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 $h_{\text{max}}$ (Bonet and Geffner 2001), critical path heuristics $h^m$ (Haslum and Geffner 2000), landmark heuristics $h^L$ and $h^{L-A}$ (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

¹This work of both authors is partly supported by Israel Science Foundation grant 670/07.

Copyright © 2009, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.
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 $\Pi = (V, A, I, G, c)$, where:

- $V$ is a set of state variables, each $v \in V$ is associated with a finite domain $D(v)$; 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 $S = D(V)$ is the state space of $\Pi$. $I$ is an initial state.
- $G$ is a partial assignment to $V$; a state $s$ is a goal state iff $G \subseteq s$.
- $A$ is a finite set of actions. Each action $a$ is a pair $(\text{pre} (a), \text{eff} (a))$ of partial assignments to $V$ called preconditions and effects, respectively. By $A_0 \subseteq A$ we denote the actions affecting the value of $v$. $c : A \rightarrow \mathbb{R}^+_0$ is a real-valued, non-negative action cost function.

An action $a$ is applicable in a state $s$ iff $s[v] = \text{pre} (a)[v]$ whenever $\text{pre} (a)[v]$ is specified. Applying $a$ changes the value of $v$ to $\text{eff} (a)[v]$ if $\text{eff} (a)[v]$ is specified. The resulting state is denoted by $s[a]$; by $s\{[a_1, \ldots, a_k]\}$ 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 $G \subseteq \{[a_1, \ldots, a_k]\}$, 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 $\Pi$; $h^*(s) = \min_{s', s'' \geq G} c(s, s')$ is the custom notation for the cost of the optimal $s$-plan in $\Pi$. We also use two well-known graphical structures induced by a planning task $\Pi$ with variables $V$.

- The causal graph $CG(\Pi)$ of $\Pi$ is a digraph over nodes $V$. An arc $(v, \nu)$ is in $CG(\Pi)$ iff $v \neq \nu$ and there exists an action $a \in A$ such that both $\text{eff} (a)[v']$ and either $\text{pre} (a)[v]$ or $\text{eff} (a)[v]$ are specified.

- The domain transition graph $DTG(\nu, \Pi)$ of $\nu \in V$ is an arc-labeled digraph over the nodes $D(\nu)$ such that an arc $(\emptyset, \emptyset)$ labeled with $a \in A$ belongs to $DTG(\nu, \Pi)$ iff both $\text{eff} (a)[v'] = \emptyset$ and either $\text{pre} (a)[v] = \emptyset$ or $\text{pre} (a)[v]$ 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, $h : S \rightarrow \mathbb{R}^+_0 \cup \{\infty\}$. 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 $\Pi$’s transition system over states $S$ to an abstract transition system over states $S^\alpha$. The mapping is defined by an abstraction function $\alpha : S \rightarrow S^\alpha$ 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 \( \Pi \), 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 memoized 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 \( h_L^A \) (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 \( \Pi \) maps \( \Pi \)'s transition system over states \( S \) to an abstract transition system over states \( S^\alpha \). The mapping is defined by an abstraction function \( \alpha : S \rightarrow S^\alpha \) 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 \( h^\alpha \) is implied by the distance conservancy of \( \alpha \).

The most well-studied abstraction heuristics are pattern database (PDB) heuristics. Given a planning task \( \Pi \) over state variables \( V \), such a heuristic is based on projecting \( \Pi \) onto a subset of its variables \( V^\alpha \subseteq V \). Such a homomorphism abstraction \( \alpha \) 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 \( S^\alpha \) 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, \(|S^\alpha| = O(1)\). 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 \( \Pi_i \) can be transported by some \( \Theta(i) \) vehicles, then starting from some \( j \), \( h^\alpha \) will not account at all for movements of vehicles essential for solving \( \Pi_j \) (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 \( h^\alpha \)-partition. Instead of computing \( h^\alpha(s) = h^\alpha(\alpha(s)) \) 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 \( h^\alpha(\alpha(s)) \) 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 \( h^\alpha \)-partition for a set of \( k \) states is \( O(|S^\alpha|(|S^\alpha| + |A| + k)) \) and \( O(|S^\alpha|) \), respectively. This is precisely what makes PDB heuristics so attractive in practice.

