Optimal Planning and Shortcut Learning: An Unfulfilled Promise

Erez Karpas and **Carmel Domshlak**
Faculty of Industrial Engineering and Management, Technion — Israel Institute of Technology

Abstract

An existential optimal landmark is a set of actions, one of which must be used in some optimal plan. Recently, Karpas and Domshlak (2012) introduced a technique for deriving such existential optimal landmarks, which is based on using shortcut rules — rules which take a path, and attempt to find a cheaper path that achieves some of the propositions that the original path achieved. The shortcut rules that were originally used were of a limited form, and only attempted to remove parts of the given path. One would expect that using more sophisticated shortcut rules would result in a more informative heuristic, although possibly at the cost of increased computation time. We show that, somewhat surprisingly, more sophisticated shortcut rules, which are learned online, during search, result in a very small increase in informativeness on IPC benchmarks. Together with the increased computational cost, this leads to a decrease in the number of problems solved, and leaves finding efficient, informative shortcut rules as a standing challenge.

Introduction

Until not long ago, admissible heuristics were perceived as a necessary component of optimal heuristic search. However, recently, Karpas and Domshlak (2012) defined the notions of *global admissibility* and *global path-admissibility* of a heuristic. We denote the cost of an optimal path from s to the closest goal by $h^*(s)$. A heuristic h is globally admissible if there exists some optimal solution ρ, such that for every state s along ρ, $h(s) \leq h^*(s)$. A path-dependent heuristic h is *globally path-admissible* if there exists some optimal solution ρ, such that for every prefix π of ρ, $h(\pi) \leq h^*(s_0[\![\pi]\!])$, where $s_0[\![\pi]\!]$ is the state reached by path π. Both of these properties are weaker than admissibility, but are still enough to guarantee optimality of the solution. Karpas and Domshlak described a globally path-admissible heuristic, based upon existential optimal landmarks ($\exists$-opt landmarks, for short). An $\exists$-opt landmark is a set of actions, one of which must be used in *some* optimal plan. These $\exists$-opt landmarks are derived using the notion of intended effects of a path π — the possible justifications for why π might be a prefix of an optimal solution.

Because finding the exact set of intended effects is computationally infeasible, a sound approximation of the intended effects was used, which is based upon *shortcut rules*. A shortcut rule can be viewed as a function that takes as its input a path π, and attempts to find a cheaper path π', which achieves some of the facts that π achieves. Any subset of facts that is achieved by π' can not be an intended effect of π, because there is a cheaper way to achieve it. Therefore, any continuation of π into an optimal plan must use some fact which was achieved by π, but not by π'. Thus, $\Phi = s_0[\![\pi]\!] \setminus s_0[\![\pi']\!]$ describes an $\exists$-opt landmark for π, consisting of all actions which have a precondition in Φ.

While any type of shortcut rule can be used to derive $\exists$-opt landmarks, Karpas and Domshlak implemented only shortcut rules of a limited form, which attempt to remove some actions from π, without trying to add any new actions to replace them. One would expect that using more sophisticated shortcut rules, which combine action removal with adding actions, would result in a more informative heuristic. Similarly to our work, the planning by rewriting paradigm (Ambite and Knoblock 2001) is also based on shortcut rules. However the rules considered by Ambite and Knoblock are manually specified, while we attempt to learn these shortcut rules automatically, online. One possible source for new shortcut rules are plan improvement methods (Nakhost and Müller 2010; Chrpa, McCluskey, and Osborne 2012). However, these are designed to be used as a post-processing step, and are too slow to be used on every evaluated state. Additionally, these methods are based on a specified goal, while shortcut rules try to find shortcuts which lead to some $\exists$-opt landmark, without any specific goal to guide them.

We note that there is an interesting similarity between shortcut rules in planning and learned conflict clauses in SAT problems (Marques-Silva and Sakallah 1996). A shortcut rule can be seen as a "proof of suboptimality", demonstrating why some path can never be a prefix of an optimal solution, or, more generally, pose some constraints about the continuation of some path into an optimal solution. Similarly, a learned conflict clause can prune a partial assignment (that is, a path) in a SAT problem, or pose additional constraints on it. In SAT planning, online clause learning has increased the effectiveness of SAT solvers significantly (Marques-Silva and Sakallah 1996). Thus, we would expect that online learning of shortcut rules will, at the very least, result in a significant increase in the informativeness of a heuristic that is based upon the $\exists$-opt landmarks derived from them.

