Abstraction Heuristics Extended with Counting Abstractions

Blai Bonet
Departamento de Computación
Universidad Simón Bolívar
Caracas, Venezuela
bonet@ldc.usb.ve

Abstract

State-of-the-art abstraction heuristics are those constructed by the merge-and-shrink approach in which an abstraction consists of a labeled transition system, and the composition of abstractions correspond to the synchronized product of transition systems. Merge-and-shrink heuristics build a composite abstraction from atomic abstractions that are directly associated with the variables of the planning problem. In this paper, we show that the framework of labeled transition systems is more general, and propose a new type of abstraction called the counting abstraction. Counting abstractions can be transparently combined with other type of abstractions to get more informative heuristics. We show how to effectively construct the counting abstractions and presents preliminary experiments over benchmark problems.

Introduction

Abstraction heuristics is a one of the four dominating classes of admissible heuristics used in planning (Helmert and Domshlak 2009). This class includes pattern-database heuristics (Edelkamp 2001; Haslum, Bonet, and Geffner 2005; Haslum et al. 2007; Katz and Domshlak 2008b) and merge-and-shrink heuristics (Helmert, Haslum, and Hoffmann 2007) which are known to dominate, with the proper choice of abstractions and merges, landmark heuristics (Karpas and Domshlak 2009; Helmert and Domshlak 2009) and additive $h^{\max}$ heuristics (Haslum, Bonet, and Geffner 2005; Katz and Domshlak 2008a).

The merge-and-shrink value for a state s is the length of a shortest path, in an abstraction of the problem, from the abstract state associated to s to an abstract goal state. A merge-and-shrink abstraction is a labeled transition system that is the result of composing atomic abstractions associated with the SAS$^+$ variables of the planning problem, and an abstraction mapping that maps states into abstract states.

However, labeled transition systems are powerful enough to represent more general abstractions than those captured by the merge-and-shrink heuristics. In this paper, we propose a new type of abstraction, called the counting abstraction, that naturally fits into the merge-and-shrink framework and that can be used to obtain more informative heuristics. Briefly, a counting abstraction captures quantitative aspects of the planning problem that *cut across multiple SAS$^+$ variables*; e.g., one can define a counting abstraction that tracks the number of goals that remain to be achieved from a state in an admissible manner.

The use of counting abstraction is not new in search. Prieditis (1993) considers search problems specified in an extended STRIPS language and a set of transformations for automatically creating abstractions (heuristics). These transformations contain operations that define counting abstractions. However, Prieditis does not mention the difficulties involved with unsafe operators (see below) nor considers labeled transition systems.

The following sections revise the class of problems considered and relevant concepts of transitions systems. Then, we present the counting abstractions and their effective construction and integration with merge and shrink. The paper concludes with preliminary experiments and a discussion.

Planning Problems

We consider SAS$^+$ problems $P = \langle V, O, s_\circ, s_\star \rangle$ with variables $X \in V$, each with domain D_X, operators $o \in O$, initial state $s_\circ$ and goal state $s_\star$. Each operator $o \in O$ is specified by a precondition and postcondition denoted by pre$[o]$ and post$[o]$ respectively. The precondition and postcondition are partial SAS$^+$ states (i.e. partial valuations of variables). The value of a variable X at a full or partial state s is $X(s)$; e.g., $X(\text{pre}[o])$ denotes the value of X in the precondition of o. Unlike pure SAS$^+$, we also consider operators o such that $X(\text{post}[o]) \neq \bot$ and $X(\text{pre}[o]) = \bot$ for some X, that set the value of X to $X(\text{post}[o])$ *independently* of its previous value; such operators frequently appear when a STRIPS problem is automatically translated into SAS$^+$.