Aiming at preserving the attractiveness of the PDB \( h^\alpha \)-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 \( S^\alpha \) 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 \( h^\alpha \)-partition. The time and space complexity for the linear abstraction strategy of Helmert et al. are respectively \( O(|V| |S^\alpha| (|S^\alpha| + |A| + k \cdot |V|)) \) and \( O(|S^\alpha|) \). Similarly to PDB abstractions, the per-node computation of \( h^\alpha(s) \) with an MS abstraction \( \alpha \) is just a lookup in a data structure storing \( h^\alpha(\alpha(s)) \) 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 $S^\alpha$ is not limited even by $|V|$, and the size of the abstract space is not limited even by the size of the original space. 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 $\Pi$ can be found in polynomial time if

1. the causal graph of $\Pi$ forms a fork, and the root variable is binary-valued, or
2. the causal graph of $\Pi$ forms an inverted fork, and the domain of the root variable is fixed.

Now, given a general planning task $\Pi$, for each variable $v_i \in V$, let $V^i_0 \subseteq V$ contain $v_i$ and all its immediate successors in $CG(\Pi)$, and $V^i_1 \subseteq V$ contain $v_i$ and all its immediate predecessors in $CG(\Pi)$. At a high-level summary, the fork-decomposition of $\Pi$ with $|V| = n$ is obtained by

1. schematically constructing a set of projection abstractions \{\Pi^i_0, \Pi^i_1\}_{i=1}^n,
2. reformulating the actions of \{\Pi^i_0, \Pi^i_1\}_{i=1}^n to single-effect actions so that the causal graphs of $\Pi^i_0$ and $\Pi^i_1$ become respectively forks and inverted forks rooted in $v_i$, and
3. within each $\Pi^i_0$, abstracting the domain of $v_i$ to \{0, 1\}, and within each $\Pi^i_1$, abstracting the domain of $v_i$ to \{0, 1, \ldots, b\} with $b = O(1)$.

The problems \{\Pi^i_0, \Pi^i_1\}_{i=1}^n correspond to distance-conservative (though, after step (2), not necessarily homomorphic) abstractions of $\Pi$, 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 $|V^i_0|$ and $|V^i_1|$ 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, $h^F$, 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) HSPF 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 HSPF were the two best-performing cost-optimal planners at IPC-2008.

As it appears from Table 1, at least on the Logistics domain, the abstraction-based heuristic search favors competitors with the leading optimal-search planners: Both $h^F$ and $MS-10^5$ outperform HSPF and Gamer on these planning tasks. The comparison between $h^F$ and $MS-10^5$, however, is more important to us here. On the one hand, $h^F$ 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 $h^F$ 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) $h^F$ 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

