (in columns s) is the number of tasks in D solved by p. Columns $\%S$ provide the same information in terms of percentage from "solved by any". Boldfaced results indicate the best performance within the corresponding domain. The last three rows summarize the performance of the planners via three measures. First, $s(p)$ is the total number of tasks solved in all the 23 domains; this is basically the measure used in planner evaluation in IPC-2008. As domains are not equally challenging, and do not provide the same discrimination between the planners' performance, the second, "domain-normalized" measure of solved instances in total $\hat{s}(p) = \sum_D s(D, p)/S(D)$. Finally, $w(p)$ is the number of domains in which p achieved superior performance.

Overall, Table 2 clearly suggests that heuristic search with "databased" fork-decomposition heuristics favorably competes with the state-of-the-art of optimal planning. In particular, A^* with the fork-decomposition heuristic $h^{\mathcal{F}}$ exhibited the best overall performance according to all three measures. The overall performance of A^* with the other two heuristics $h^{\mathcal{J}}$ and $h^{\mathcal{FJ}}$ was not as good as with $h^{\mathcal{F}}$, and yet, in terms of, e.g., absolute number of solved instances, $h^{\mathcal{FJ}}$ and $h^{\mathcal{J}}$ came second and forth, respectively. Likewise, $h^{\mathcal{J}}$ still slightly outperforms $h^{\mathcal{F}}$ on the Rovers and Trucks domains, so no clear-cut dominance between them.

Finally, the contribution of employing h-partitions to the success of the structural-pattern heuristics was dramatic. Looking back on the results for Logistics-ipc2 in Table 1, note that the timeout of 30 minutes would prevent us from solving instances 11-1 and 12-1, ending up with only 20 solved instances. In contrast, with structural-pattern databases we manage to solve these two instances within the time limit, and the difference in running time is spectacular: While without employing h^α-partition, solving instance 12-1 took us more than 7 hours, with structural-pattern databases we solved all the 22 instances from Table 1 together in less than 35 seconds. This difference in runtimes is representative for all the domains in our evaluation, and thus h-partitions play a key role in bringing structural patterns to the state of the art in cost-optimal planning.

References

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

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *AIJ* 129(1–2):5–33.

Culberson, J., and Schaeffer, J. 1998. Pattern databases. *Comp. Intell.* 14(4):318–334.

Dräger, K.; Finkbeiner, B.; and Podelski, A. 2006. Directed model checking with distance-preserving abstractions. In *SPIN*, 1934.

Edelkamp, S., and Kissmann, P. 2009. Optimal symbolic planning with action costs and preferences. In *IJCAI*.

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

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

Felner, A.; Korf, R. E.; and Hanan, S. 2004. Additive pattern database heuristics. *JAIR* 22:279–318.

Haslum, P., and Geffner, H. 2000. Admissible heuristics for optimal planning. In *ICAPS*, 140–149.

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

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

Haslum, P. 2008. Additive and reversed relaxed reachability heuristics revisited. In *IPC*.

Helmert, M., and Mattmüller, R. 2007. Accuracy of admissible heuristic functions in selected planning domains. In *AAAI*, 938–943.

Helmert, M.; Haslum, P.; and Hoffmann, J. 2007. Flexible abstraction heuristics for optimal sequential planning. In *ICAPS*, 200–207.

Helmert, M. 2006. The Fast Downward planning system. *JAIR* 26:191–246.

Hernadvölgyi, I., and Holte, R. 1999. PSVN: A vector representation for production systems. Technical Report 1999-07, University of Ottawa.

Karpas, E., and Domshlak, C. 2009. Cost-optimal planning with landmarks. In *IJCAI*.

Katz, M., and Domshlak, C. 2008a. Optimal additive composition of abstraction-based admissible heuristics. In *ICAPS*, 174–181.

Katz, M., and Domshlak, C. 2008b. Structural patterns heuristics via fork decomposition. In *ICAPS*, 182–189.

Richter, S.; Helmert, M.; and Westphal, M. 2008. Landmarks revisited. In *AAAI*, 975–982.