Tunneling and Decomposition-Based State Reduction for Optimal Planning

Raz Nissim and **Udi Apsel** and **Ronen Brafman** [1]

Abstract. Action pruning is one of the most basic techniques for improving a planner's performance. The challenge of preserving optimality while reducing the state space has been addressed by several methods in recent years. In this paper we describe two optimality preserving pruning methods: The first is a generalization of tunnel macros. The second, the main contribution of this paper, is a novel partition-based pruning method. The latter requires the introduction of new automated domain decomposition techniques which are of independent interest. Both methods prune the actions applicable at state s based on the last action leading to s, and both attempt to capture the intuition that, when possible, we should focus on one subgoal at a time. As we demonstrate, neither method dominates the other, and a combination of both allows us to obtain an even stronger pruning rule. We also introduce a few modifications to A* that utilize properties shared by both methods to find an optimal plan. Our empirical evaluation compares the pruning power of the two methods and their combination, showing good coverage, reduction in running time, and reduction in the number of expansions.

1 INTRODUCTION

Action pruning is one of the most basic techniques for improving a planner's performance. By considering only a subset of the applicable actions at a state, one can focus on a subset of the possible search paths, and in some cases, eliminate some search nodes. If the pruning technique is fast and effective, this can lead to significant speed-up, as well. One of the most influential and effective pruning techniques is FF's *helpful actions* [11]. However, FF's helpful actions pruning is heuristic, and sacrifices optimality and completeness. A number of other pruning techniques maintain completeness and optimality, among these are commutativity pruning [15, 8], tunneling [5], and other methods based on the principle of stubborn sets [4, 19]. Our work falls into this category, and specifically, into the class of state reduction pruning techniques.

In this paper, we present two new pruning techniques that share similar intuitions. First, we provide a slight generalization of the domain-independent *tunnel-macros* technique introduced by Coles and Coles [5] which was motivated by tunnel macros in Sokoban [13]. Very roughly, tunnel pruning is based on the following idea: suppose that we just applied an action a that only changes the value of some variable p to v_1. Further, suppose that p's goal value is not v_1, and that any action that changes p from v_1 to some other value does not affect other variables. Then, after a, we might as well apply one of those actions that changes p. Our generalized tunneling applies this idea to actions that affect more than a single variable.

[1] Ben-Gurion University of the Negev, Israel, email: raznis,apsel,brafman@cs.bgu.ac.il

The second and more significant contribution, is a new partition-based pruning technique. This technique requires as input a partition of the set of actions. Given such a partition, it allows us to prune actions roughly as follows: after performing an action a, if this action affects only actions from its own partition (i.e., by supplying or destroying their pre or prevail conditions) then the next action should be from that same partition. The power of this method depends on the quality of the partition used. In some domains natural partitions suggest themselves. For example, when there is a natural notion of agents in a domain (e.g., trucks in Logistics, satellites in Satellite, etc.), then it is natural to partition the set of actions to sets consisting of actions involving a specific agent. However, most domains offer no obvious partition, and hence one of our contributions is a principled and efficient automated domain decomposition method.

Both the above methods prune according to the last action leading to the current state. Hence, to yield an optimal plan, slight modifications of A* are required, both to maintain this extra information about a state, as well as to properly address the issue of duplicate detection in this context. With these modifications, A* combined with our pruning rules (and in fact, a general class of path-based pruning rules) finds an optimal plan, when one exists.

Given a number of pruning techniques, it is natural to ask whether one dominates the other. We show that, despite some similarities, generalized tunneling and partition-based pruning do not dominate each other, but cannot be used together without sacrificing optimality. However, we describe an optimality preserving pruning rule that combines elements of both methods. Lastly, we provide an empirical analysis across various planning domains, comparing our pruning methods to a baseline planner. Our results show an increase in coverage, as well as a reduction in running time and node expansion.

2 BACKGROUND/PRELIMINARIES

This paper considers cost-optimal planning, using the SAS+ formalism [1]. Such a planning problem is defined as a tuple $\Pi =< \mathcal{V}, s_0, s_\star, \mathcal{A} >$ where $\mathcal{V}$ is a set of **variables** which can be assigned a value from a finite domain D_v, s_0 is the **initial state**, $s_\star$ is a partial assignment of $\mathcal{G} \subseteq \mathcal{V}$ denoting the **goal** conditions, and $\mathcal{A}$ is the set of **actions**, where each $a =< \mathit{pre}, \mathit{prevail}, \mathit{eff} > \in \mathcal{A}$ is given by its preconditions, prevail conditions (required, but not affected by the action) and effects. For convenience, if an action has (v, p) as a precondition, and (v, p') as an effect, we refer to it as having a *pre-post* condition (v, p, p'). An action sequence π is **legal** if the preconditions of every action in π hold when executing π starting at s_0 and is called a **plan** if its execution ends in a goal state. A **path** P is a sequence of states induced by a legal sequence of actions, where each state is associated with the action leading to its generation.

Two actions a, a' are **commutative** [8] if neither one achieves or destroys a precondition of the other, and they don't have conflicting effects. By this definition, in any legal plan π having two consecutive *commutative* actions a_i, a_{i+1}, the permutation π' of π where the order of a_i, a_{i+1} is switched, is an effect-equivalent legal plan.

