Partial Order Based Reduction in Planning: A Unifying Theory and New Algorithms

You Xi and Yixin Chen

Partial order based reduction (POR) has recently attracted research in planning. POR algorithms reduce search space by recognizing interchangeable orders between actions and expanding only a subset of all possible orders during the search. POR has been extensively studied in model checking and proved to be an enabling technique for reducing the search space and costs. Recently, two POR algorithms, including the expansion core (EC) and stratified planning (SP) algorithms, have been proposed. Being orthogonal to the development of accurate heuristic functions, these reduction methods show great potential to improve the planning efficiency from a new perspective. However, it... Read complete abstract on page 2.

Follow this and additional works at: https://openscholarship.wustl.edu/cse_research

Part of the Computer Engineering Commons, and the Computer Sciences Commons

Recommended Citation
Partial Order Based Reduction in Planning: A Unifying Theory and New Algorithms

You Xi and Yixin Chen

Complete Abstract:

Partial order based reduction (POR) has recently attracted research in planning. POR algorithms reduce search space by recognizing interchangable orders between actions and expanding only a subset of all possible orders during the search. POR has been extensively studied in model checking and proved to be an enabling technique for reducing the search space and costs. Recently, two POR algorithms, including the expansion core (EC) and stratified planning (SP) algorithms, have been proposed. Being orthogonal to the development of accurate heuristic functions, these reduction methods show great potential to improve the planning efficiency from a new perspective. However, it is unclear how these POR methods relate to each other and whether there exist stronger reduction methods. We propose a unifying theory for POR. The theory gives a necessary and sufficient condition for two actions to be semi-commutative, a condition that enables POR. We interpret both EC and SP in the theoretical framework. Further, based on the new theory, we propose new, stronger POR algorithms. Experimental results on various planning domains show significant search cost reduction.
Partial Order Based Reduction in Planning: A Unifying Theory and New Algorithms

Authors: You Xu and Yixin Chen

Corresponding Author: {yx2, chen}@cse.wustl.edu

Abstract: Partial order based reduction (POR) has recently attracted research in planning. POR algorithms reduce search space by recognizing interchangeable orders between actions and expanding only a subset of all possible orders during the search. POR has been extensively studied in model checking and proved to be an enabling technique for reducing the search space and costs.

Recently, two POR algorithms, including the expansion core (EC) and stratified planning (SP) algorithms, have been proposed. Being orthogonal to the development of accurate heuristic functions, these reduction methods show great potential to improve the planning efficiency from a new perspective. However, it is unclear how these POR methods relate to each other and whether there exist stronger reduction methods.

We propose a unifying theory for POR. The theory gives a necessary and sufficient condition for two actions to be semi-commutative, a condition that enables POR. We interpret both EC and SP in the theoretical framework. Further, based on the new theory, we propose new, stronger POR algorithms. Experimental results on various planning domains show significant search cost reduction.

Type of Report: Other
Partial Order Based Reduction in Planning: A Unifying Theory and New Algorithms

Abstract

Partial order based reduction (POR) has recently attracted research interest in planning. POR algorithms reduce search space by recognizing interchangeable orders between actions and expanding only a subset of all possible orders during the search. POR has been extensively studied in model checking and proved to be an enabling technique for reducing the search space and costs.

Recently, two POR algorithms, including the expansion core (EC) and stratified planning (SP) algorithms, have been proposed. Being orthogonal to the development of accurate heuristic functions, these reduction methods show great potential to improve the planning efficiency from a new perspective. However, it is unclear how these POR methods relate to each other and whether there exist stronger reduction methods.

We propose a unifying theory for POR. The theory gives a necessary and sufficient condition for two actions to be semi-commutative, a condition that enables POR. We interpret both EC and SP in the theoretical framework. Further, based on the new theory, we propose new, stronger POR algorithms. Experimental results on various planning domains show significant search cost reduction.

Introduction

Search is one of the most successful approaches to planning. As shown by recent work, search even with almost perfect heuristic guidance has some fundamental limitations (Helmert & Röger 2008). Heuristic planners still face scalability challenges for large-scale problems. Recently an orthogonal way to reduce the search cost, partial order based reduction (POR), has been studied for planning. Two POR algorithms for planning, the expansion core (EC) algorithm (Chen & Guo 2009) and stratified planning (SP) algorithm (Chen, Xu, & Guo 2009), have been proposed.

EC and SP both reduce the search space by expanding only a subset of applicable actions at each state. They are both completeness and optimality preserving: a complete and/or optimal search combined with any of them remains complete and/or optimal.

Theoretical properties for POR in planning are still not fully investigated. In this paper, we study the underlying mechanisms for EC and SP and interpret them in a unifying framework. We reveal that a semi-commutativeness relation between actions is a key source of reduction and we give a necessary and sufficient condition for determining semi-commutativeness. We characterize the completeness and optimality preserving property by the notion of representative subcategory, after abstracting paths and searches to categories. We interpret EC and SP in this framework. We then propose new POR reduction algorithms based on insights gained from the theoretical development. Our experiments show that the new POR algorithms have strong performance. Impressively, our new algorithm can give orders of magnitude of reduction on difficult domains for which EC and SP failed to provide any reduction.

Basic Definitions

Classical planning can be represented in STRIPS and SAS+ formalisms. Although EC and SP are developed on SAS+, we develop our theory using STRIPS. For space reasons, we give some useful notations, but do not give full definitions of STRIPS and SAS+.

A STRIPS planning task $\Sigma$ is a tuple $\Sigma = (F, O, I, G)$, where $F$ is a set of facts, $O$ is a set of actions, $I \subseteq F$ and $G \subseteq F$ are the sets of initial facts and goal facts, respectively. For each action $o \in O$, $P_o$, $A_o$, and $D_o$ denote the set of preconditions, add effects, and delete effects, respectively.

A SAS+ planning task $\Pi$ is defined as a tuple $\Pi = (X, O, S, s, s_g)$, $X = \{x_1, \ldots, x_N\}$ is a set of multi-valued state variables, each with an associated finite domain $\text{Dom}(x_i)$. $O$ is a set of actions and each action $o \in O$ is a tuple $(\text{pre}(o), \text{eff}(o))$, where both $\text{pre}(o)$ and $\text{eff}(o)$ define some partial assignments of variables in the form $x_i = v_i, v_i \in \text{Dom}(x_i)$. $s_g$ is a partial assignment that defines the goal. $S$ is the set of states. A state $s \in S$ is a full assignment to all the state variables.