Abstractions

In the approach of Helmert, Haslum, and Hoffmann (2007), adapted from the one of Dräger, Finkbeiner, and Podelski (2006) for directed model checking, an abstraction of a transition system $\mathcal{T}$ is a transition system $\mathcal{T}'$ together with a mapping α that maps states in $\mathcal{T}$ into states in $\mathcal{T}'$, and the composition of abstractions correspond to the *synchronized product* of abstractions. Hence, starting from *atomic abstractions* associated to single SAS$^+$ variables, Helmert, Haslum, and Hoffmann explore different *composition strate-*

gies to build a global composite that defines a heuristic for the planning problem. In this section, we revise the fundamental concepts of abstraction as defined by them.

A transition system is a tuple $\mathcal{T} = \langle S, L, A, s_0, S_T \rangle$ made of states S, set L of labels (actions), labeled transitions $A \subseteq S \times L \times S$, initial state $s_0 \in S$ and target states $S_T \subseteq S$. An abstraction for $\mathcal{T}$, on the other hand, is a pair $\mathcal{A} = \langle \mathcal{T}', \alpha \rangle$ where $\mathcal{T}' = \langle S', L, A', s'_0, S'_T \rangle$ is a transition system over the same set of labels and $\alpha : S \to S'$ is a homomorphism from $\mathcal{T}$ to $\mathcal{T}'$; i.e., $\langle \alpha(s), l, \alpha(t) \rangle \in A'$ for all $\langle s, l, t \rangle \in A$, $\alpha(s_0) = s'_0$ and $\alpha(S_T) \subseteq S'_T$.

An abstraction $\mathcal{A} = \langle \mathcal{T}', \alpha \rangle$ for $\mathcal{T}$ defines an admissible heuristic $h^{\mathcal{A}}$ for finding shortest paths on $\mathcal{T}$, where $h^{\mathcal{A}}(s)$ is the length of a shortest path from $\alpha(s)$ to an abstract goal state. The admissibility of $h^{\mathcal{A}}$ follows directly from the homomorphism because every goal-reaching path from s on $\mathcal{T}$ induces a path of the same length, via α, from $\alpha(s)$ to an abstract goal state.

Two abstractions $\mathcal{A}' = \langle \langle S', L, A', s'_0, S'_T \rangle, \alpha' \rangle$ and $\mathcal{A}'' = \langle \langle S'', L, A'', s''_0, S''_T \rangle, \alpha'' \rangle$ for $\mathcal{T}$ are combined into a composite abstraction $\mathcal{A} \otimes \mathcal{A}' \doteq \langle \langle S, L, A, s_0, S_T \rangle, \alpha \rangle$ where $\alpha(s) = (\alpha'(s), \alpha''(s))$, $S = S' \times S''$, and $\langle (s, s'), l, (t, t') \rangle \in A$ iff $\langle s, l, t \rangle \in A'$ and $\langle s', l, t' \rangle \in A''$. It is not hard to show that α is a homomorphism from $\mathcal{T}$ into $\langle S, L, A, s_0, S_T \rangle$ and thus $h^{\mathcal{A}' \otimes \mathcal{A}''}$ is admissible and, furthermore, $\max\{h^{\mathcal{A}'}, h^{\mathcal{A}''}\} \leq h^{\mathcal{A}' \otimes \mathcal{A}''}$.

Given a SAS$^+$ problem $P = \langle V, O, s_\circ, s_\star \rangle$, the transition system $\mathcal{T}_P = \langle S, L, A, s_0, S_T \rangle$ consists of the collection of SAS$^+$ states S, $L = O$, $s_0 = s_\circ$, $S_T = \{s : s|_{\text{vars}(s_\star)} = s_\star\}$, and $\langle s, o, s' \rangle \in A$ iff the operator o is applicable at s and results in s'. $\mathcal{T}_P$ is a faithful representation of P: all plans of P, and only them, correspond to paths on $\mathcal{T}_P$. Furthermore, this correspondence preserves path lengths and thus a shortest plan on P corresponds to a shortest path on $\mathcal{T}_P$.