In this paper, we propose and examine three new types of shortcut rules, which are based upon online learning, during search. Our learning procedure exploits the fact that search often discovers several paths leading to the same state. When this occurs, we attempt to extract some relevant information about where these paths differ, and learn a shortcut rule for deriving more existential optimal landmarks from that information.

Surprisingly, our empirical results show that these more sophisticated shortcut rules result in a very small improvement in search guidance. Furthermore, and not that surprisingly, the overall number of problems solved under a time limit decreases, due to the increased computational cost per search node. However, we believe that additional work can place online shortcut learning at the state-of-the-art of cost-optimal planning, and leave our findings as a basis for future research.

Preliminaries

We consider planning tasks formulated in STRIPS with action costs; our notation mostly follows that of Helmert and Domshlak (2009). A planning task is described by a 5-tuple $\Pi = \langle P, A, \mathcal{C}, s_0, G \rangle$, where P is a set of propositions, A is a set of actions, each of which is a triple $a = \langle \mathsf{pre}(a), \mathsf{add}(a), \mathsf{del}(a) \rangle$, $\mathcal{C} : A \to \mathbb{R}^{0+}$ is a cost function on actions, $s_0 \subseteq P$ is the initial state, and $G \subseteq P$ is the goal.

An action a is applicable in state s if $\mathsf{pre}(a) \subseteq s$, and if applied in s, results in the state $s' = (s \setminus \mathsf{del}(a)) \cup \mathsf{add}(a)$. A sequence of actions $\langle a_0, a_1, \ldots, a_n \rangle$ is applicable in state s_0 if a_0 is applicable in s_0 and results in state s_1, a_1 is applicable in s_1 and results in s_2, and so on. The cost of action sequence $\pi = \langle a_0, a_1, \ldots, a_n \rangle$ is $\sum_{i=0}^{n} \mathcal{C}(a_i)$, and is denoted by $\mathcal{C}(\pi)$. The state resulting from applying action sequence π in state s is denoted by $s[\![\pi]\!]$. If π_1 and π_2 are action sequences, by $\pi_1 \cdot \pi_2$ we denote the concatenation of π_1 and π_2. Action sequence π is an s-path if it is applicable in state s, and it is also an s-plan if $G \subseteq s[\![\pi]\!]$. Optimal plans for Π are its cheapest s_0-plans, and the objective of cost-optimal planning is to find such an optimal plan for Π. We denote the cost of a cheapest s-plan by $h^*(s)$.

Let $\pi = \langle a_0, a_1, \ldots a_n \rangle$ be an s-path. The triple $\langle a_i, p, a_j \rangle$ forms a *causal link* (Tate 1977) in π if $i < j$, $p \in \mathsf{add}(a_i)$, $p \in \mathsf{pre}(a_j)$, and for $i < k < j$, $p \notin \mathsf{del}(a_k) \cup \mathsf{add}(a_k)$. In other words, a_i is the actual provider of precondition p for a_j. In such a causal link, a_i is called the *provider*, and a_j is called the *consumer*.

The *causal structure* of a given path π is a graph whose nodes are the action occurrences in π, and which has an edge from a_i to a_j if there is a causal link where a_i supports a_j. Figure 1 illustrates this, by showing the causal structure of the path $\langle drive(t_1, A, B),\ load(t_1, p_1, B),\ drive(t_1, B, A) \rangle$. The shortcut rules of Karpas and Domshlak (2012) look for certain patterns in the causal structure, and attempt to remove actions which fit these patterns.

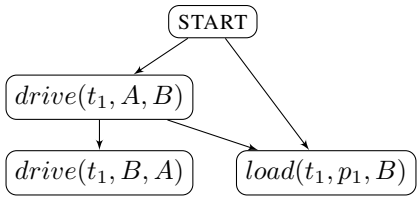

Figure 1: Causal structure of path $\langle drive(t_1, A, B),\ load(t_1, p_1, B),\ drive(t_1, B, A) \rangle$.

Learning Shortcut Rules

We now turn our attention to learning shortcut rules online. We begin by discussing when learning takes place, and then describe how the learning is actually done.

When to Learn

The purpose of a shortcut rule is to take a given path π, and produce a different path π', or several such paths, that are (a) cheaper than π, and (b) achieve at least one of the propositions that π achieves. Because each such path π' generates an $\exists$-opt landmark that consists of the facts that π achieved and π' did not, a good shortcut rule should generate a shortcut π' that achieves "almost all of" $s_0[\![\pi]\!]$. One extreme example of such a pair of paths is two paths π, π' that reach the same state.