For a given state $s$ and an action $o$, when all variable assignments in $\text{pre}(o)$ are met in state $s$, action $o$ is applicable at state $s$. After applying $o$ to $s$, the state variable assignment will be changed to a new state $s'$ according to $\text{eff}(o)$. We denote the resulting state of applying an applicable action $o$ to $s$ as $s' = \text{apply}(s, o)$.

There is correspondence between SAS+ and STRIPS formalisms of a planning task. For an action $o$, $P_o$, $A_o$, and $D_o$ correspond to the variables assignments in $\text{pre}(o), \text{eff}(o) \setminus \text{pre}(o)$, and $\text{pre}(o) \setminus \text{eff}(o)$, respectively.
A Unifying Theory

The theory we develop is for STRIPS tasks $\Sigma = (F, O, I, G)$. We first define some notations. The union of two sets $A$ and $B$ is written as $A + B$. The intersection of $A$ and $B$ is written as $AB$. A state $s$ is a subset of the fact set $F$, and we define $\overline{s} = F - s$ to be the complementation. In our deduction we use the following rules.

- $A(B + C) = AB + AC$ (distributive law)
- $\overline{AB} = \overline{A} + \overline{B}$ and $\overline{A + B} = \overline{A} \overline{B}$ (De Morgan’s laws)

Semi-Commutative Action and Path Pairs

The basic structure used in our theory is the concept of semi-commutative action pairs. Intuitively, if for any path, an action sequence $(a, b)$ can be replaced by $(b, a)$, then $a$ and $b$ are semi-commutative.

Definition 1 (Valid Path) For a STRIPS task $\Sigma$ and a state $s_0$, a sequence of actions $p = (\alpha_1, \ldots, \alpha_n)$ is a valid path if, let $s_i = \text{apply}(s_{i-1}, \alpha_i)$, $i = 1, \ldots, n$, $\alpha_i$ is applicable at $s_{i-1}$ for $i = 1, \ldots, n$. We also say that applying $p$ to $s$ results in the state $s_n$.

Definition 2 An ordered action pair $(a, b)$, $a, b \in O$ is a state-dependent semi-commutative action pair at state $s_0$ if when $(a, b)$ is a valid path at $s_0$, $(b, a)$ is also a valid path that results in the same state. We denote such a relationship by $s_0 : b \Rightarrow a$.

Definition 3 An ordered action pair $(a, b)$, $a, b \in O$ is a state-independent semi-commutative action pair (or semi-commutative action pair for short) if $(a, b)$ semi-commutative at any state $s \subseteq F$. We denote this relationship by $b \Rightarrow a$.

Note the following. 1) Semi-commutativeness is not a symmetric relationship, $b \Rightarrow a$ does not imply $a \Rightarrow b$. 2) The order in $b \Rightarrow a$ suggests that we should always try $(b, a)$ only during the search instead of trying both $(a, b)$ and $(b, a)$.

Theorem 1 An ordered action pair $(a, b)$, $a, b \in O$ is a state-dependent semi-commutative action pair at a state $s_0$ if and only if $P_aD_b = P_bD_a = P_b\overline{A}_a = A_bD_a = A_aD_b = \emptyset$.

Proof. First we prove the direction from left to right. Suppose $s_0 : a \Rightarrow b$, we have $P_b\overline{A}_a = \emptyset$ since $b$ is applicable at $s_0$. Hence, $P_b\overline{A}_a = \emptyset$. Since $a$ is applicable at $s_0$, and $(a, b)$ is a valid path, we have

\[
\emptyset = P_b - (s_0 - D_b + A_a) = P_b(s_0D_a + A_a) = P_b(s_0 - D_a + A_a) = P_b(s_0 + D_a)\overline{A}_a = P_b\overline{A}_a = A_bD_a = \emptyset,
\]

which implies $P_bD_a\overline{A}_a = \emptyset$. Note that $D_a\overline{A}_a = D_a$, thus we have $P_bD_a = \emptyset$. Similarly, since $(b, a)$ is also a valid path at $s_0$, we have $P_a - (s_0 - D_b + A_b) = \emptyset$, from which we can derive $P_aD_b = \emptyset$.

Finally, we consider the two states $s_1$ and $s_2$, resulted from applying $(a, b)$ and $(b, a)$ to $s_0$, respectively:

\[
s_1 = (s_0 - D_a + A_a) - D_b + A_b
\]

\[
s_2 = (s_0 - D_b + A_b) - D_a + A_a.
\]

Using $A - B = AB$, we get:

\[
s_1 = (s_0 + A_a + A_a)(\overline{D}_a + A_a + A_b)(A_b + D_b)
\]

\[
s_2 = (s_0 + A_a + A_a)(\overline{D}_b + A_a + A_b)(A_a + D_a)
\]

Let $T = (s_0 + A_a + A_b)$, we can simplify $s_1$ to:

\[
s_1 = T(\overline{D}_aA_b + \overline{D}_aD_b + A_aA_b + A_aD_b + A_bD_b)
\]

\[
= T\overline{D}_aD_b + TA_aD_b + TAb
\]

\[
= s_0\overline{D}_aD_b + A_aD_b + A_b
\]

Similarly, $s_2 = s_0D_b\overline{D}_a + A_bD_a + A_a$. We know that $s_1$ is identical to $s_2$ if and only if $s_1 - s_2 = \emptyset$ and $s_2 - s_1 = \emptyset$. We denote $s_0D_b\overline{D}_a$ by $K$ and have:

\[
s_1 - s_2 = (K + A_aD_b + A_a)(\overline{K} + A_bD_a + A_a)
\]

\[
= A_b\overline{K}D_a = (\overline{s_0} + D_a + B_a)A_bD_a
\]

\[
= \overline{s_0}A_bD_a + A_bD_a + D_bA_b + D_bA_b = A_bD_a
\]

Therefore, we can see that the necessary and sufficient condition for $s_1 - s_2 = \emptyset$ is that $A_bD_a = \emptyset$. Symmetrically, $s_2 - s_1 = \emptyset$ if and only if $A_bD_a = \emptyset$.

Now we prove the second part. Suppose we have $P_aD_b = P_bD_a = P_b\overline{A}_a = A_bD_a = A_aD_b = \emptyset$, assume $(a, b)$ is a valid path at $s_0$, we prove that $(b, a)$ is a valid path and leads to the same state.