Our pruning methods require that the actions of a SAS+ problem be partitioned, i.e. mapped to k disjoint sets. A **partitioned SAS+ problem** is therefore defined as $\Pi =< \mathcal{V}, s_0, s_\star, \{\mathcal{A}_i\}_{i=1}^k >$ where for $1 \leq i \leq k$, $\mathcal{A}_i$ is the set of actions in partition i. Given such a partition, we can now distinguish between **private** and **public** actions – public actions are the ones that are not commutative with actions in other partitions, while all other actions are private. A value $p \in D_v$ of variable v is said to be **private** if it is required and achieved only by actions of a single partition, and **public** otherwise. We note that for ease of presentation, all actions which achieve some variable's goal value are considered public. This is assumed throughout the paper, but our techniques are easily modified to remove it.

To get a clearer picture of a partitioned SAS+ problem and the dependencies induced by it, consider the well known Logistics planning domain, in which a set of packages should be moved on a road-map from their initial to their target locations using a given fleet of vehicles such as trucks, airplanes, etc. The packages can be loaded onto and unloaded off the vehicles, and each vehicle can move along a certain subset of road segments. Possible actions are therefore *move, load, and unload*. Associating each vehicle with a partition, we might map all *move, load* and *unload* actions in which it is involved to a single partition. Now, since a vehicle's location can only be changed by its *move* actions, all the *move* actions are *private*. On the other hand, *load/unload* actions are *public* only if they affect the position of a package in some of its public locations, i.e., locations that can be reached by at least two vehicles.

We note that the notion of partitioning actions and the distinction between private and public actions was presented in the context of multi-agent STRIPS (MA-STRIPS) planning by Brafman and Domshlak [2]. The multi-agent aspects of MA-STRIPS are not of relevance to this work, but the partition of actions into disjoint groups given by MA-STRIPS is essential for our pruning methods, so we adapted these definitions for SAS+ planning.

Finally, we introduce a property relevant to pruning methods, which we use throughout the paper. Pruning method ρ is **optimality preserving** if for every SAS+ problem Π and for every optimal plan π, there exists another, effect-equivalent optimal plan π', which is not pruned by ρ. The three pruning methods we shall describe are optimality preserving since for any legal plan π, an effect equivalent permutation of π which is not ρ-pruned exists. In Section 6 we show that a slightly modified version of A* which uses an optimality preserving pruning method is complete and optimal.

3 ACTION TUNNELING

Tunnel macros were first described as part of a Sokoban solver [13]. The idea here is to group related atomic actions into a single macro, or composite action. In Sokoban, a tunnel is a part of the grid where the maneuverability of the player is restricted to a width of one. When pushing a box into a tunnel, assuming all points along the tunnel are indistinguishable, the only sensible choice is to push the box to its end, since this would have to be done eventually if it is part of an optimal plan. Therefore, all *push* actions along the tunnel can be treated as a *macro*, and heuristic evaluation and decision making in intermediate locations is not necessary.

Coles and Coles [5] provided the first domain-independent generalization of the tunnel-macro idea focused on actions with a single effect. We now present a generalization of this definition to arbitrary operators.

For each action $a =< \textit{pre}, \textit{prevail}, \textit{eff} >$, we define $s_{\textit{min}}(a) = \{\textit{prevail} \cup \textit{eff}\}$, i.e. $s_{\textit{min}}(a)$ is the minimal partial assignment which always holds following the execution of a. Action *a allows a tunnel* if the following conditions hold:

1. $\exists (v_i, p) \in \textit{eff}$ such that $(v_i, p') \in s_\star$ and $p \neq p'$.
2. Any action having a prevail $(v_i, p_i) \in \textit{eff}$, has some pre-post condition (v_j, p_j, p'_j) such that $(v_j, p_j) \in \textit{eff}$.
3. For any action a' having a *pre-post* condition (v_i, p_i, p'_i) where $(v_i, p_i) \in \textit{eff}$, then:
   - the preconditions of a' are satisfied by $s_{\textit{min}}(a)$.
   - a' affects only the variables affected by a.[2]

If a allows a tunnel, we define *tunnel*(a) to be all operators which have a pre-post condition $< v_i, p_i, p'_i >$ or prevail condition $< v_i, p_i >$ where $(v_i, p_i) \in \textit{eff}$. Otherwise, *tunnel*$(a) = \mathcal{A}$.

Pruning rule 1 (Action Tunneling Pruning). *Following action a, prune all actions not in tunnel*(a).

A valid sequence of actions $\pi = (a_1, a_2 \ldots, a_k)$ is said to be **tunnel-pruned** if for some action a_i, $a_{i+1} \notin \textit{tunnel}(a_i)$.