Abstractions (hence heuristics) for $\mathcal{T}_P$ are obtained by forming synchronized products from atomic abstractions. Helmert, Haslum, and Hoffmann build complex abstractions from abstractions $\mathcal{A}_X$ for SAS$^+$ variables: $\mathcal{A}_X \doteq \langle \langle S, L, A, s_0, S_T \rangle, \alpha \rangle$ has states $S = D_X$, labels $L = O$, $s_0 = X(s_\circ)$, S_T equals $\{X(s_\star)\}$ (resp. D_X if $X(s_\star) = \bot$), and $\langle x, o, x' \rangle \in A$ iff 1) $X(\text{pre}[o]) = x$ and $X(\text{post}[o]) = x'$, 2) $X(\text{pre}[o]) = \bot$ and $X(\text{post}[o]) = x'$, or 3) $X(\text{pre}[o]) = X(\text{post}[o]) = \bot$ and $x = x'$. The abstraction map is $\alpha(s) = X(s)$.

Counting Abstractions

The above framework is general enough to accommodate abstractions of a more general type. Here, we present counting abstractions, another type of abstraction, that capture *quantitative aspects* of the planning task. These abstractions may be combined among them or with the composites obtained from the variables to build better composites.

An atom is an assignment '$X{=}x$' where X is a variable and $x \in D_X$.[1] It holds at state s, written $s \models X{=}x$, iff $X(s) = x$. For a collection $\mathcal{C}$ of atoms, the *counting variable* for $\mathcal{C}$ is the variable that assigns to each state s the number $\mathcal{C}(s)$ of atoms in $\mathcal{C}$ holding at s; i.e., $\mathcal{C}(s) \doteq |\{p \in \mathcal{C} : s \models p\}|$. Clearly, the domain of $\mathcal{C}$ is $D_{\mathcal{C}} \doteq \{0, 1, \ldots, |\mathcal{C}|\}$.

A counting variable $\mathcal{C}$ defines an abstraction $\mathcal{A}_{\mathcal{C}} = \langle \mathcal{T}_{\mathcal{C}} = \langle S, L, A, s_0, S_T \rangle, \alpha \rangle$ of $\mathcal{T}_P$ made of states $S = D_{\mathcal{C}}$, labels $L = O$, initial state $s_0 = \mathcal{C}(s_\circ)$, target states $S_T = \{\mathcal{C}(s) : s \models s_\star\}$, and transitions $\langle v, o, v' \rangle \in A$ iff there is a state s in $\mathcal{T}_P$ such that o is applicable at s, $\mathcal{C}(s) = v$ and $\mathcal{C}(\text{result}(s, o)) = v'$. The abstraction map is $\alpha(s) = \mathcal{C}(s)$.

For example, let $\mathcal{C}_\star$ be the collection of goal atoms so that $\mathcal{C}_\star(s)$ counts the number of 'goals' achieved in s. Then, the length of a shortest path from $\mathcal{C}_\star(s)$ to an abstract goal is an admissible estimate for the length of a shortest plan from s to $s_\star$. This estimate is related, but not necessarily equal, to the one obtained by counting the number of unsatisfied goals in s. Indeed, $h^{\mathcal{A}_{\mathcal{C}_\star}}$ is admissible while the latter is not because a naive counting assumes that goals are independent and does not take into account the possible interactions among goals; e.g., operators that establish two or more goals simultaneously.

By definition, $\mathcal{A}_{\mathcal{C}}$ is an abstraction of $\mathcal{T}_P$ and thus provides admissible estimates that can be freely combined with other abstractions via synchronized products. However, $\mathcal{A}_{\mathcal{C}}$ is defined in terms of the transition system $\mathcal{T}_P$ that is not directly accessible. In the rest of this section we show how to *efficiently construct* an approximation abstraction $\widehat{\mathcal{A}_{\mathcal{C}}}$ from the SAS$^+$ representation of the planning problem.

Effective Construction

The challenge for computing a sound abstraction is to take care of the variables that have undefined precondition values in the operators. Such operators are closely related to the "unsafe" operators in the approaches to planning based on Petri nets (Hickmott et al. 2007; Bonet et al. 2008). Operators o that delete atoms $p \in \mathcal{C}$ which are not preconditions cause uncertainty about the transitions labeled by o: if p holds at s then p contributes -1 to the difference $\mathcal{C}(\text{result}(s, o)) - \mathcal{C}(s)$, while if p does not hold at s then it contributes 0 to the difference.