Since $(a, b)$ is a valid path at $s_0$, we have $P_b\overline{A}_a = \emptyset$. Also, we have assumed that $P_b\overline{A}_a = \emptyset$. Hence we have $P_a - (s_0 - D_a + A_b) = P_a\overline{A}_b + P_aD_b = \emptyset + \emptyset = \emptyset$. Thus, $(b, a)$ is a valid path at $s_0$.

Last, since $s_1 - s_2 = A_bD_a = \emptyset$ and $s_2 - s_1 = A_bD_a = \emptyset$, $(a, b)$ and $(b, a)$ lead to the same state.

Corollary 1 An ordered action pair $(a, b)$, $a, b \in O$ is a semi-commutative action pair if and only if $P_aD_b = P_bD_a = P_b\overline{A}_a = A_bD_a = A_aD_b = \emptyset$.

Definition 4 For a path $p = (a_1, \ldots, a_n)$, another path $q = (b_1, \ldots, b_n)$ is 1-swap away from $p$ if there exists $i, 2 \leq i \leq n$ such that $b_i = a_{i-1}$, $b_{i-1} = a_i$, and $a_j = b_j$ for any $j, j \notin \{i - 1, i\}$, $1 \leq j \leq n$.

Definition 5 (1-Swap Semi-Commutative Path Pair) For a STRIPS planning task $\Sigma = (F, O, I, G)$, for two paths $p$ and $q$ that are 1-swap away, consider the action pair $(a, b)$ that is swapped, $p$ and $q$ are 1-swap semi-commutative if $b \Rightarrow a$.

Definition 6 (Semi-Commutative Path Pair) Two paths $p$ and $q$ are semi-commutative if $p = q$ or if there exists a sequence of paths $p_1 = p, p_2, \ldots, p_k = q$ such that $p_{j-1}$ is 1-swap semi-commutative with $p_j$ for $j = 2, \ldots, k$. We denote the relation as $q \Rightarrow p$. 

2
Intuitively, if \( q \Rightarrow p \), then \( q \) leads to the same state as \( p \) does and contains the same set of actions. Hence, a search can explore \( q \) only instead of both \( q \) and \( p \) without sacrificing completeness or optimality. Here, we assume the optimality metric is to minimize the total action cost, where each action has a positive cost.

**Categories of search and reduction**

In the following, we use category theory to describe partial order based reduction. See for example (Barr & Wells 1990) for introduction to category theory. Category is a relatively new alternative to set to the foundational notion of mathematics, a representation upon which logical constructions can be codified.

A category is a directed graph whose vertices (called objects) and arrows (called morphisms) satisfy certain additional requirement. In a category, each object \( A \) has an identity morphism \( 1_A : A \to A \), and each pair of morphisms \( f : A \to B \) and \( g : B \to C \) is assigned another morphism \( g \circ f : A \to C \) as the composition of morphisms. All the morphisms should satisfy the identity laws (for \( g : A \to B \), \( g \circ 1_A = g \), \( 1_B \circ g = g \)) and associative laws (\( f \circ (g \circ h) = (f \circ g) \circ h \)).

**Definition 7 (Category of Paths)** For a STRIPS planning task \( \Sigma = (F, O, I, G) \), the category of paths \( \text{pth}_\Sigma \) defines the following data:

- The objects include all finite-length paths whose elements are in \( O \);
- There is a morphism from a path \( p \) to a path \( q \) if and only if \( q \) leads to the same state as \( p \)

We see that \( \text{pth}_\Sigma \) is indeed a category since it satisfies the identity laws and associative laws.

A subcategory of a category \( C \) is a category \( S \) whose objects are objects in \( C \) and whose morphisms are morphisms in \( C \) with the same identities and composition of morphisms.

**Definition 8 (Representative Subcategory)** For a category \( C \), a subcategory \( S \) of \( C \) is representative if for each \( C \)-object \( B \) there exists a \( S \)-object \( A_B \) and a \( C \)-morphism \( r_B : B \to A_B \).

**Definition 9 (Category of Searches)** For a STRIPS planning task \( \Sigma = (F, O, I, G) \), the category of searches \( \text{sch}_\Sigma \) defines the following data:

- The objects include all subcategories of \( \text{pth}_\Sigma \);
- There is a morphism between \( A \to B \) if and only if \( B \) is a representative subcategory of \( A \).

We can verify that \( \text{sch}_\Sigma \) is a category since each object has an identity morphism and the morphisms satisfy the identity and associative laws.

**Definition 10 (Search)** For a STRIPS task \( \Sigma \), a search is an object in \( \text{sch}_\Sigma \). A search \( A \) is a goal search if every path in \( A \) leads to a goal state from \( I \). The optimal paths in a goal search form an optimal search.

The above abstract definition defines a search on a planning task as a set of paths with structures within the set, represented by the morphisms between paths.

**Definition 11 (Action-Preserving Reduction)** For a search \( A \) of a STRIPS planning task \( \Sigma = (F, O, I, G) \), an action preserving reduction is another search \( B \) such that there exists a morphism \( A \to B \) in \( \text{sch}_\Sigma \).

Intuitively, a search \( B \) is a category of paths. If there is a morphism \( B \to S \) in \( \text{sch}_\Sigma \), then \( S \) is a representative subcategory of \( B \). That is, for each object (path) \( p \) in \( B \), there exists a path \( q \) in \( S \), such that there is a \( B \)-morphism \( p \to q \). However, a \( B \)-morphism \( p \to q \) implies that \( p \) and \( q \) form a semi-commutative path pair, which means that \( q \) is a valid path if \( p \) is and they lead to the same state. Hence, a representative subcategory of a search \( B \) represents a completeness and optimality preserving reduction of the search.

**Interpretations of POR Algorithms**

In this section, we interpret two previous POR algorithms, stratified planning (SP) (Chen, Xu, & Guo 2009) and expansion core (EC) (Chen & Guo 2009), in the above theoretical framework.

**Stratified planning (SP)**

We summarize the key idea of the SP algorithm. A more formal treatment of SP can be found in (Chen, Xu, & Guo 2009). SP is based on the SAS+ formalism. For a SAS+ planning task, for an action \( o \in O \), define:

- the dependent variable set \( \text{dep}(o) \) is the set of state variables that appear in the assignments in \( \text{pre}(o) \);
- the transition variable set \( \text{trans}(o) \) is the set of state variables that appear in both \( \text{pre}(o) \) and \( \text{eff}(o) \);
- the affected variable set \( \text{aff}(o) \) is the set of state variables that appear in the assignments in \( \text{eff}(o) \).