Lemma 1. *Action tunneling pruning is an optimality-preserving pruning method.*

Proof. Let Π be a SAS+ problem for which an optimal solution $\pi = (a_1, a_2, \ldots, a_k)$ exists. If π is not tunnel-pruned, the lemma trivially holds. Otherwise, consider the first occurrence of an action pair a_i, a_{i+1}, such that $a_{i+1} \notin \textit{tunnel}(a_i)$. Given the conditions on a_i, we know that one of the variables v it affects does not have its goal value after applying a_i. Consequently, there exists an action appearing after a_i in P that changes v's value. Let a_j be the first such action in the plan following a_i. Consider the actions $a_{i+1}, \ldots a_{j-1}$. By assumption, none of them changes the value of a_i's effects. By the definition, this means that none of them require any of a_i's effects, either as preconditions or prevail conditions. By definition, a_j cannot change any variable that is not affected by a_i, and a_j is applicable in $s_{\textit{min}}(a_i)$. Therefore, a_j can be applied immediately after a_i, and $a_{i+1}, \ldots a_{j-1}$ are executable after a_j. This means that $\pi' = (a_1, \ldots a_i, a_j, a_{i+1} \ldots, a_{j-1}, a_{j+1}, \ldots, a_k)$ is a legal plan that is effect-equivalent to π.

By applying this procedure repeatedly, we obtain an optimal, effect-equivalent permutation of π which is not tunnel-pruned. □

4 PARTITION-BASED PATH PRUNING

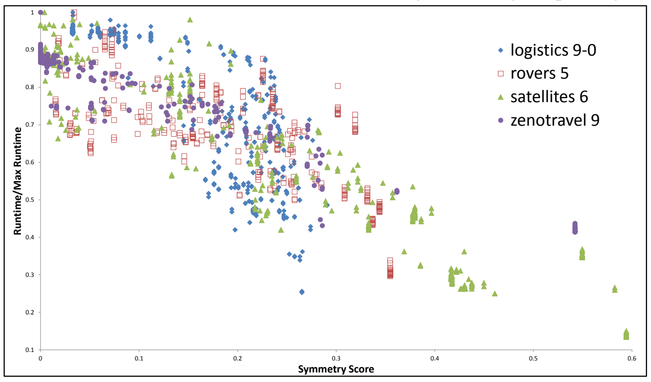
Given a SAS+ planning problem and a partition of its actions, we propose our pruning rule, *partition-based (PB) pruning*:

Pruning rule 2 (Partition-Based Pruning). *Following a private action $a \in \mathcal{A}_i$, prune all actions not in $\mathcal{A}_i$.*

A valid sequence of actions $P = (a_1, a_2 \ldots, a_k)$ is said to be **PB-pruned** if for some *private* action $a_i \in \mathcal{A}_i$, $a_{i+1} \notin \mathcal{A}_i$.

[2] Coles and Coles allow a' to affect irrelevant resources. However, these are essentially variables that can be removed from the problem description in preprocessing without affecting correctness.

Figure 1. Correlation between Γ and running time using PB pruning.



Lemma 2. *PB pruning is an optimality-preserving pruning method.*

Proof. Let Π be a SAS+ planning problem, let $\bigcup_i \mathcal{A}_i = \mathcal{A}$ be a partition of its actions, and let π be an optimal plan for Π. By definition of private actions, we know that private actions are commutative with actions of other partitions. Therefore, we can move any ordered sequence of private actions of $\mathcal{A}_i$, so that it would be immediately before the subsequent public action of $\mathcal{A}_i$, and maintain the legality and the effect of the plan. Doing this for all ordered sequences of private actions, will produce a non-PB-pruned plan which is a permutation of π, and therefore optimal as well. $\square$

As an example, consider a logistics problem consisting of two trucks, and a partition of the actions such that for $i \in \{1, 2\}$, all actions of $truck_i$ are in $\mathcal{A}_i$. When expanding state s, for which the creating action was $a = move(truck_1, loc_1, loc_2)$, all actions corresponding to $truck_2$ are pruned. Since a was performed in order to achieve some precondition for $truck_1$'s public *load/unload* action at loc_2, PB pruning focuses the search effort on applying that action.

Generally, in optimal plans, private actions are executed only to enable some public action, since otherwise, the private action can be removed and the plan is not optimal[3]. Therefore, when pruning the actions of other partitions after applying a private action a, the search effort focuses on achieving the cause for applying a in the first place. The reader may have observed that in optimal search in general, when reaching state s via a private action $a \in \mathcal{A}_i$, all public actions of $\mathcal{A}_i$ constitute a disjunctive action landmark in state s.

We note that our logistics example exhibits multi-agent structure, and therefore, a natural decomposition into partitions (*vehicle=partition*). Although this structure is evident in some benchmark planning domains (e.g. Logistics, Rovers, Satellites, Zenotravel etc.), in general there isn't always an obvious way of decomposing the problem. In what follows, we describe an automated method for decomposing a general planning problem, making PB pruning applicable in the general setting.

4.1 Decomposition

We now discuss the important question of *how* to decompose a given planning problem Π. Since not all planning problems exhibit "multi-agent" structure some general method of problem decomposition is required. We now describe one such method.

Given a SAS+ planning problem Π, we define the **action graph** (AG) of Π. The nodes of AG correspond to the actions in $\mathcal{A}$. There is an undirected edge between actions a_1, a_2 if they are not commutative, i.e. if one achieves or destroys a precondition of the other, or if they have a conflicting effect. A **partition** of AG maps all nodes of AG to $k > 1$ *disjoint* sets $\mathcal{A}_i$ such that $\cup_{1 \leq i \leq k} \mathcal{A}_i = \mathcal{A}$, clearly inducing a *partitioned* SAS+ *problem*. Moreover, the distinction between public and private actions is immediate – action $a_i \in \mathcal{A}_i$ is public if there exists an edge (a_i, a_j) for some $a_j \in \mathcal{A}_j, i \neq j$, while all other actions are defined as private. In other words, $a_i \in \mathcal{A}_i$ is private if it is connected only to actions belonging to $\mathcal{A}_i$, and public otherwise.