One way to deal with such operators is to consider all safe versions of them. This method however is not practical as the number of versions may be exponential. On the other hand, this is not really needed; we only need to know the aggregate contribution of the atoms in $\mathcal{C}$ to the difference $\mathcal{C}(\text{result}(s, o)) - \mathcal{C}(s)$, for every possible s.

In fact, fix an operator o and let $base_o$ be the number of atoms in $\mathcal{C}$ that appear as preconditions of o, and let

$$\delta_o^+ \doteq \{X : X{=}X(\text{pre}[o]) \notin \mathcal{C} \wedge X{=}X(\text{post}[o]) \in \mathcal{C}\},$$
$$\delta_o^- \doteq \{X : X{=}X(\text{pre}[o]) \in \mathcal{C} \wedge X{=}X(\text{post}[o]) \notin \mathcal{C}\},$$
$$\alpha_o \doteq \{X : X(\text{pre}[o]) = \bot \wedge X{=}X(\text{post}[o]) \in \mathcal{C}\},$$
$$\beta_o \doteq \text{Vars}_{\mathcal{C}} \cap \{X : X(\text{pre}[o]) = \bot \wedge X{=}X(\text{post}[o]) \notin \mathcal{C}\}$$

where $\text{Vars}_{\mathcal{C}}$ is the set of variables mentioned in $\mathcal{C}$.

Lemma 1. *Let s, s' be two SAS$^+$ states, o an operator applicable at s and $s' = \text{result}(s, o)$. Then, $\mathcal{C}(s) \geq base_o$ and $\mathcal{C}(s') = \mathcal{C}(s) + |\delta_o^+| - |\delta_o^-| + k$ for some $-|\beta_o| \leq k \leq |\alpha_o|$.*

[1]We only consider atoms over single variables yet the framework supports atoms of the form $\mathbf{X}{=}\,\mathbf{x}$ where $\mathbf{X}$ is a subset of variables and $\mathbf{x}$ is a value for $\mathbf{X}$.

Proof. Clearly, $\mathcal{C}(s) \geq base_o$ since $s \models \text{pre}[o]$. For the second claim, let def $\doteq \{X : X(\text{pre}[o]) \neq \bot\}$ and define

$$\begin{aligned}
\delta_s^+ &\doteq \text{def} \cap \{X : X{=}X(s) \notin \mathcal{C} \wedge X{=}X(\text{post}[o]) \in \mathcal{C}\}\,, \\
\delta_s^- &\doteq \text{def} \cap \{X : X{=}X(s) \in \mathcal{C} \wedge X{=}X(\text{post}[o]) \notin \mathcal{C}\}\,, \\
\alpha_s^+ &\doteq \overline{\text{def}} \cap \{X : X{=}X(s) \notin \mathcal{C} \wedge X{=}X(\text{post}[o]) \in \mathcal{C}\}\,, \\
\alpha_s^0 &\doteq \overline{\text{def}} \cap \{X : X{=}X(s) \in \mathcal{C} \wedge X{=}X(\text{post}[o]) \in \mathcal{C}\}\,, \\
\beta_s^- &\doteq \overline{\text{def}} \cap \{X : X{=}X(s) \in \mathcal{C} \wedge X{=}X(\text{post}[o]) \notin \mathcal{C}\}\,, \\
\beta_s^0 &\doteq \overline{\text{def}} \cap \{X : X{=}X(s) \notin \mathcal{C} \wedge X{=}X(\text{post}[o]) \notin \mathcal{C}\}\,.
\end{aligned}$$