**Definition 12** Given a SAS+ planning task \( \Pi \) with state variable set \( X \), its causal graph \( \text{CG}(\Pi) = (X, E) \) with \( X \) as the vertex set. There is an edge \( (x, x') \in E \) if and only if \( x \neq x' \) and there exists an action \( o \) such that \( x \in \text{trans}(o) \) and \( x' \in \text{dep}(o) \), or \( x \in \text{aff}(o) \) and \( x' \in \text{trans}(o) \).

SP uses a stratification of the CG. A stratification of \( \text{CG}(\Pi) = (X, E) \) is a partition of the set \( X = (X_1, \ldots, X_k) \) in such a way that there exists no edge \( e = (x, y) \) where \( x \in X_i, y \in X_j \) and \( i > j \).

By stratification, each state variable is assigned a level \( L(x) \), where \( L(x) = i \) if \( x \in X_i, 1 \leq i \leq k \). Subsequently, each action \( o \) is assigned a level \( L(o) \), \( 1 \leq L(o) \leq k \). \( L(o) \) is the level of the state variable(s) in \( \text{trans}(o) \). Note that all state variables in a same \( \text{trans}(o) \) must be in the same level.
Definition 13 (Follow-up Action) For a SAS+ task Π, an action $b$ is a follow-up action of $a$ (denoted as $a ∪ b$) if $aff(a) \cap dep(b) \neq \emptyset$ or $aff(b) \cap aff(a) \neq \emptyset$.

The SP algorithm can be combined with standard search algorithms, such as breadth-first search, depth-first search, and best first search (including $A^*$). During the search, for each state $s$ that is going to be expanded, the SP algorithm examines the action $a$ that leads to $s$. Then, for each applicable action $b$ at state $S$, SP makes the following decision:

- If $L(b) < L(a)$ and $b$ is not a follow-up action of $a$, then do not expand $b$ (we say that $b$ is not SP expandable after $a$). Otherwise, expand $b$.

Now we interpret SP in our framework. For a SAS+ task Π, consider its equivalent STRIPS task $Σ$. Each search algorithm corresponds to a set of paths it explores, which corresponds to an object $A$ in $sch_Σ$. Consider the set of paths that will be examined when the search is combined with SP. Let the SP-reduced path set correspond to an object $A_{SP}$ in $sch_Σ$.

Lemma 1 If an action $b$ is not SP-expandable after $a$, then $b \Rightarrow a$.

Proof. If $b$ is not SP-expandable after $a$, then $L(a) > L(b)$ and $b$ is not a follow-up action of $a$. Since $b$ is not a follow-up action of $a$, we know that $aff(a) \cap dep(b) = aff(a) \cap aff(b) = \emptyset$. Therefore, $P_b A_a = \emptyset$ and $D_b A_a = \emptyset$. Also, we see that $P_b A_o = \emptyset$ because $b$ is applicable immediately after $a$ is executed. Moreover, since $L(a) > L(b)$ implies that there is no edge from a state variable associated with $a$ to a state variable associated with $b$, from Definition 12, we can show that $P_o A_b = D_o A_b = \emptyset$. Thus, we proved that $P_b A_o = P_o A_b = A_b D_o = A_o D_b = \emptyset$. According to Corollary 1, we have $b \Rightarrow a$.

Theorem 2 For any search $A$ in $sch_Σ$, its SP-reduced search $A_{SP}$ is a representative subcategory of $A$.

Proof. We need to show that $A_{SP}$ is a representative subcategory of $A$. That is, for each path $p$ in $A$, we show that there is a path $p_{SP}$ in $A_{SP}$ such that there is a morphism $p \rightarrow p_{SP}$ in $pth_Σ$. We prove this by induction on $n$, the length of $p$.

The case is true when $n = 1$ since any action is a follow-up action of no-op. Now we assume for any path $p$ with length no more than $k$ in $p$, the proposition is true. We prove the case where $n = k + 1$.

For a path $p_0 = (a_1, \ldots, a_{k+1})$, consider the prefix $p_1 = (a_1, \ldots, a_k)$. By the induction hypothesis, there is a path $p_2 = (a_1, \ldots, a_k)$ such that $p_1 \rightarrow p_2$ is a morphism in $pth_Σ$.

Now we consider a new path $p^3 = (a_1, \ldots, a_k, a_{k+1})$. If $a_{k+1}$ is SP-expandable after $a_k$, then $p^3 \rightarrow p^2$ is a morphism in $pth_Σ$.

If $a_{k+1}$ is not SP-expandable after $a_k$, consider a new path $p^4 = (a_1, \ldots, a_{k-1}, a_{k+1}, a_k)$. From Lemma 1, we know that $a_{k+1} \Rightarrow a_k$, which implies that $p^2$ is a valid path leading to the same state as $p^4$.

Let $p_3 = (a_1, \ldots, a_{k-1}, a_{k+1})$, we know that there is a path $p_4 = (a_i, a_{i+1}, a_{k+1})$ such that $p_3 \rightarrow p_4$ is a morphism in $pth_Σ$. Define $p^3 = (a_1, \ldots, a_k, a_{k+1})$.

Comparing $p^2$ and $p^3$, we know that $L(a_{k+1}) > L(a_k)$, namely, the level of the last action in $p^2$ is strictly larger than that in $p^3$. We can repeat the above process to generate $p^4, p^5, \ldots$, as long as $p^0 \rightarrow p^i$ is not a morphism in $pth_Σ$.

Since we know that the level of the last action in $p^i$ is monotonically decreasing as $i$ increases, such a process must stop in a finite number of iterations and yield a path $p^f$ such that $p^0 \rightarrow p^f$ is a morphism in $pth_Σ$.

The above proof explains SP as a reduction that reduces each search in $sch_Σ$ to a representative subcategory. Hence, SP preserves completeness and optimality since any path explored by a search can be mapped to an equivalent path explored under SP-reduction.

Expansion core (EC) algorithm

We give a short outline of EC first. For detailed description of the EC algorithm, refer to (Chen & Guo 2009).