As an exponential number of partitions exist, some measure of partition quality is required. Using PB pruning, action pruning is performed only when the first of two consecutive actions is private and the second is one belonging to a different partition. We introduce the notion of **symmetry score** (Γ), which measures the probability of such a sequence appearing in the search:

$$\Gamma(\{\mathcal{A}\}_{i=1}^{k}) = \sum_{i=1}^{k} (pr(a \in \mathcal{A}_i \text{ and } a \text{ is private}) * pr(a \notin \mathcal{A}_i))$$

Γ is, of course, an approximation, since it regards the probability of each action appearing at any point in the search as equal. On two extremes, Γ is equal to zero:

1. If each action is mapped to a different partition ($k = |\mathcal{A}|$), and assuming the action graph is connected, there are no private actions and the first term in the multiplication is always zero.
2. If all actions belong to a single partition ($\mathcal{A}_1 = \mathcal{A}$), then the second term is always zero.

In both cases, PB pruning will, in fact, be useless. On the other hand, if the actions are distributed evenly between k partitions which have no effect on one another, then the symmetry score will be $k * (1/k) * (k - 1)/k$, which approaches 1 as k rises. In this case, PB pruning will reduce the search space dramatically, as many effect-equivalent action sequences exist.

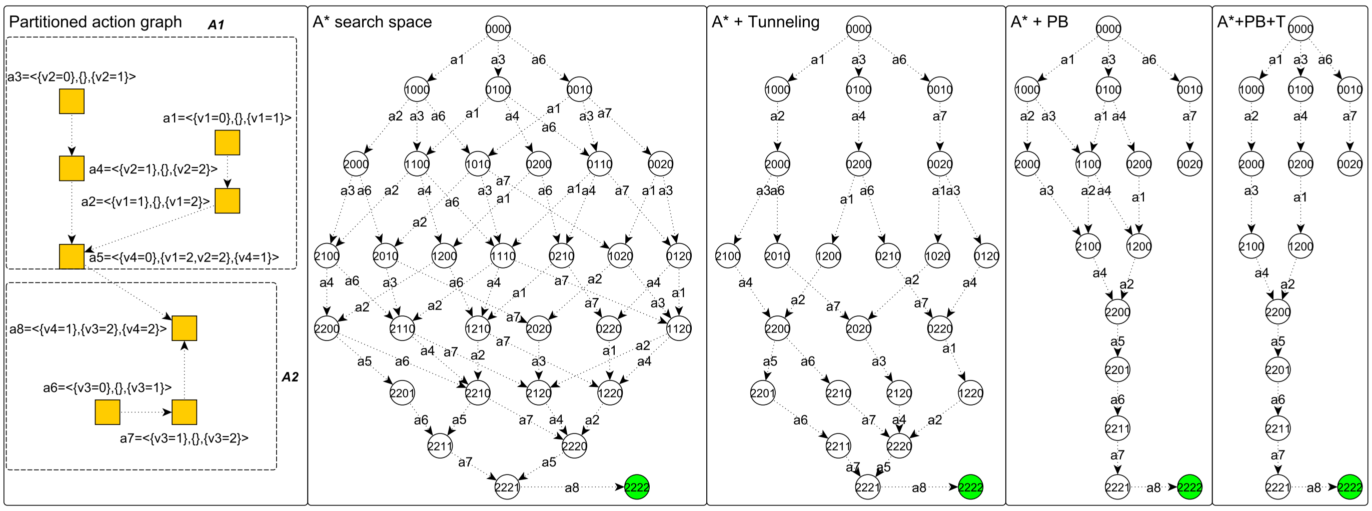
In order to verify that there exists a correlation between the symmetry score Γ of a partition and the effectiveness of PB pruning in practice, we conducted an empirical study. Figure 1 depicts the running time of our slightly modified version of A* (presented in Section 6) with the $h_{lm\text{-}cut}$ heuristic [10], using PB pruning, on four benchmark planning problems. Each problem was solved 400 times using different partitions with varying symmetry score. We used the METIS package [14] as our graph partitioning software. METIS is a fast and effective graph partitioning tool, and was also used in the experiments in Section 7. Since Γ is only an approximation, higher Γ values do not necessarily mean the problem is solved faster. However, in general we observe that running time decreases as Γ rises. Satellites 6, for example, was solved almost $\times 10$ faster when using PB pruning with the best partition.

Interestingly, in the case of the Rovers and Satellites domains, the best partitions obtained correspond exactly to the natural multi-agent (MA) structure of the problem. Problems of both these domains represent loosely-coupled MA systems, in which many private actions exist. In other cases, such as Logistics, however, the best partitions found had higher symmetry score than the natural MA formulations. This is due to the tightly-coupled MA systems that are represented as Logistics problems. In these problems, an artificial partition of the actions, which may group together a few (natural) agents as one, can achieve a higher symmetry score.