It is not hard to see that every X in $\alpha_s^0 \cup \beta_s^0$ contributes 0 to the difference $\mathcal{C}(s') - \mathcal{C}(s)$, that every X in $\delta_s^- \cup \beta_s^-$ contributes -1 to the difference, and that every X in $\delta_s^+ \cup \alpha_s^+$ contributes 1 to the difference. Also, $\alpha_s^+ \subseteq \alpha_o$ and $\beta_s^- \subseteq \beta_o$. Since $\delta_s^+ = \delta_o^+$ and $\delta_s^- = \delta_o^-$ when o is applicable at s,

$$\begin{aligned}
\mathcal{C}(s') - \mathcal{C}(s) &= |\delta_s^+| - |\delta_s^-| + |\alpha_s^+| - |\beta_s^-| \\
&= |\delta_o^+| - |\delta_o^-| + a - b
\end{aligned}$$

for some $0 \leq a \leq |\alpha_o|$ and $0 \leq b \leq |\beta_o|$. Thus, $\mathcal{C}(s') = \mathcal{C}(s) + |\delta_o^+| - |\delta_o^-| + k$ for some $-|\beta_o| \leq k \leq |\alpha_o|$. $\square$

The approximation $\widehat{\mathcal{A}_\mathcal{C}}$ is thus $\langle\langle S, L, A, s_0, S_T\rangle, \alpha\rangle$ where $S = \{0, \ldots, |\mathcal{C}|\}$, $L = O$, $s_0 = \mathcal{C}(s_\circ)$, $S_T = \{v \in S : \mathcal{C}(s_\star) \leq v\}$, $\langle v, o, v'\rangle \in A$ iff $v \geq base_o$ and v' is such that $v' = v + |\delta_o^+| - |\delta_o^-| + k$ for some $-|\beta_o| \leq k \leq |\alpha_o|$ with the constraint $0 \leq v' \leq |\mathcal{C}|$, and $\alpha(s) = C(s)$. Finally, observe that when $\mathcal{C}$ only consists of atoms $X{=}x$ such that X is a variable that appears in the goal, then the terminal states can be strengthened to $S_T = \{\mathcal{C}(s_\star)\}$. The following result is a direct consequence of the definition and Lemma 1.

Theorem 2. *The approximation $\widehat{\mathcal{A}_\mathcal{C}}$ is an abstraction of $\mathcal{A}_\mathcal{C}$ and, by transitivity, an abstraction of $\mathcal{T}_P$ as well. Furthermore, $\widehat{\mathcal{A}_\mathcal{C}}$ can be constructed in polynomial time.*

Merge-and-Shrink Strategies

A composition strategy dictates how to form a global composite beginning from a collection Λ of atomic abstractions. The simplest type of strategy is a *linear strategy* that maintains a unique non-atomic abstraction $\mathcal{A}$ that is iteratively enlarged with atomic abstractions. Initially, $\mathcal{A}$ is the trivial non-informative abstraction made of a single state, and at each iteration an atomic abstraction $\mathcal{A}_i \in \Lambda$ is extracted to update $\mathcal{A}$ as $\mathcal{A} \otimes \mathcal{A}_i$. The process ends when Λ becomes empty, a time at which $\mathcal{A}$ consolidates all the information contained in the atomic abstractions. Since $\oplus$ is associative and commutative, the extraction order is irrelevant, but becomes relevant when the size of $\mathcal{A}$ is controlled.

The size of $\mathcal{A} \otimes \mathcal{A}_i$ is the product of the sizes of $\mathcal{A}$ and $\mathcal{A}_i$. Hence, given a requirement N on the maximum amount of available memory, the partial composites are shrunk appropriately to satisfy the constraint; Helmert, Haslum, and Hoffmann propose to always shrink $\mathcal{A}$ to size $\lfloor N/\text{size}(\mathcal{A}_i)\rfloor$ before forming the product $\mathcal{A} \otimes \mathcal{A}_i$.

Shrinking is done by a sequence of coalesce operations until the size reaches the target value; the coalesce of a group of states is a new abstraction in which all states in the group are replaced by a new single state. Helmert, Haslum, and Hoffmann discuss operations that preserve the heuristic values, called g-, h- and f-preserving operations. However, even if the heuristic values are preserved, the *potential* of the abstraction always diminishes when states are fusioned because structure, and hence information, is lost.