We exploit the fact that pairs of paths that reach the same state are discovered during search, and we use these occasions to learn new shortcut rules. Recall that A^* handles the case of a cheaper path to a closed state s being discovered by reopening s, and this is one occasion when learning takes place. However, we can also learn whenever a more *expensive* path to a known state is discovered.

Algorithm 1 shows the pseudo-code of a slightly modified version of A^*. The notations $\hat{h}$ and $\hat{g}$ refer to the currently known heuristic estimate and cost-to-go, respectively, and $\hat{f} := \hat{g} + \hat{h}$. $Pa(s)$ is the current parent (state and action) of s, and $trace(s)$ is the currently best known path to s, obtained by following the parent pointers back until the initial state. Finally, h refers to the function which performs the actual computation of the heuristic estimate given a path.

The difference between *path-A^** (Karpas and Domshlak 2012) and A^* is that *path-A^** reevaluates the heuristic value of a state s, whenever a cheaper path to s is discovered (line 20). This is necessary to ensure optimality with a globally path-admissible heuristic. On top of that, learning *path-A^** attempts to learn a new shortcut rule whenever a new path to a known state is discovered (line 17), regardless of whether the new path is cheaper or not.

How to Learn

Having seen the context in which learning takes place, we now turn our attention to how learning works. The input to the LEARN method is a pair of different paths, π and π', from s_0 to the same state s. LEARN begins by building the causal structures of the two paths. Then, for each fact that holds in s, only the causally relevant part of the causal

Algorithm 1 Learning *path-A**

```
1  Closed ← ∅, Open ← ∅
2  ĝ(s_0) ← 0, ĥ(s_0) ← h(trace(s_0)), f̂(s_0) ← ĝ(s_0) + ĥ(s_0)
3  Open.insert(s_0)
4  while Open ≠ ∅ do
5      remove s with minimum f̂(s) from Open
6      if is_goal(s) then
7        return trace(s)
8      end if
9      Closed.insert(s)
10     for ⟨a, s'⟩ ∈ succ(s) do
11       if s' ∉ Closed ∪ Open then
12         ĝ(s') ← ĝ(s) + C(a), Pa(s') ← ⟨s, a⟩
13         ĥ(s') ← h(trace(s'))
14         f̂(s') ← ĝ(s') + ĥ(s')
15         Open.insert(s')
16       else
17         LEARN(trace(s) · ⟨a⟩, trace(s'))
18         if ĝ(s) + C(a) < ĝ(s') then
19           ĝ(s') ← ĝ(s) + C(a), Pa(s') ← ⟨s, a⟩
20           ĥ(s') ← h(trace(s'))
21           f̂(s') ← ĝ(s') + ĥ(s')
22           Open.insert(s')
23         end if
24       end if
25     end for
26 end while
27 return NO SOLUTION
```

structure is extracted; this is easily obtained by the transitive closure of edges, going backwards from the last action to achieve each fact.

Now we have, for each fact p in s, two causal structures which achieve p. These causal structures are similar to partially ordered plans, consisting of actions with causal links. However, we can not guarantee that all orderings that are consistent with these causal links will be valid plans, because causal structures do not account for threats — an action which could delete some precondition of another action, if applied in the wrong order. Nevertheless, we can guarantee that applying these actions according to the original order of the plan will generate a valid plan achieving p.

The simplest type of shortcut rule we consider, called *concrete shortcut rule*, exploits this fact. A concrete shortcut rule consists of a pair of action sequences, the head and the tail. Given two partial causal structures which achieve the same fact, we construct the two corresponding actions sequences, according to the order of the actions in the original plan. The cheaper action sequence π' becomes the tail, and the more expensive action sequence π becomes the head. In order for our shortcut rules to be more general, we trim the common prefix and suffix from the head and tail of the rule.

When a concrete shortcut rule $\pi \leftarrow \pi'$ is applied to some path ρ, it looks for π as a subsequence of ρ. Suppose $\rho = \rho_1 \cdot \pi \cdot \rho_2$, for some prefix ρ_1 and suffix ρ_2. Then the action sequence that is obtained from applying the concrete shortcut rule $\pi \leftarrow \pi'$ is simply $\rho' = \rho_1 \cdot \pi' \cdot \rho_2$. Since $\mathcal{C}(\pi') < \mathcal{C}(\pi)$, we know that $\mathcal{C}(\rho') < \mathcal{C}(\rho)$. However ρ' might not be applicable, because π' might not achieve all the facts necessary for ρ_2. Therefore, we apply actions by following ρ' as far as possible, until some action is no longer applicable, resulting in path ρ''. Clearly, $\mathcal{C}(\rho'') < \mathcal{C}(\rho')$, so ρ'' is indeed a shortcut, allowing us to generate some $\exists$-opt landmark. For example, if ρ achieved the propositions $\{x, y\}$ and ρ'' achieves $\{x, z\}$, then we can deduce that the possible consumers of $\{y\} = \{x, y\} \setminus \{x, z\}$ form an $\exists$-opt landmark of ρ.