We note that Γ is not a perfect quality measure, but rather a rough approximation. Its main deficiency lies in the fact that it assumes a uniform distribution of action application during the search. As this is a phenomenon which rarely occurs, Γ is far from optimal. However, it is simple and intuitive, and our experiments seem to validate its utility. Finding more accurate measures remains an open challenge.

[3] This is assuming, as we do, that all goal-achieving actions are public.

Figure 2. Action graph of an example planning problem and search space of A*with tunneling and partition-based pruning. Actions are represented as $< pre, prevail, eff >$ and states are denoted by the values of variables v_1, v_2, v_3 respectively (112 denotes the state where $v_1 = 1, v_2 = 1, v_3 = 2$.)



5 PB-PRUNING AND TUNNELING

Intuitively, tunneling and PB pruning seem similar. Both are based on the idea that search should aim to "finish what it started", i.e. after performing an action aimed at eventually achieving some variable value, focus the search effort only on achieving it. Both methods are state reduction techniques, which reduce the number of reachable states as well as transitions, unlike other partial order reduction techniques such as *sleep sets*, *commutativity pruning* and *stratified planning*, which reduce the number of state *transitions* but not the number of reachable states.

Although based on a similar intuition, PB and tunnel pruning do *not* prune the same set of states, nor can it be said that one method strictly dominates the other[4].

To better understand the differences between the methods, consider the example in Figure 2. In this example, the actions $a_1, \dots a_8$ are divided between two partitions A_1, A_2, such that a_5 and a_8 are public, and the rest are private. Additionally, there are 3 actions a_1, a_3, a_6 after which we can tunnel. $s_0 = 0000$ and $s_\star = (v_4, 2)$. We first address the state reduction obtained by action tunneling. Consider states 1000 and 0010, both created by an action after which we can tunnel. In these states, action tunneling will prune actions a_6, a_1, respectively. Since these two actions are the only creating operators of state 1010, that state is never generated. Altogether, 7 out of 31 states are pruned using action tunneling. Now consider the search space when using PB pruning. State 0020, created by a private action $a_7 \in A_2$, may only expand using actions from A_2. Since there are no such applicable actions (which do not constitute a self-loop), 0020 effectively becomes a dead-end, never generating states 1020 and 1120. Notice that when reaching state 2200, the only state with $g = 4$, the search space is reduced to a single path leading directly to the goal. In total, 16 out of 31 states are never generated.

Although the two pruning rules seem to be applicable together, combining them, as presented, results in a pruning method which is not optimality-preserving. As an example, consider planning problem Π, for which a_i, a_{i+1} are two consecutive actions in a non-PB-pruned optimal plan, such that $a_i \in \mathcal{A}_i$ is a public action, affecting a single *private* variable value and having a public prevail condition, and such that a_i has the private action $a_k \in \mathcal{A}_i$ in its tunnel. Then, if $a_{i+1} \in \mathcal{A}_j$, $j \neq i$, moving a_k between a_i and a_{i+1} would not preserve the PB pruning rule. There is a way, however, to combine the two pruning rules and maintain optimality:

Pruning rule 3 (Tunnel+PB Pruning)**.** *Always apply pruning rule 2, and apply pruning rule 1 only if the creating action is private.*

This combined pruning rule yields non-PB-pruned plans, such that internally, the private action sequences of every partition are not tunnel-pruned. Clearly, this dominates PB pruning when using the same partition of $\mathcal{A}$.

Lemma 3. *Tunnel+PB pruning is an optimality-preserving pruning method.*

Proof. Let Π be a partitioned SAS+ planning problem, and let π be an optimal plan for Π. By Lemma 2 there exists an optimal plan π' which is not PB-pruned . If π' is not tunnel-pruned, the lemma trivially holds. Otherwise, consider the first occurrence in π' of an action pair a_i, a_{i+1}, such that $a_{i+1} \notin tunnel(a_i)$. By definition, $a_i \in \mathcal{A}_i$ is private, so it affects and requires only private variable values. Therefore, any action in its tunnel must be in $\mathcal{A}_i$. Since one of the variables v does not have a goal value after applying a_i, there exists another action in π' appearing after a_i that changes v's value. Let $a_j \in \mathcal{A}_i$ be the first such action. Similarly to Lemma 1, a_j can be applied directly after a_i without effecting the legality and outcome of the plan. Moreover, since a_i, a_j belong to the same partition, this new plan is not PB-pruned.

By applying this procedure repeatedly, we obtain a non-tunnel-pruned plan, which is a permutation of π, and therefore optimal. □

Going back to the rightmost part of Figure 2, we see that performing tunneling after private actions can lead to further state reduction. Specifically, in states 1000 and 0100, actions a_1 and a_3 are pruned respectively, and state 1100 is never generated.

6 PATH PRUNING A*

In this section we present a slight modification of the A* algorithm, which allows the application of optimality preserving pruning methods for the purpose of optimal planning. The path pruning A*, aka PP-A*, is a search algorithm which receives a planning problem Π

[4] In general, it is difficult to discuss dominance of PB pruning, since the effect of PB pruning is dependent on the partition of the actions.

and a pruning method ρ as input, and produces a plan π, which is guaranteed to be optimal provided that ρ respects the following properties: (i) ρ is optimality preserving, and (ii) ρ prunes only according to the last action. It is easy to see, for example, that both PB pruning and tunnel pruning respect the second condition, since both pruning rules fire only according to the last action. Of course, both pruning methods are also optimality preserving, and can therefore be easily integrated in PP-A*, as we later demonstrate.