<table>
<thead>
<tr>
<th>Task</th>
<th>$h^F$ nodes</th>
<th>$h^F$ time</th>
<th>MS-10^5 nodes</th>
<th>MS-10^5 time</th>
<th>HSPF nodes</th>
<th>HSPF time</th>
<th>Gamer time</th>
</tr>
</thead>
<tbody>
<tr>
<td>04-0</td>
<td>20</td>
<td>21</td>
<td>0</td>
<td>21</td>
<td>0</td>
<td>0.3</td>
<td>0.3</td>
</tr>
<tr>
<td>04-1</td>
<td>19</td>
<td>20</td>
<td>0</td>
<td>20</td>
<td>0</td>
<td>0.3</td>
<td>0.3</td>
</tr>
<tr>
<td>04-2</td>
<td>15</td>
<td>16</td>
<td>0</td>
<td>16</td>
<td>0</td>
<td>0.3</td>
<td>0.3</td>
</tr>
<tr>
<td>05-0</td>
<td>27</td>
<td>28</td>
<td>0</td>
<td>28</td>
<td>0</td>
<td>0.4</td>
<td>0.4</td>
</tr>
<tr>
<td>05-1</td>
<td>17</td>
<td>18</td>
<td>0</td>
<td>18</td>
<td>0</td>
<td>0.4</td>
<td>0.4</td>
</tr>
<tr>
<td>05-2</td>
<td>8</td>
<td>9</td>
<td>0</td>
<td>9</td>
<td>0</td>
<td>0.4</td>
<td>0.4</td>
</tr>
<tr>
<td>06-0</td>
<td>25</td>
<td>26</td>
<td>0</td>
<td>26</td>
<td>1.2</td>
<td>0.4</td>
<td>0.4</td>
</tr>
<tr>
<td>06-1</td>
<td>14</td>
<td>15</td>
<td>0</td>
<td>15</td>
<td>1.3</td>
<td>0.4</td>
<td>0.4</td>
</tr>
<tr>
<td>06-2</td>
<td>25</td>
<td>26</td>
<td>0</td>
<td>26</td>
<td>1.3</td>
<td>0.4</td>
<td>0.4</td>
</tr>
<tr>
<td>06-9</td>
<td>24</td>
<td>25</td>
<td>0</td>
<td>25</td>
<td>1.2</td>
<td>0.4</td>
<td>0.4</td>
</tr>
<tr>
<td>07-0</td>
<td>36</td>
<td>37</td>
<td>3.9</td>
<td>37</td>
<td>4.9</td>
<td>362179</td>
<td>4531</td>
</tr>
<tr>
<td>07-1</td>
<td>44</td>
<td>45</td>
<td>3.9</td>
<td>45</td>
<td>7.2</td>
<td>114335</td>
<td>95.8</td>
</tr>
<tr>
<td>07-3</td>
<td>56</td>
<td>57</td>
<td>3.9</td>
<td>57</td>
<td>5.9</td>
<td>32017</td>
<td>12.2</td>
</tr>
<tr>
<td>07-4</td>
<td>50</td>
<td>51</td>
<td>3.7</td>
<td>51</td>
<td>9.4</td>
<td>6720</td>
<td>26.5</td>
</tr>
<tr>
<td>07-8</td>
<td>45</td>
<td>46</td>
<td>15.2</td>
<td>46</td>
<td>9.7</td>
<td>599645</td>
<td>3376</td>
</tr>
<tr>
<td>07-11</td>
<td>42</td>
<td>43</td>
<td>12.5</td>
<td>43</td>
<td>43</td>
<td>1457130</td>
<td>43</td>
</tr>
<tr>
<td>11-0</td>
<td>48</td>
<td>49</td>
<td>214.3</td>
<td>49</td>
<td>37.4</td>
<td>701100</td>
<td>37.4</td>
</tr>
<tr>
<td>11-1</td>
<td>60</td>
<td>61</td>
<td>21597</td>
<td>61</td>
<td>55610</td>
<td>95.9</td>
<td>13.6</td>
</tr>
<tr>
<td>12-0</td>
<td>52</td>
<td>53</td>
<td>20.1</td>
<td>53</td>
<td>107</td>
<td>775996</td>
<td>41.6</td>
</tr>
<tr>
<td>12-1</td>
<td>68</td>
<td>69</td>
<td>252415</td>
<td>69</td>
<td>123</td>
<td>222340</td>
<td>87.5</td>
</tr>
</tbody>
</table>

Table 1: The results for $A^*$ on Logistics-2000 domain with the additive fork-decomposition heuristic “all forks” under uniform action cost partitioning. Comparison with $A^*$ and merge-and-shrink abstraction with $|S^\alpha| \leq 10^5$, HSPF heuristic-search planner, and Gamer planner. $h^F$, 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.
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 $h^*(s)$. The online part of the $h^*$-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 $\Pi = \{V, A, I, G, c\}$ be a planning task with a fork causal graph rooted in a binary-valued variable $r$. There exists an $h^*$-partition for $\Pi$ 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 $d = \max\, D(v)$.

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 $\Pi$. Let $D(r) = \{0, 1\}$, where $s[r] = 0$; for $\vartheta \in D(r)$, let $-\vartheta$ denote the opposite value 1 $- \vartheta$. Let $\sigma(r)$ be a 0/1 sequence of length 1 $+ \vartheta$, such that, for $1 \leq i \leq |\sigma(r)|$, $\sigma(r)[i] = 0/1$ if $i$ is odd/even. Let $D^+(\sigma(r))$ be the set of all non-empty prefixes of $\sigma(r)$ if $G[v]$ is unspecified, and the set of all non-empty prefixes of $\sigma(r)$ ending with $G[v]$, otherwise. For each $\sigma \in D^+(\sigma(r))$, $c(\sigma)$ denotes the (straightforwardly computable) cost of achieving the respective sequence of value changes of $r$.