The total order on the actions, however, might be too restrictive, as the actions in the shortcut rule might be applicable even if the order of the actions changes. Therefore, our second type of shortcut rule, called *unordered shortcut rule*, relaxes this total order. While we would like to use the partial order information from the causal structure, this makes reasoning about these shortcut rules much more complicated. Therefore, we completely ignore any information about ordering between actions, and represent the head and tail of an unordered shortcut rule as sets of actions. The learning procedure for unordered shortcut rules is the same as for concrete shortcut rules, except that we add a final stage, where we convert the action sequences into sets of actions.

When such an unordered shortcut rule $A_1 \leftarrow A_2$ is applied to path ρ, we first check whether A_1 is a subset of the actions in ρ. If so, we remove these actions from ρ, and start applying the actions from ρ, in order, until we reach the first action that was removed. From this point, we attempt to apply either an action from A_2 that was not applied already, or, if no such action is applicable, the next action from ρ. This process terminates when there are no more applicable actions, and generates the action sequence ρ'. Since $\mathcal{C}(A_1) < \mathcal{C}(A_2)$, ρ' is a shortcut, generating an $\exists$-opt landmark. For example, assume that action a_1 achieves $\{x\}$, a_2 achieves $\{y\}$, and a_3 achieves $\{x, y\}$. Then we could learn the unordered shortcut rule $\{a_1, a_2\} \leftarrow \{a_3\}$, and for any action sequence containing both a_1 and a_2, in any order, we would attempt to remove them, and add a_3 instead.

While unordered shortcut rules are more general than concrete shortcut rules, we can exploit the fact that planning problems are typically described concisely in PDDL, and specifically the fact that actions are defined by an operator type with a list of arguments. Our final shortcut rule uses this structure, and attempts to generalize concrete shortcut rules. Consider for example, the concrete shortcut rule $\langle$drive(t_1, A, B), drive$(t_1, B, C)\rangle \leftarrow \langle$drive$(t_1, A, C)\rangle$, learned when truck t_1 drove the long way from location A to location C. This rule can be generalized to specify that any truck should not drive the long way between any two locations, written here as: $\langle$drive$(?t, ?X, ?Y)$, drive$(?t, ?Y, ?Z)\rangle \leftarrow \langle$drive$(?t, ?X, ?Z)\rangle$.

Such *generalized shortcut rules* are learned the same way as concrete shortcut rules. However, instead of treating the action sequences as sequences of ground actions, we treat them as sequences of terms, similar to terms in first-order logic. When a generalized shortcut rule $\pi \leftarrow \pi'$ is applied to path ρ, we look for a subsequence of ρ which matches the

operator types in π, disregarding any action parameters. If such a subsequence is found, we then attempt to unify π with the subsequence. If this is not possible, the shortcut rule is not applicable. Otherwise, we have a substitution θ, which we apply to π'. Then, we attempt to apply actions from ρ without the subsequence, and then from π' and the rest of ρ, in a similar manner to unordered shortcut rules.

Using the Learned Knowledge

Having described how and when shortcut rules are learned, we must still use them. The learned shortcut rules are used whenever a path is evaluated (lines 13 and 20 in Algorithm 1). For each such evaluated path, we test all learned shortcut rules for applicability. However, this testing is fairly expensive, and it is quite possible that the cost of testing whether a shortcut rule is applicable could outweigh its benefits in terms of increased accuracy — this is known as the utility problem (Minton 1990).

Therefore, we keep track of how many times each shortcut rule has been used, and of how many times each shortcut rule produced a valid shortcut. If a rule has been tried many times, but produced very few valid shortcuts, we erase this low-utility rule. The exact numeric values controlling this behavior are parameters of the learning method.

Empirical Evaluation

In order to evaluate how effective the learned shortcut rules are, we implemented all three learning schemes on top of the Fast Downward planner (Helmert 2006). We base all of our experiments on the admissible landmarks heuristic (Karpas and Domshlak 2009) using an optimal cost partitioning over all regular single fact landmarks (Keyder, Richter, and Helmert 2010) and the $\exists$-opt landmarks derived from the original shortcut rules (Karpas and Domshlak 2012). We compare four different variants of the landmarks heuristic, differing in the type of learned shortcut rules they employ.