6.1 PP-A* versus A*

PP-A* is identical to A* except for the following three changes. **First**, a different data-type is used for recording an open node. In PP-A*, an open list node is a pair (A, s), where s is the state and A is a set of actions, recording various possible ways to reach s from a previous state. **Second**, node expansion is subject to the pruning rules of method ρ. Namely, PP-A* executes an applicable action a' in state s only if there is at least one action $a \in A$ s.t. the execution of a' is allowed after a under ρ's pruning rules. **Third**, duplicate states are handled differently. In A*, when a state s which is already open is reached by another search path, the open list node is updated with the action of the lower g value, and in case of a tie – drops the competing path. In contrast, ties in PP-A* are handled by preserving the last actions which led to s in each of the paths. Hence, if action a led to an open state s via a path of cost g, and if the existing open list node (A, s) has the same g value, then the node is updated to $(A \cup \{a\}, s)$, thus all actions leading to s with path cost g are saved. Tie breaking also affects the criterion under which closed nodes are reopened. In A*, nodes are reopened only when reached via paths of lower g values. In PP-A*, if an action a leading to state s of some closed node (A, s) is not contained in A, and if the g values are equal, then the node reopens as $(\{A \cup \{a\}\}, s)$. However, when the node is expanded, only actions that are now allowed by ρ and were previously pruned, are executed. We now move to prove the correctness of PP-A*.

6.2 Proof of Correctness and Optimality

The next lemma refers to PP-A*, and assumes ρ to be an optimality preserving pruning method, which prunes according to the last action. We say that node (A, s) is **optimal on path** P, if A contains an action a which leads to state s on path P, and $g(s) = g^*(s)$. The notation $s \prec_P s'$ denotes the fact that state s precedes state s' in optimal path P.

Lemma 4. *In* PP-A*, *for any non-closed state s_k and for any optimal non-ρ-pruned path P from I to s_k, there exists an open list node (A', s') which is optimal on P.*

Proof. Let P be an optimal non-ρ-pruned path from I to s_k. If I is in the open list, let $s' = I$ and the lemma is trivially true since $g(I) = g^*(I) = 0$. Suppose I is closed. Let Δ be the set of all nodes (A_i, s_i) optimal on P, that were closed. Δ is not empty, since by assumption, I is in Δ. Let the nodes in Δ be ordered such that $s_i \prec_P s_j$ for $i < j$, and let j be the highest index of any s_i in Δ.

Since the closed node (A_j, s_j) has an optimal g value, it had been expanded prior to closing. From the properties of PP-A*, it follows that the expansion of (A_j, s_j), which is optimal on P, is followed with an attempt to generate a node (A_{j+1}, s_{j+1}) which is optimal on P as well. Generation of (A_{j+1}, s_{j+1}) must be allowed, since under the highest index assumption there can be no closed node containing s which is optimal on P. Naturally, $s_j \prec_P s_{j+1}$.

At this point, we note that actions in A_{j+1} cannot be removed by any competing path from I to s_{j+1}, since (A_{j+1}, s_{j+1}) has an optimal g value. It is possible, though, that additional actions leading to s_{j+1} are added to the node. The updated node can be represented by $(A'_{j+1} \supseteq A_{j+1}, s_{j+1})$, and the property of optimality on P holds. Additionally, node (A'_{j+1}, s_{j+1}) cannot be closed after its generation, since again, this contradicts the highest index property. Hence, there exists an open list node (A', s') which is optimal on P. This concludes the proof. □

Corollary 1. *If h is admissible,* PP-A* *is admissible.*

Proof. This follows directly from Lemma 4, the optimality preserving property of ρ and the properties of PP-A*, which allow every optimal, non-ρ-pruned path to be generated. □

7 EMPIRICAL EVALUATION

In order to evaluate our pruning methods, we performed experiments on 23 benchmark planning domains, used in the International Planning Competition [12]. Our pruning methods, as well as the changes to A*, were integrated into the implementation of A* in the Fast Downward (FD) planner [9]. In our implementation of PP-A*, only a single creating action is cached by state s, and if it is reached optimally via two paths which prune different sets of its applicable actions, no pruning is done at s. Caching all creating actions can be costly, and empirically, fully expanding such a state s had no negative effect on state-space reduction. All configurations used the state-of-the-art heuristic $h_{\textit{lm-cut}}$. Each test was given a 10 minute time limit including preprocessing, and was limited to 1GB of memory.

In configurations which include PB pruning, problem decomposition was handled in a preprocessing phase, using the METIS graph partitioning package. During this phase, the action graph was constructed and given as input to METIS, which computed 120 partitions under varying parameters. The partition with the highest symmetry score was chosen, and the problem was decomposed accordingly.

Table 1 depicts a per-domain coverage summary for all configurations, as well as ratios summarizing running time (with and without preprocessing), and number of nodes expanded and generated per domain. For example, the expansion ratio for configuration c and domain d is: $\sum_p expanded(baseline, d, p) / \sum_p expanded(c, d, p)$ for all p solved by both baseline and c[5]. Using PB pruning and PB+Tunneling (denoted PBT in the table) improved coverage in 3 domains, while hurting coverage in a single domain (*mprime*). Tunneling alone did not improve coverage and solved one problem less in the *mprime* domain as well.