For each $v \in V \setminus \{r\}$, let $DTG^0_v$ and $DTG^1_v$ 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 $\vartheta, \vartheta' \in D(v)$, compute the shortest (i.e., cost-minimal) paths from $\vartheta$ to $\vartheta'$ in $DTG^0_v$ and $DTG^1_v$.

2. For each $\sigma \in D^+(\sigma(r))$, and each $v \in V \setminus \{r\}$, build a layered digraph $L^0_v$ with $\sigma[r] + 1$ layers $L_0, \ldots, L_{|\sigma[r]|}$, where $L_0$ consists of only $s[v]$, and for $1 \leq i \leq |\sigma|$, $L_i$ consists of all nodes reachable from the nodes $L_{i-1}$ in $DTG^0_v/DTG^1_v$, if $i$ is odd/even. For each $\vartheta \in L_{i-1}$, $\vartheta' \in L_i$, $L_v(\sigma)$ contains an arc $(\vartheta, \vartheta')$ weighted with the distance from $\vartheta$ to $\vartheta'$ in $DTG^0_v/DTG^1_v$ if $i$ is odd/even.

3. Set $R = \emptyset$. For each $\sigma \in D^+(\sigma(r))$, a plan $\rho_\sigma$ for $\Pi$ is constructed as follows.

   i) For each $v \in V \setminus \{r\}$, find a shortest path from $s[v]$ to $G[v]$ in $L_v(\sigma)$. If no such path exists, go to the next $\sigma \in D^+(\sigma(r))$. The $\vartheta$-edge on this path (say, from $\vartheta \in L_{i-1}$ to $\vartheta' \in L_i$) corresponds to the shortest path from $\vartheta$ to $\vartheta'$ in either $DTG^0_v$ or $DTG^1_v$, and we denote this path by $S^\vartheta_\sigma$.

   ii) Set $R = R \cup \{\rho_\sigma\}$, $\rho_\sigma = S^1_\sigma \cdot a_{\sigma[2]} \cdot S^2_\sigma \cdots a_{\sigma[m]} \cdot S^m_\sigma$, where $m = |\sigma|$, $S^i_\sigma$ is obtained by an arbitrary merge of $\{S^i_v\}_{v \in V \setminus \{r\}}$, and $a_\vartheta$ is the action that changes the value of $r$ to $\vartheta$.

4. If $R = \emptyset$, then “unsolvable”. Otherwise, return $\rho = \arg\min_{\rho_\sigma \in R} c(\rho_\sigma)$.

This algorithm can be formulated as follows.

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

\[
\begin{align*}
g_{\vartheta;i} &= \begin{cases}
p_{\vartheta,v},\vartheta;\vartheta_r, & i = 1 \\
\min_{\vartheta'} g_{\vartheta;\vartheta';\vartheta_r} + p_{\vartheta',\vartheta;\vartheta_r}, & 1 < i \leq \delta_\vartheta, i \text{ is odd} \\
\min_{\vartheta'} g_{\vartheta;\vartheta';\vartheta_r} + p_{\vartheta',\vartheta;\vartheta_r}, & 1 < i \leq \delta_\vartheta, i \text{ is even} \\
g_{\vartheta;i-1}, & \delta_\vartheta < i \leq d + 1
\end{cases}
\end{align*}
\]

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

\[
h^*(s) = \min_{\sigma \in D^+(\sigma(r))} \left\{ c(\sigma) + \sum_{v \in V \setminus \{r\}} g_{\vartheta,v;\vartheta;\vartheta_r} \right\}
\]

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 $v \in V \setminus \{r\}$, $\vartheta \in D(v)$, $1 \leq i \leq d + 1$, and $\vartheta_r \in D(r)$, let $g_{\vartheta;i}(\vartheta_r)$ be the cost of the cheapest sequence of actions changing $\vartheta$ to $\vartheta_r$ provided the value changes of $r$ induce a 0/1 sequence of length $i$ starting with $\vartheta_r$. Very similarly to before, the set $\{g_{\vartheta;i}(\vartheta_r)\}$ is given by the solution of the recursive equation

\[
\begin{align*}
g_{\vartheta;i}(\vartheta_r) &= \begin{cases}
p_{\vartheta,G[v];\vartheta;\vartheta_r}, & i = 1 \\
\min_{\vartheta'} g_{\vartheta;i-1}(\vartheta_r) + p_{\vartheta',\vartheta;\vartheta_r}, & 1 < i \leq \delta_\vartheta \\
g_{\vartheta;i-1}(\vartheta_r), & \delta_\vartheta < i \leq d + 1
\end{cases}
\end{align*}
\]