- none — no online learning
- concrete — concrete shortcut rules
- unordered — unordered shortcut rules
- generalized — generalized shortcut rules

All of the experiments reported here were run on a single core of an Intel E8400 CPU, with a time limit of 30 minutes and a memory limit of 6 GB, on a 64-bit linux OS.

Table 1a shows the number of problems solved in each domain by learning *path-A** using each of the above heuristics. These results show that, indeed, the overhead of learning shortcut rules online is significant, and using them reduces the number of problems solved in total. While there is no clear winner between the concrete shortcut rules and unordered shortcut rules, generalized shortcut rules fare the worst, because the overhead of unification and substitution is quite significant.

The reduction in the number of problems solved might be due to an inefficient implementation of the shortcut rules. We therefore examine a measure which is relatively independent of such concerns — the number of states expanded by the search algorithm. Here, we only consider the problems that were solved using all four heuristics.

Table 1b lists the total number of states expanded to solve all the problems that were solved using all four heuristics in each domain. While we would expect the more sophisticated shortcut rules to lead to a substantial improvement in search guidance, the results tell a different tale. Although there is a small decrease in the number of expanded states, it is not very significant. In many domains, the learned shortcut rules do not increase informativeness at all, and on average, all of the new shortcut rules reduce the number of expanded states by about 1%. This slight increase in informativeness is not enough to compensate for the increase in computation time, thus partly explaining the results in Table 1a.

While these results are quite surprising, we believe that it should be possible to learn effective shortcut rules online, and that this could lead to state-of-the-art performance in optimal planning. However, at the moment, the simple shortcut rules of Karpas and Domshlak (2012) appear to result in the best trade-off between heuristic computation time and heuristic guidance.

Acknowledgements

This work was carried out in and supported by the Technion-Microsoft Electronic-Commerce Research Center. We thank Malte Helmert and the anonymous reviewers for discussions and helpful advice.

References

Ambite, J. L., and Knoblock, C. A. 2001. Planning by rewriting. *Journal of Artificial Intelligence Research* 15:207–261.

Chrpa, L.; McCluskey, T. L.; and Osborne, H. 2012. Optimizing plans through analysis of action dependencies and independencies. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS 2012)*. AAAI Press.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In Gerevini, A.; Howe, A.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 162–169. AAAI Press.

Helmert, M. 2006. The Fast Downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

Karpas, E., and Domshlak, C. 2009. Cost-optimal planning with landmarks. In Boutilier, C., ed., *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, 1728–1733.

Karpas, E., and Domshlak, C. 2012. Optimal search with inadmissible heuristics. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS 2012)*. AAAI Press.

Keyder, E.; Richter, S.; and Helmert, M. 2010. Sound and complete landmarks for and/or graphs. In Coelho, H.; Studer, R.; and Wooldridge, M., eds., *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI 2010)*, 335–340. IOS Press.

coverage	none	concrete	unordered	generalized
airport (50)	**26**	**26**	**26**	25
blocks (35)	**26**	21	18	16
depot (22)	**6**	5	5	2
driverlog (20)	**10**	9	8	8
elevators-opt08-strips (30)	**10**	8	6	3
elevators-opt11-strips (20)	**8**	6	4	1
floortile-opt11-strips (20)	**2**	**2**	1	0
freecell (80)	**50**	48	47	34
grid (5)	**2**	**2**	1	1
gripper (20)	**6**	5	5	4
logistics00 (28)	20	20	20	20
logistics98 (35)	**4**	3	**4**	3
miconic (150)	**141**	**141**	140	140
mprime (35)	**18**	**18**	17	15
mystery (30)	15	15	15	15
nomystery-opt11-strips (20)	**18**	**18**	**18**	14
openstacks-opt08-strips (30)	**14**	11	**14**	6
openstacks-opt11-strips (20)	**9**	6	**9**	1
parcprinter-08-strips (30)	**13**	12	**13**	12
parcprinter-opt11-strips (20)	**9**	8	**9**	8
parking-opt11-strips (20)	**1**	**1**	**1**	0
pathways (30)	4	4	4	4
pegsol-08-strips (30)	**26**	**26**	19	9
pegsol-opt11-strips (20)	**16**	**16**	8	1
pipesworld-notankage (50)	**14**	13	13	11
pipesworld-tankage (50)	**8**	**8**	**8**	7
psr-small (50)	**48**	**48**	**48**	46
rovers (40)	**6**	5	**6**	5
scanalyzer-08-strips (30)	**14**	13	13	12
scanalyzer-opt11-strips (20)	**11**	10	10	9
sokoban-opt08-strips (30)	**16**	6	6	3
sokoban-opt11-strips (20)	**13**	3	3	1
storage (30)	**14**	**14**	12	11
tidybot-opt11-strips (20)	**11**	10	9	4
tpp (30)	**6**	**6**	**6**	5
transport-opt08-strips (30)	**9**	**9**	7	5
transport-opt11-strips (20)	**4**	**4**	2	0
visitall-opt11-strips (20)	**12**	**12**	11	9
woodworking-opt08-strips (30)	**13**	11	12	11
woodworking-opt11-strips (20)	**8**	6	7	6
SUM (1310)	**661**	609	585	487