For a SAS+ task, each state variable $X_{s, i} = 1, \ldots, N$ is associated with a domain transition graph (DTG) $G_i$, a directed graph with vertex set $V(G_i) = Dom(x_i)$ and edge set $E(G_i)$. An edge $(v_i, v'_i)$ belongs to $E(G_i)$ if there is an action $o$ with $v_i \in pre(o)$ and $v'_i \in eff(o)$ in which case we say that $o$ is associated with the edge $e_i = (v_i, v'_i)$ (denoted as $o \leftarrow e_i$).

Definition 14 An action $o$ is associated with a DTG $G_i$ (denoted as $o \leftarrow G_i$) if $o$ is associated with any edge $e$ in $G_i$.

Definition 15 For a SAS+ task, for each DTG $G_i, i = 1, \ldots, N$, for a vertex $v \in V(G_i)$, an edge $e \in E(G_i)$ is a potential descendant edge of $v$ (denoted as $v \leftarrow e$) if 1) $G_i$ is goal-related and there exists a path from $v$ to the goal state in $G_i$ that contains $e$; or 2) $G_i$ is not goal-related and $e$ is reachable from $v$. A vertex $w \in V(G_i)$ is a potential descendant vertex of $v$ (denoted as $v \leftarrow w$) if 1) $G_i$ is goal-related and there exists a path from $v$ to the goal state in $G_i$ that contains $w$; or 2) $G_i$ is not goal-related and $w$ is reachable from $v$.

Definition 16 For a SAS+ task, given a state $s = (s_1, \ldots, s_N)$, for any $1 \leq i, j \leq N, i \neq j$, $s_i$ is a potential precondition of the DTG $G_i$ if there exist $o \in O$ and $e_j \in E(G_j)$ such that

$$s_j < e_j, o \leftarrow e_j, 	ext{ and } s_i \in pre(o)$$

(1)

Definition 17 Given a SAS+ state $s = (s_1, \ldots, s_N)$, for any $1 \leq i \neq j \leq N, s_i$ is a potential dependent of the DTG $G_j$ if there exist $o \in O, e_i = (s_i, s'_i) \in E(G_i)$ and $w_j \in V(G_j)$ such that

$$s_j < w_j, o \leftarrow e_j, w_i \in pre(o)$$
Definition 18 For a state s, the potential dependency graph PDG(s) is the directed graph with DTGs as vertices and there is an edge from $G_i$ to $G_j$ if and only if $s_i$ is a potential precondition or potential dependent of $G_j$.

Definition 19 For a directed graph $H$, a subset $C$ of $V(H)$ is a dependency closure if there do not exist $v, w \in V(H) - C$ such that $(v, w) \in E(H)$.

At a state s, EC method only expands actions in those DTGs within such a dependency closure of the PDG(s) that contains at least one DTG with an unachieved goal.

For any state s, an action a is goal-relevant if there exists a path from s to a goal state that contains a.

Lemma 2 For a state $s$ and a dependency closure $C$ of PDG(s), for any goal-relevant action a associated with a DTG in PDG(s) \ C, and any action b associated with a DTG in C that is applicable at s, we have $b \Rightarrow a$.

Proof. Since b is applicable at s, we know $P_b \subseteq s$. Since b is associated with a DTG within C, no fact in $P_b$ is a potential precondition of a and we have $P_bD_a = \emptyset$, which leads to $P_bD_a = \emptyset$ since $D_a \subseteq P_a$. On the other hand, since DTGs in C are not a potential dependent of those not in C, a precondition of b is not affected by a and we have $P_bD_a = \emptyset$ and $P_bA_a = \emptyset$. Finally, since a and b do not associate with the same DTG, we have $A_bD_a = A_aD_b = \emptyset$. All the five conditions in Corollary 1 are met.

To ensure action-preserving reduction, we give a list of conditions that is similar to the idea in stubborn set (Valmari 1989), a well-known technique for search space reduction in model checking.

Definition 20 (Stubborn Set) For a planning task, a set of actions $T(s)$ is a stubborn set at a state s if

A1 For any action $b \in T(s)$ and actions $b_1, \ldots, b_k \notin T(s)$, if $(b_1, \ldots, b_k, b)$ is a prefix of a path from s to a goal state, then $(b, b_1, \ldots, b_k)$ is a valid path from s and leads to the same state as $(b_1, \ldots, b_k, b)$ does.

A2 Any valid path from s to a goal state contains at least one action in $T(s)$.

A valid path $(a_1, \ldots, a_n)$ is stubborn-set conforming at a state $s_i$ if $a_i \in T(s_j)$ for $i = 1, \ldots, n$ where $s_{i+1} = \text{apply}(s_i, a_i)$. For any search $A$ in $\text{sch}_\Sigma$, the stubborn-set reduced search of $A$, $\text{ASS}$, is the subset of $A$ that includes all stubborn-set conforming paths.

Theorem 3 For any goal search $A$ in $\text{sch}_\Sigma$, there is a morphism $A \rightarrow \text{ASS}$ in $\text{sch}_\Sigma$.

Proof. We sketch the main idea. The proof is essentially the same as the proof to the stubborn set method in model checking (Valmari 1989), which is based on an induction on the length of paths. For any state s, for each path $p = (a_1, \ldots, a_n)$ from s to goal, according to A2 in Definition 20, we know that there must exist an action $a_i, 1 \leq i \leq n$ such that $a_i \in T(s)$. Then, according to A1, we can permute $p$ into a path $q = (a_1, a_3, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n)$ that also reaches the goal. Using induction, we can prove that any path $p$ can be permuted to a path $q$ such that $q \Rightarrow p$ and $\text{ASS}$ is a representative subcategory of $A$.

Lemma 3 The actions that the EC algorithm expands at any state $s$ form a stubborn set $T(s)$.

Proof. For a state $s$, let the dependency closure chosen by EC be $C \in \text{PDG(s)}$. For any action $b$ expanded by EC, and actions $b_1, \ldots, b_k$ that do not associate with a DTG in $C$, if $(b_1, \ldots, b_k, b)$ is a prefix of a path to goal, then we know $b \Rightarrow b_i$ for $i = 1, \ldots, k$ from Lemma 2. Therefore, $(b, b_1, \ldots, b_k)$ is also a valid path and A1 in Definition 20 is proved. Moreover, since $C$ includes at least one DTG $G$ that with an unachieved goal, some action in $G$ must be used in any path to goal. Since $G$ is in the closure, all actions in $G$ are expanded and A2 in Definition 20 is shown.