In our experiments, we tried two different merge strategies. The first is the default strategy of merge-and-shrink in which the variables are statically ordered and the next abstraction $\mathcal{A}_X$ to pick corresponds to the first variable that is either 1) causally connected to a variable in $\mathcal{A}$, or 2) a goal variable. The second strategy is a new strategy that is like the previous but in which the causally connected variables are considered in a breadth-first manner, by using a FIFO queue that maintains the causally connected variables. Another option is to use a LIFO queue that implements a depth-first order; yet, it did not produce good results.

Independently of the merge strategy, if the counting variables are brought into the global composite at last, the result is a heuristic that is at least as good as the one that considers no counting variables. In our experiments, this heuristic does not improve over merge-and-shrink either because the counting abstractions do not provide additional information or because the shrinking operations had reduced the potential of the global abstraction. Thus, we treat the counting abstractions as first-class citizens like the standard atomic abstractions. In any case, the relevant thing is to think about the counting abstractions as an effective way to boost the pure merge-and-shrink heuristic; something this is not clear how to do it in the original setting because once the atomic abstractions are exhausted, there are not other sources of information available.

Preliminary Experiments

Experiments were performed on problems from the International Planning Competition with an Intel Xeon processor of 1.86GHz and with 2Gb of RAM. We implemented the counting abstractions on top of the merge-and-shrink heuristic of the Fast Downward distribution.

The experiment consisted of extending the collection of standard atomic abstractions with a *fixed number* of 5 counting abstractions: $\mathcal{C}_\star$ that "counts goal atoms", $\mathcal{C}_\circ$ that consists of the goal variables evaluated at $s_\circ$, and three counting abstractions with 2 atoms each obtained by randomly sampling a variable and a value. The maximum number of nodes in an abstraction is set to parameter N.

Table 1 gives results for selected problems with the length $h^*(s_\circ)$ of an optimal plan and the number of expanded nodes until the last f-layer, an objective measure for the quality of a heuristic, for four heuristics: two pure merge-and-shrink (M&S-1 and M&S-2) and two merge-and-shrink with counting variables (M&S-#1 and M&S-#2). The versions in each group differ on the merge strategy: the first uses the original static strategy and the second the LIFO strategy.

The results are mixed because no heuristic dominates the others. However, it is clear that there is some sort of dominance in a per-domain basis. Another interesting observation is the column of zeroes for M&S-#2 across all instances of Gripper. These zeroes mean that M&S-#2 equals the perfect heuristic h^* on these instances. Indeed, it can be shown that M&S-#2 is perfect in Gripper because it counts the number of balls at each room and tracks the position of the robot, enough information to build a perfect abstraction.

Independently of these preliminary results, we believe that the performance of merge-and-shrink heuristics can be improved by defining more interesting counting variables and with better merge strategies. A clear advantage of counting variables over SAS$^+$ variables is that one is not limited by their number; any set of fluents defines a counting variable that may be relevant for the problem.

Discussion

We defined counting abstractions that are transparently combined with SAS$^+$ abstractions within the framework of labeled transition systems. The abstractions are combined using a merge strategy. We tried the original merge-and-shrink strategy and a new one that explore causally connected variables in a breadth-first order.

In general, any function that maps states into a domain D defines an abstraction that can be expressed as a labeled transition system. However, an abstraction is *feasible in practice* only if it can be effectively constructed from the description of the planning problem. In our case, we showed how to construct counting abstractions from SAS$^+$ representations. It is matter of future research to study other types of general and feasible abstractions.

Acknowledgments Thanks to J. Hoffmann, M. Helmert and P. Haslum for interesting discussions, and the reviewers.

References

Bonet, B.; Haslum, P.; Hickmott, S.; and Thiébaux, S. 2008. Directed unfolding of petri nets. *Trans. on Petri Nets and Other Models of Concurrency I* 1:172–198.

Dräger, K.; Finkbeiner, B.; and Podelski, A. 2006. Directed model checking with distance-preserving abstractions. *Proc. 13th Int. SPIN Workshop*, 19–34.

Edelkamp, S. 2001. Planning with pattern databases. *Proc. 6th European Conf. on Planning*, 13–24.