(a) Number of Problems Solved in Each Domain

expansions	none	concrete	unordered	generalized
airport (25)	50136	50136	50136	50136
blocks (16)	17712	16616	**16203**	16446
depot (2)	1016	1016	1016	1016
driverlog (8)	375488	**372859**	373260	375424
elevators-opt08-strips (3)	56436	56436	56436	56436
elevators-opt11-strips (1)	38125	38125	38125	38125
floortile-opt11-strips (0)	N/A	N/A	N/A	N/A
freecell (34)	10940	10940	10940	10940
grid (1)	141	141	**126**	133
gripper (4)	81988	81988	81988	81988
logistics00 (20)	816589	816589	816589	816589
logistics98 (3)	13227	13227	13227	13227
miconic (140)	48483	48483	48483	48483
mprime (15)	20694	17400	**16662**	20024
mystery (17)	96186	101899	**92983**	93873
nomystery-opt11-strips (14)	4778	4778	4778	4778
openstacks-opt08-strips (6)	31279	31279	31279	31279
openstacks-opt11-strips (1)	3658	3658	3658	3658
parcprinter-08-strips (12)	735545	732904	**730269**	735545
parcprinter-opt11-strips (8)	735473	732832	**730197**	735473
parking-opt11-strips (0)	N/A	N/A	N/A	N/A
pathways (4)	58156	58165	**58089**	58156
pegsol-08-strips (9)	9783	9783	**9715**	9783
pegsol-opt11-strips (1)	246	246	246	246
pipesworld-notankage (11)	33615	**32772**	32952	33408
pipesworld-tankage (7)	8875	**7904**	8285	8804
psr-small (46)	202184	**192403**	193633	198306
rovers (5)	**98776**	99474	99516	99566
scanalyzer-08-strips (12)	4564	4564	4564	4564
scanalyzer-opt11-strips (9)	4545	4545	4545	4545
sokoban-opt08-strips (3)	701	701	701	701
sokoban-opt11-strips (1)	51	51	51	51
storage (11)	20680	**20548**	20593	20644
tidybot-opt11-strips (4)	**4827**	4903	4867	4899
tpp (5)	4227	4227	4227	4227
transport-opt08-strips (5)	9510	9487	**9464**	9506
transport-opt11-strips (0)	N/A	N/A	N/A	N/A
visitall-opt11-strips (9)	4217	4217	4217	4217
woodworking-opt08-strips (11)	92184	87187	**82786**	92182
woodworking-opt11-strips (6)	90482	85559	**81246**	90482
SUM (489)	3785517	3758042	**3736052**	3777860

(b) Total Number of Expanded States Over Problems Solved by All

Table 1: Empirical Results

Marques-Silva, J. P., and Sakallah, K. A. 1996. GRASP - a new search algorithm for satisfiability. In *Proceedings of the 1996 IEEE/ACM International Conference on Computer-Aided Design (ICCAD 1996)*, 220–227.

Minton, S. 1990. Quantitative results concerning the utility of explanation-based learning. *Artificial Intelligence* 42(23):363–391.

Nakhost, H., and Müller, M. 2010. Action elimination and plan neighborhood graph search: Two algorithms for plan improvement. In Brafman, R.; Geffner, H.; Hoffmann, J.; and Kautz, H., eds., *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling (ICAPS 2010)*, 121–128. AAAI Press.

Tate, A. 1977. Generating project networks. In Reddy, R., ed., *Proceedings of the 5th International Joint Conference on Artificial Intelligence (IJCAI 1977)*, 888–893. William Kaufmann.