Looking at the total time (running time including preprocessing) ratios, we first notice that tunneling incurs very little overhead, having almost no negative effect on running time, even when no pruning occurs (domains where generated ratio is 1). In *satellites, driverlog* and *psr*, tunneling performs very well, speeding up total time by 22%, 45%, and 52% respectively. PB pruning behaves differently than tunneling, exhibiting much larger speedups (e.g. 170%, 274% and 1274% speedup for *zenotravel*, *logistics98* and *satellites* respectively), but in some cases it incurs large preprocessing overhead (as high as 37% slowdown for *mprime*)[6]. Focusing on search time, we see that PB and PBT never cause slowdown, and can have an immense effect – exhibiting over 50% speedup in 7 domains, up to

[5] Restricting the ratio results only to problems solved does not misrepresent the results, since coverage scores are almost identical.

[6] This overhead can be reduced, for example, by stopping partitioning if after trying a few common parameter settings, we get a symmetry score of 0.

Table 1. Coverage and comparison of A*+$h_{lm\text{-}cut}$ using our 3 pruning rules. Time and node ratios are relative to the baseline planner.

	Coverage				Total Time			Search Time			Expanded			Generated		
Domain	A*	PB	T	PBT	PB	T	PBT	PB	T	PBT	PB	T	PBT	PB	T	PBT
airport	27	27	27	27	0.96	**1.01**	0.97	1.01	1.01	**1.03**	1	1	1	**1.03**	1	**1.03**
blocks	28	28	28	28	1	1	**1.01**	1	1	**1.01**	1	1	1	1	1	1
depot	7	7	7	7	1.03	1	**1.04**	1.04	1	**1.04**	1	1	1	**1.1**	1	**1.1**
driverlog	13	13	13	13	1.2	**1.45**	1.24	1.2	**1.45**	1.25	1.13	1.1	**1.24**	1.49	1.62	**1.68**
freecell	15	15	15	15	0.94	**1.02**	0.94	1.01	**1.02**	1.01	1	1	1	1	1	1
grid	2	2	2	2	0.97	**1.01**	0.97	1.01	**1.02**	1.01	1	1	1	1	1	1
gripper	6	6	6	6	1	1.01	**1.02**	1	1.01	**1.02**	1	1	1	**1.01**	1	**1.01**
logistics00	20	20	20	20	2.15	1	**2.16**	2.15	1	**2.16**	**1.52**	1	**1.52**	**2.88**	1	**2.88**
logistics98	6	6	6	6	**3.74**	1	3.71	**3.78**	1	3.75	**2.02**	1	**2.02**	**4.65**	1	**4.65**
miconic	141	141	141	141	0.82	**1.01**	0.83	0.99	1.02	**1.02**	1	1	1	1	1	1
mprime	**23**	20	22	20	0.74	**0.99**	0.75	1.03	0.99	**1.05**	**1.01**	1	**1.01**	**1.15**	1	**1.15**
mystery	15	15	15	15	0.63	0.98	0.64	1	0.98	**1.01**	1	1	1	**1.01**	1	**1.01**
openstacks	7	7	7	7	1	1	1	1	1	1	1	1	1	1	1	1
pathways-noneg	5	5	5	5	1.65	1.02	**1.7**	1.65	1.02	**1.7**	**1.31**	1	**1.31**	**1.87**	1	**1.87**
pipes-notankage	16	16	16	16	1.04	1.01	**1.05**	1.09	1.01	**1.1**	1	1	1	**1.09**	1	**1.09**
pipes-tankage	9	9	9	9	1	1	**1.01**	1.02	1	**1.04**	1	1	1	**1.04**	1	**1.04**
psr-small	49	49	49	49	1	**1.52**	0.99	1	**1.52**	1	1	**1.48**	1	1.08	**1.51**	1.08
rovers	7	**8**	7	**8**	2.61	1.04	**2.64**	2.63	1.04	**2.66**	**1.78**	1	**1.78**	3.23	1.02	**3.24**
satellite	7	**12**	7	**12**	13.74	1.22	**14.09**	14.55	1.22	**14.91**	**6.85**	1.04	**6.85**	19.18	1.35	**19.33**
tpp	6	6	6	6	1.03	1	**1.04**	**1.04**	1	**1.04**	**1.02**	1	**1.02**	**1.37**	1	**1.37**
transport	11	11	11	11	**1.5**	1	**1.5**	**1.54**	1	**1.54**	**1.24**	1	**1.24**	**1.79**	1	**1.79**
trucks	9	9	9	9	0.96	0.99	0.95	1	0.99	1	1	1	1	**1.01**	1	**1.01**
zenotravel	12	**13**	12	**13**	**2.7**	0.99	2.68	**2.81**	0.99	2.79	**1.54**	1	**1.54**	**2.86**	1	**2.86**
Total/Geometric Mean	441	**445**	440	**445**	1.339	1.047	**1.350**	1.424	1.048	**1.437**	1.226	1.023	**1.231**	1.558	1.055	**1.567**

1391% speedup on satellites. The two rightmost columns show that all three methods have a positive effect on the size of the explored search space as well.