From Theorem 3 and Lemma 3, we can prove that EC is an action-preserving reduction.

Theorem 4 For any goal search $A$ in $\text{sch}_\Sigma$, its EC-reduced search $A_{EC}$ is a representative subcategory of $A$.

We see that EC is stronger than SP in the sense that it is an action preserving reduction for goal searches only, not any search. By restricting the preserving-ness to goal searches, EC does not guarantee that a non-goal path will be mapped to a EC-conforming path, which is a good thing that helps avoid exploring useless paths.

New POR Algorithms for Planning

From the last section, we see that, in essence, both SP and EC detect and exploit the semi-commutativity of actions. We have the following insights.

- Both SP and EC are based on the SAS* formalism, and utilize the notion of DTGs (state variables). However, the use of DTGs is not essential for POR reduction. They are used only to ensure certain conditions in Theorem 1. Further, both SP and EC give sufficient but not necessary conditions for finding semi-commutative action pairs. For example, for domains such as pipesworld where the CG has only one strongly-connected component, SP and EC can give no reduction, although semi-commutative action pairs still exist in these domains.

- Both SP and EC have certain advantages. For a state $s$ and applicable actions $a_1, \ldots, a_n$, SP will expand each of $a_i, i = 1, \ldots, n$ and for each $a_i$ may prune some actions $b_j$ if $b_j \Rightarrow a_i$. EC has the advantage of not having to expand all $a_i$. Instead, it divides actions into those in $T(s)$ and those not in $T(s)$. As a cost, it
needs to ensure that any action in $T(s)$ must be semi-commutative with any action not in $T(s)$ (i.e., a closure), which may miss certain semi-commutativeness and chance of reduction.

Based on the above two observations, we propose our new algorithm, action closure (AC) reduction. Unlike SP and EC, the AC algorithm does not analyze the CG and use DTG as the basic unit of decision (whether to be expanded or not). Instead, it treats each action as the basic unit of decision.

**Definition 21 (Action Dependency Graph)** For a STRIPS planning task, its action dependency graph (ADG) is defined as a directed graph in which each vertex is an action, and there is an edge from action $a$ to $b$ if and only if $P_a D_b \neq \emptyset$ or $P_b D_a \neq \emptyset$ or $P_b A_a \neq \emptyset$ or $A_b D_a \neq \emptyset$ or $A_b D_b \neq \emptyset$.

**Definition 22 (Contracted ADG)** Given an ADG, its contracted ADG (CADG) is a graph where each vertex is a maximum strongly connected component (SCC) of the ADG and there is an edge between two SCCs if there is an edge in the ADG from a vertex in one SCC to a vertex in another SCC.

A topological sort on the CADG generates an ordered sequence of its vertices: $(\text{SCC}_1, \ldots, \text{SCC}_N)$, where $\text{SCC}_1$ is the SCC with zero in-degree in the CADG. The topological sort is not unique and we currently choose one randomly. Given a topological sort of the CADG, each action $a$ is assigned a layer $l(a)$, which is the index of the SCC the action belongs to, i.e., $a \in \text{SCC}_{l(a)}$.

**Definition 23** An action $b$ is supported by an action $a$ if and only if $P_b A_a \neq \emptyset$.

The AC algorithm works as follows. For each state $s$, let the action that leads to $s$ be $a$,

B1 If $A_a$ includes a goal fact, it expands all applicable actions;

B2 Otherwise, it finds the minimum index $M$, $M \leq N$, such that $\text{SCC}_1 \cup \cdots \cup \text{SCC}_M$ include all the applicable actions that are supported by $a$.

**Lemma 4** For any two actions $a$ and $b$ such that $l(b) < l(a)$, we have $b \Rightarrow a$.

**Proof.** If $l(b) < l(a)$, there is no edge from $a$ to $b$ in the ADG. Thus, $P_a D_b = P_b D_a = P_b A_a = A_b D_a = A_a D_b = \emptyset$. The conditions in Corollary 1 are met. □

A path is AC-conforming if it can be possibly generated by a search with the AC algorithm. For any search $A$, the AC-reduced search is the subcategory of $A$ that includes all AC-conforming paths as objects.

**Definition 24 (Optimality Stubborn Set)** For a planning task, a set of actions $T(s)$ is an optimality stubborn set at a state $s$ if

$O1$ For any action $b \in T(s)$, and actions $b_1, \ldots, b_k \notin T(s)$, if $(b_1, \ldots, b_k, b)$ is a prefix of a path to goal, then $(b, b_1, \ldots, b_k)$ is a valid path from $s$ that leads to the same state as $(b_1, \ldots, b_k, b)$.

$O2$ Any optimal path from $s$ to a goal contains at least one action in $T(s)$.

**Theorem 5** For any optimal search $A$ in $\text{sch}_{\Sigma}$, there is a morphism $A \rightarrow A_{\text{OSS}}$ in $\text{sch}_{\Sigma}$, where $A_{\text{OSS}}$ is the subcategory of $A$ that conforms to optimality stubborn sets.

Theorem 5 can be proved using a proof parallel to that of Theorem 3.

**Lemma 5** The actions that the AC algorithm expands at any state $s$ form an optimality stubborn set $T(s)$.

**Proof.** For a state $s$, assume AC expands applicable actions in $T = \{\text{SCC}_1 \cup \cdots \cup \text{SCC}_M\}$. Consider any actions $b_1, \ldots, b_k$ that are not in $T$. If $(b_1, \ldots, b_k, b)$ is the prefix of an optimal path, then we know $b \Rightarrow b_i$ for $i = 1, \ldots, k$ from Lemma 4. Therefore, $(b, b_1, \ldots, b_k)$ is also a valid path and $O1$ is proved. Moreover, if no action is used in $T$, since $T$ includes all the actions supported by $a$, we can delete $a$ to obtain a better plan (unless $a$ adds a goal which is covered by condition B1 in the AC algorithm). Hence, any optimal path must include at least one action in $T$ and $O2$ is satisfied. □

From Theorem 5 and Lemma 5, we have shown that AC is an action preserving reduction.

**Theorem 6** For any optimal search $A$ in $\text{sch}_{\Sigma}$, its AC-reduced search $A_{\text{AC}}$ is a representative subcategory of $A$.