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

\[
h^*(s) = \min_{\sigma \in D^+(\sigma(r))} \left\{ c(\sigma) + \sum_{v \in V \setminus \{r\}} g_{\vartheta,v;\vartheta;\vartheta_r} \right\}
\]

With the new formulation, the only computation that has to be performed per search node is this of the final minimization over $D^+(\sigma(r))$, which is the lightest part of the whole algorithm anyway. The major computation, notably this of $\{\rho_\vartheta,\vartheta';\vartheta_r\}$ 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 $O(d^2|V| + |A_r|)$; the $|A_r|$ component stems from
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 $\mathbb{E}^*\sigma(r)$.

Figure 1(b) shows the structural-pattern database created for a fork-structured problem with a binary-valued root variable $r$, 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 $r = 0 \land |\sigma| = 5$.

**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 $|D(r)| = b = O(1)$. There exists an $h^*$-partition for $\Pi$ such that, for any set of $k$ states, the time and space complexity of that $h^*$-partition is $O(|V||A_r|^{b-1} + d^2|V| + k \cdot |V||A_r|^{b-1})$ and $O(|V||A_r|^{b-1} + d^2|V|)$, respectively, where $d = \max_v |D(v)|$.

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 $\Pi$ 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 $\vartheta$ to $\vartheta'$ in $\mathcal{D}(v)$.

2. Enumerate all $\Theta(|A_r|^{b-1})$ cycle-free paths from $s[r]$ to $G[r]$ in $\mathcal{D}(v)$. For each such path, construct a plan for $\Pi$ 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 $\Pi$, 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 $\vartheta$ to $\vartheta'$. 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 $\mathcal{D}(v)$, let $g_\pi$ be the cost of the cheapest path from $s$ in $\Pi$ based on $\pi$, and the shortest paths computed in (1). Each $g_\pi$ can be computed as

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

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

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

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 $\pi$, and each parent variable $v \in V \setminus \{r\}$, we know what would be the first value of $v$ required by $\pi$. Given that, we can pre-compute the cost-optimal plans induced by each $\pi$ 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 $v \in \mathcal{D}(r)$ and each cycle-free path $\pi = a_1 \cdots a_m$ from $\vartheta_r$ to $G[r]$ in $\mathcal{D}(v)$, let a “proxy” state $s_{\pi}$ be

$$s_{\pi}[v] = \begin{cases} \vartheta_r, & v = r \\ G[v], & \forall 1 \leq i \leq m : \text{pre}(a_i)[v] \text{ is unspecified}, \\ \text{pre}(a_i)[v], & i = \arg\min_{j} \{\text{pre}(a_j)[v] \text{ is specified}\} \\ \end{cases}$$
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 (\( \tilde{s} \)), and the number of domains in which the planners achieved superior performance (\( w \)).

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 \( h^D \) 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 \( h^2 \) 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 \( h^D \) heuristic is the union of these for \( h^D \) and \( h^2 \). The action cost partitioning for all three ensembles was set to “uniform”.

We compare with two baseline approaches, namely blind search (\( A^0 \) with a heuristic function which is 0 for goal states and 1 otherwise) and \( A^+ \) with the \( h_{\text{max}} \) 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, \(|S^0| \leq 10^4\)
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\(p^c_2 \) (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\(p^c_2 \) 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”.

The overall performance of \( A^+ \) with the fork-decomposition heuristic \( h^3 \) exhibited the best overall performance according to all three measures. The overall performance of \( A^+ \) with the other two heuristics \( h^2 \) and \( h^{2\alpha} \) was not as good as with \( h^3 \), and yet, in terms of, e.g., absolute number of solved instances, \( h^{2\alpha} \) and \( h^3 \) came second and forth, respectively. Likewise, \( h^3 \) still slightly outperforms \( h^2 \) 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 \( h^\alpha \)-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