It is no coincidence that PB pruning performs well on domains having natural "multi-agent" structure. These problems are highly decomposable, and the partitioned SAS+ problems include many private actions, after which pruning is performed. It is important to note the positive effect that adding internal tunneling has on PB pruning – PBT is the best overall performer in all five criteria.

8 RELATED WORK

Numerous works presented pruning techniques which preserve the optimality of search algorithms. *Transition reduction* techniques include *Sleep Sets* [7], *Commutativity Pruning* [8] and *Stratified Planning* [3], which perform action pruning, reducing the amount of paths to each reachable state, but *not* the set of reachable states. More closely related to our work are numerous *State Reduction* techniques, such as *Expansion Core* [4] and *Stubborn Action Core* [19], which are instances of the *stubborn set* method [17][7]. All stubborn-set based methods are *not* path dependent, i.e. perform the same pruning at state s regardless of s's creating action. Our methods, use additional information regarding the path to state s, and are not instances of stubborn sets. Aside from tunnel macros, the work by Coles and Coles [5] presents an interesting method for identifying irrelevant actions, which can be removed altogether from the problem description. This can be combined with our pruning methods seamlessly. In addition, symmetry detection methods [6, 5, 16] identify and prune symmetric states during the search process, and should complement our pruning approaches.

It remains an open question whether combinations of these pruning methods maintain optimality, and if not, how they can be combined effectively, while maintaining optimality. In general, more work must be done on providing sufficient conditions for which two optimality preserving pruning methods could be combined. Creating such a portfolio of pruning rules would strengthen existing planners, as the benefit of pruning is orthogonal to that of improved heuristics.

ACKNOWLEDGEMENTS

The authors were partly supported by ISF Grant 1101/07, the Paul Ivanier Center for Robotics Research and Production Management, and the Lynn and William Frankel Center for Computer Science. The authors would like to thank Malte Helmert and Martin Wehrle for their keen eyes and insight, and for the fruitful discussions.

[7] A recent paper [18] provides more information on stubborn set pruning methods in planning.

REFERENCES

[1] Christer Bäckström and Bernhard Nebel, 'Complexity results for SAS$^+$ planning', *Computational Intelligence*, **11**, 625–656, (1995).

[2] Ronen I. Brafman and Carmel Domshlak, 'From one to many: Planning for loosely coupled multi-agent systems', in *ICAPS*, pp. 28–35, (2008).

[3] Yixin Chen, You Xu, and Guohui Yao, 'Stratified planning', in *IJCAI*, pp. 1665–1670, (2009).

[4] Yixin Chen and Guohui Yao, 'Completeness and optimality preserving reduction for planning', in *IJCAI*, pp. 1659–1664, (2009).

[5] Amanda Jane Coles and Andrew Coles, 'Completeness-preserving pruning for optimal planning', in *ECAI*, pp. 965–966, (2010).

[6] Maria Fox and Derek Long, 'The detection and exploitation of symmetry in planning problems', in *IJCAI*, pp. 956–961, (1999).

[7] Patrice Godefroid, *Partial-Order Methods for the Verification of Concurrent Systems – An Approach to the State-Explosion Problem*, volume 1032 of *Lecture Notes in Computer Science*, Springer, 1996.

[8] Patrik Haslum and Hector Geffner, 'Admissible heuristics for optimal planning', in *AIPS*, pp. 140–149, (2000).

[9] Malte Helmert, 'The Fast Downward planning system', *J. Artif. Intell. Res. (JAIR)*, **26**, 191–246, (2006).

[10] Malte Helmert and Carmel Domshlak, 'Landmarks, critical paths and abstractions: What's the difference anyway?', in *ICAPS*, (2009).

[11] Jörg Hoffmann and Bernhard Nebel, 'The FF planning system: fast plan generation through heuristic search', *J. Artif. Int. Res.*, **14**(1), 253–302, (2001).

[12] ICAPS. The 2011 International Planning Competition. `http://www.plg.inf.uc3m.es/ipc2011-deterministic/`.

[13] Andreas Junghanns and Jonathan Schaeffer, 'Sokoban: Enhancing general single-agent search methods using domain knowledge', *Artif. Intell.*, **129**(1-2), 219–251, (2001).

[14] George Karypis and Vipin Kumar, 'A fast and high quality multilevel scheme for partitioning irregular graphs', *SIAM Journal on Scientific Computing*, **20**, 359–392, (1998).

[15] Richard E. Korf, 'Finding optimal solutions to Rubik's cube using pattern databases', in *AAAI*, pp. 700–705, (1997).

[16] Nir Pochter, Aviv Zohar, and Jeffrey S. Rosenschein, 'Exploiting problem symmetries in state-based planners', in *AAAI*, (2011).

[17] Antti Valmari, 'A stubborn attack on state explosion', in *CAV*, eds., Edmund M. Clarke and Robert P. Kurshan, volume 531 of *Lecture Notes in Computer Science*, pp. 156–165. Springer, (1990).

[18] Martin Wehrle and Malte Helmert, 'About partial order reduction in planning and computer aided verification', in *ICAPS*, (2012).

[19] You Xu, Yixin Chen, Qiang Lu, and Ruoyun Huang, 'Theory and algorithms for partial order based reduction in planning', *CoRR*, **abs/1106.5427**, (2011).