The AC algorithm is a stubborn set method. Last, we propose an enhanced version of the AC algorithm that adds the idea of stratified planning. For each state $s$ with a leading action $a$, the AC$^+$ algorithm applies the same conditions B1 and B2 as used by AC, while imposing one more restriction.

B3 For each applicable action $b$, if $b \Rightarrow a$, then it does not expand $b$.

This condition B3 can be viewed as a SP style reduction, added to the stubborn set reduction in the AC algorithm. The correctness of AC$^+$ is obvious as it simply checks the semi-commutativeness of two adjacent actions. For any path $p$ there is a path $q$ that conforms to B3. Hence, any optimal search $A$ can be reduced to a search $A_{\text{AC}}$ and then to $A_{\text{AC}^+}$, all action-preserving.

**Theorem 7** For any optimal search $A$ in $\text{sch}_{\Sigma}$, its AC$^+$-reduced search $A_{\text{AC}^+}$ is a representative subcategory of $A$.

**Experimental Results**

We test on STRIPS problems in the recent International Planning Competitions (IPCs): IPC3, IPC4, and IPC5.
We implemented our algorithm in the Fast Downward (FD) planner (Helmert 2006). We only modified the state expansion part.

Table 1 shows the results of FD, SP, AC, and AC+ on the testing domains except for pipesworld and freecell domains, whose results are shown in Table 2. All algorithms give the same solution quality. We see that the performance of the original SP is consistently better than the original FD. AC+ can significantly improve SP in driverlog and tpp domains in terms of the numbers of generated and expanded nodes. AC is generally better than FD in terms of both generated and expanded nodes.

Comparing AC against SP, we see that typically AC generates more states but expands less, since AC is a stubborn set style reduction which tends to expand less nodes. Due to a deferred heuristic evaluation scheme in FD, the number of heuristic evaluations is determined by the number of expanded nodes. As a result, the CPU time of AC often is less than that of SP, even if AC generates more nodes. The trucks, storage, and tpp domains best illustrate this point. AC+ has a similar comparison against SP and is faster than SP in most instances except for the trucks domain.

Comparing AC+ against AC, we see that AC+ is better in driverslog, depot, storage, and tpp. AC+ is orders of magnitude better than AC for some instances from driverslog and depot. AC+ is faster than the original FD in most instances except for the trucks domain.

Now we turn to the surprising results on the pipesworld and freecell domains in Table 2. These two domains are deemed very difficult since their CG is densely connected and cannot be decomposed into multiple strongly connected components. Therefore, SP, EC and AC all fail to give any reduction. Surprisingly, AC+ can give significant reduction. In Table 1, we compare FD and AC+. We did not report SP and AC since they cannot give any reduction and their state expansions are the same as FD. We see that AC+ can reduce the number of expanded and generated nodes by orders of magnitude for many instances such as freecell-15 and pipesworld-12. It is encouraging that POR algorithms can work not only for those largely decomposable domains but also those domains whose state variables are highly inter-dependent.

In conclusion, we have proposed a theory to unify various POR algorithms and explained their completeness and optimality preserving properties. Based on the new theory, we have proposed two new reduction algorithms and evaluated their performance. There are still many open problems in this direction. For example, the use of categorical notions can be further studied. The category theory provides a foundation for describing abstract algebraic structures. For example, an important goal of POR reduction is to find the minimum action preserving set of paths in a search, which can be represented by the notion of the terminal object in schΣ.

<table>
<thead>
<tr>
<th>Domains</th>
<th>Fast Downward ExpandedGenerated Time</th>
<th>AC+ ExpandedGenerated Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>free1</td>
<td>32 190 0.07</td>
<td>30 322 0.16</td>
</tr>
<tr>
<td>free2</td>
<td>42 262 0.06</td>
<td>56 108 0.58</td>
</tr>
<tr>
<td>free3</td>
<td>53 397 0.28</td>
<td>72 693 0.36</td>
</tr>
<tr>
<td>free4</td>
<td>116 533 0.24</td>
<td>108 542 0.16</td>
</tr>
<tr>
<td>free5</td>
<td>796 4191 1.67</td>
<td>1808 14858 3.44</td>
</tr>
<tr>
<td>free6</td>
<td>390 2825 1.68</td>
<td>475 4608 3.1</td>
</tr>
<tr>
<td>free7</td>
<td>535 3281 2.42</td>
<td>534 4246 3.34</td>
</tr>
<tr>
<td>free8</td>
<td>2379 10110 10.72</td>
<td>532 6818 2.79</td>
</tr>
<tr>
<td>free9</td>
<td>5754 53638 3.44</td>
<td>12849 125532 111.9</td>
</tr>
<tr>
<td>free10</td>
<td>2406 9001 16.7</td>
<td>721 6961 11.83</td>
</tr>
<tr>
<td>free11</td>
<td>1362 8013 9.52</td>
<td>634 6396 4.21</td>
</tr>
<tr>
<td>free12</td>
<td>12083 77311 138.41</td>
<td>11757 146286 171.75</td>
</tr>
<tr>
<td>free13</td>
<td>390 2825 1.68</td>
<td>605 7558 6.13</td>
</tr>
<tr>
<td>free14</td>
<td>4431 40529 46.83</td>
<td>2841 32298 48.39</td>
</tr>
<tr>
<td>free15</td>
<td>5329 307397 463.72</td>
<td>-</td>
</tr>
<tr>
<td>free16</td>
<td>657 4870 12.3</td>
<td>-</td>
</tr>
<tr>
<td>free17</td>
<td>-</td>
<td>330 3104 7.06</td>
</tr>
</tbody>
</table>

Table 2: Comparison of FD and AC+ on freecell (free) and pipesworld (pipe) domains. We show numbers of expanded and generated nodes. "-" means timeout after 1800s.