Haslum, P.; Botea, A.; Helmert, M.; Bonet, B.; and Koenig, S. 2007. Domain-independent construction of pattern database heuristics for cost-optimal planning. *Proc. 22th National Conf. on Artificial Intelligence*, 1007–1012.

Haslum, P.; Bonet, B.; and Geffner, H. 2005. New admissible heuristics for domain-independent planning. *Proc. 20th National Conf. on Artificial Intelligence*, 1163–1168.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? *Proc. 19th Int. Conf. on Automated Planning and Scheduling*, 162–169.

Helmert, M.; Haslum, P.; and Hoffmann, J. 2007. Flexible abstraction heuristics for optimal sequential planning. *Proc. 17th Int. Conf. on Automated Planning and Scheduling*, 176–183.

Hickmott, S.; Rintanen, J.; Thiébaux, S.; and White, L. 2007. Planning via petri net unfolding. *Proc. 20th Int. Conf. on Automated Planning and Scheduling*, 1904–1911.

Karpas, E., and Domshlak, C. 2009. Cost-optimal planning with landmarks. *Proc. 21st Int. Joint Conf. on Artificial Intelligence*, 1728–1733.

Katz, M., and Domshlak, C. 2008a. Optimal additive composition of abstraction-based admissible heuristics. *Proc. 18th Int. Conf. on Automated Planning and Scheduling*, 174–181.

Katz, M., and Domshlak, C. 2008b. Structural patterns heuristics via fork decomposition. *Proc. 18th Int. Conf. on Automated Planning and Scheduling*, 182–189.

Prieditis, A. 1993. Machine discovery of effective admissible heuristics. *Machine Learning* 12:117–141.

		static strategy		LIFO strategy	
inst.	$h^*(s_\circ)$	M&S-1	M&S-#1	M&S-2	M&S-#2
Gripper: $N = 50{,}000$					
03	23	9,318	10,298	**0**	**0**
04	29	68,186	65,681	32,514	**0**
05	35	376,494	371,720	332,629	**0**
06	41	1,982,014	1,974,279	1,934,383	**0**
07	47	10,091,966	10,080,246	10,047,485	**0**
Parcprinter: $N = 10{,}000$					
01	8	**0**	**0**	**0**	**0**
02	15	**0**	102	**0**	**0**
03	22	**0**	1,949	**0**	1,833
04	29	**0**	123,129	**0**	86,589
05	36	3,938,986	—	**0**	4,411,801
Pipesworld-notankage: $N = 2{,}500$					
06	10	1,481	**1,454**	9,739	1,582
07	8	205	**171**	2,033	245
08	10	**475**	501	27,146	498
09	13	128,528	**77,204**	1,500,464	123,148
10	18	3,002,179	**2,889,572**	—	—
Satellite: $N = 10{,}000$					
02	13	**0**	**0**	0	**0**
03	11	**0**	766	0	588
04	17	**0**	12,321	392	8,622
05	15	**49,185**	222,067	125,677	247,228
06	20	**2,365,252**	3,945,065	3,141,649	3,662,944
Sokoban: $N = 10{,}000$					
06	35	3,534	8,739	3,468	**3,440**
07	69	171,467	278,898	**162,188**	163,927
08	76	4,980,753	8,043,455	4,924,825	**4,634,145**
09	88	491,551	846,583	490,206	**431,562**
10	95	260,051	741,567	**257,767**	258,404
Trucks: $N = 10{,}000$					
02	17	4,185	11,554	**398**	3,289
03	20	181,538	111,559	**19,697**	95,228
04	23	2,591,406	1,915,555	1,707,330	**1,690,526**
05	25	—	13,328,955	**10,968,136**	11,596,010
07	23	8,066,124	5,103,416	**1,107,526**	3,632,298

Table 1: Number of expanded nodes until the last f-layer for different problems; N is the maximum # of states for the abstraction. The columns show the name of the instance, the length of the optimal plan, and the number of expanded nodes for merge-and-shrink without and with counting variables for two different merge strategies (best in boldface).