References
<table>
<thead>
<tr>
<th>Domains</th>
<th>Fast Downward</th>
<th>Stratified Planning</th>
<th>AC</th>
<th>AC*</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Expanded</td>
<td>Generated Time</td>
<td></td>
<td></td>
</tr>
<tr>
<td>driverlog1</td>
<td>280</td>
<td>2858</td>
<td>0.07</td>
<td></td>
</tr>
<tr>
<td>driverlog2</td>
<td>1810</td>
<td>21582</td>
<td>0.11</td>
<td></td>
</tr>
<tr>
<td>driverlog3</td>
<td>599</td>
<td>7155</td>
<td>0.18</td>
<td></td>
</tr>
<tr>
<td>driverlog4</td>
<td>527</td>
<td>6173</td>
<td>0.18</td>
<td></td>
</tr>
<tr>
<td>driverlog5</td>
<td>1288</td>
<td>18823</td>
<td>0.45</td>
<td></td>
</tr>
<tr>
<td>driverlog16</td>
<td>493226</td>
<td>8831575</td>
<td>105.7</td>
<td></td>
</tr>
<tr>
<td>driverlog17</td>
<td>9211</td>
<td>303992</td>
<td>5.44</td>
<td></td>
</tr>
<tr>
<td>driverlog18</td>
<td>13524</td>
<td>353873</td>
<td>17.2</td>
<td></td>
</tr>
<tr>
<td>truck6</td>
<td>339</td>
<td>5071</td>
<td>0.07</td>
<td></td>
</tr>
<tr>
<td>truck7</td>
<td>38532</td>
<td>165934</td>
<td>2.37</td>
<td></td>
</tr>
<tr>
<td>truck8</td>
<td>1966</td>
<td>11558</td>
<td>0.37</td>
<td></td>
</tr>
<tr>
<td>truck9</td>
<td>236058</td>
<td>2023160</td>
<td>17.29</td>
<td></td>
</tr>
<tr>
<td>truck10</td>
<td>325002</td>
<td>3064955</td>
<td>29.82</td>
<td></td>
</tr>
<tr>
<td>truck11</td>
<td>99902</td>
<td>1542311</td>
<td>10.45</td>
<td></td>
</tr>
<tr>
<td>depot1</td>
<td>23</td>
<td>91</td>
<td>0.01</td>
<td></td>
</tr>
<tr>
<td>depot2</td>
<td>65</td>
<td>485</td>
<td>0.02</td>
<td></td>
</tr>
<tr>
<td>depot3</td>
<td>6121</td>
<td>48336</td>
<td>1.19</td>
<td></td>
</tr>
<tr>
<td>depot4</td>
<td>9291</td>
<td>78046</td>
<td>2.64</td>
<td></td>
</tr>
<tr>
<td>depot5</td>
<td>343364</td>
<td>2884118</td>
<td>103.36</td>
<td></td>
</tr>
<tr>
<td>depot6</td>
<td>28204</td>
<td>261112</td>
<td>8.83</td>
<td></td>
</tr>
<tr>
<td>depot7</td>
<td>162784</td>
<td>1674483</td>
<td>55.54</td>
<td></td>
</tr>
<tr>
<td>depot8</td>
<td>15142311</td>
<td>766989</td>
<td>16.85</td>
<td></td>
</tr>
<tr>
<td>storage1</td>
<td>4</td>
<td>7</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>storage2</td>
<td>4</td>
<td>9</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>storage3</td>
<td>4</td>
<td>11</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>storage4</td>
<td>32</td>
<td>85</td>
<td>0.02</td>
<td></td>
</tr>
<tr>
<td>storage5</td>
<td>20</td>
<td>94</td>
<td>0.01</td>
<td></td>
</tr>
<tr>
<td>storage6</td>
<td>31</td>
<td>176</td>
<td>0.04</td>
<td></td>
</tr>
<tr>
<td>storage7</td>
<td>235</td>
<td>634</td>
<td>0.02</td>
<td></td>
</tr>
<tr>
<td>storage8</td>
<td>95</td>
<td>480</td>
<td>0.07</td>
<td></td>
</tr>
<tr>
<td>storage9</td>
<td>93</td>
<td>744</td>
<td>0.04</td>
<td></td>
</tr>
<tr>
<td>storage10</td>
<td>1521</td>
<td>4364</td>
<td>0.23</td>
<td></td>
</tr>
<tr>
<td>storage11</td>
<td>297</td>
<td>1774</td>
<td>0.18</td>
<td></td>
</tr>
<tr>
<td>storage12</td>
<td>1496</td>
<td>11550</td>
<td>0.77</td>
<td></td>
</tr>
<tr>
<td>storage13</td>
<td>4930</td>
<td>17046</td>
<td>1.31</td>
<td></td>
</tr>
<tr>
<td>storage14</td>
<td>2668</td>
<td>18730</td>
<td>1.11</td>
<td></td>
</tr>
<tr>
<td>storage15</td>
<td>325</td>
<td>2673</td>
<td>0.36</td>
<td></td>
</tr>
<tr>
<td>storage16</td>
<td>276</td>
<td>3385</td>
<td>0.64</td>
<td></td>
</tr>
<tr>
<td>tpp1</td>
<td>8</td>
<td>6</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>tpp2</td>
<td>9</td>
<td>17</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>tpp3</td>
<td>12</td>
<td>29</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>tpp4</td>
<td>15</td>
<td>44</td>
<td>0.06</td>
<td></td>
</tr>
<tr>
<td>tpp5</td>
<td>22</td>
<td>92</td>
<td>0.06</td>
<td></td>
</tr>
<tr>
<td>tpp6</td>
<td>664</td>
<td>3641</td>
<td>0.01</td>
<td></td>
</tr>
<tr>
<td>tpp7</td>
<td>1591</td>
<td>9403</td>
<td>0.17</td>
<td></td>
</tr>
<tr>
<td>tpp8</td>
<td>4685</td>
<td>37683</td>
<td>0.29</td>
<td></td>
</tr>
<tr>
<td>tpp9</td>
<td>3630</td>
<td>25924</td>
<td>0.59</td>
<td></td>
</tr>
<tr>
<td>tpp10</td>
<td>12242</td>
<td>110251</td>
<td>1.66</td>
<td></td>
</tr>
<tr>
<td>tpp11</td>
<td>13148</td>
<td>126912</td>
<td>2.08</td>
<td></td>
</tr>
<tr>
<td>tpp12</td>
<td>36690</td>
<td>364082</td>
<td>4.48</td>
<td></td>
</tr>
<tr>
<td>tpp13</td>
<td>24066</td>
<td>295068</td>
<td>4.35</td>
<td></td>
</tr>
<tr>
<td>tpp14</td>
<td>68494</td>
<td>894865</td>
<td>14.74</td>
<td></td>
</tr>
<tr>
<td>tpp15</td>
<td>37145</td>
<td>477808</td>
<td>8.71</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Comparison of several algorithms. We give number of generated nodes, number of expanded nodes, and CPU time in seconds. "-" means timeout after 300 seconds.