Report Number: wucse-2009-28

2009

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 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... **Read complete abstract on page 2.**

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.

2009-28

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 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.

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 in planning. POR algorithms reduce search space by recognizing inter-changable 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 Σ is a tuple $\Sigma = (F, O, I, G)$, where F is a set of facts, O is a set of actions, $I \subset F$ and $G \subset F$ 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 Π is defined as a tuple $\Pi = (X, O, S, s_{\mathcal{I}}, s_{\mathcal{G}})$. $X = \{x_1, \cdots, x_N\}$ is a set of multi-valued **state variables**, each with an associated finite domain $Dom(x_i)$. O is a set of actions and each action $o \in O$ is a tuple $(pre(o), eff(o))$, where both $pre(o)$ and $eff(o)$ define some partial assignments of variables in the form $x_i = v_i, v_i \in Dom(x_i)$. $s_{\mathcal{G}}$ 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 $pre(o)$ 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 *eff*(o). We denote the resulting state of applying an applicable action o to s as $s' = apply(s, o)$.

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 $pre(o)$, *eff*$(o) \setminus pre(o)$, and $pre(o) \setminus$ *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 Σ and a state s_0, a sequence of actions $p = (o_1, \ldots, o_n)$ is a valid path if, let $s_i = apply(s_{i-1}, o_i), i = 1, \ldots, n$, o_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 $P_aD_b = P_bD_a = P_b\overline{s_0}A_a = A_bD_a = A_aD_b = \emptyset$.*

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

$$\begin{aligned} \emptyset &= P_b - (s_0 - D_a + A_a) = P_b\overline{(s_0\overline{D_a} + A_a)} \\ &= P_b(\overline{s_0} + D_a)\overline{A_a} = P_b\overline{s_0}\ \overline{A_a} + P_bD_a\overline{A_a}, \end{aligned}$$

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:

$$\begin{aligned} s_1 &= (s_0 - D_a + A_a) - D_b + A_b \\ s_2 &= (s_0 - D_b + A_b) - D_a + A_a. \end{aligned}$$

Using $A - B = A\overline{B}$, we get:

$$\begin{aligned} s_1 &= (s_0 + A_a + A_b)(\overline{D_a} + A_a + A_b)(A_b + \overline{D_b}) \\ s_2 &= (s_0 + A_a + A_b)(\overline{D_b} + A_a + A_b)(A_a + \overline{D_a}) \end{aligned}$$

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

$$\begin{aligned} s_1 &= T(\overline{D_a}A_b + \overline{D_a}\ \overline{D_b} + A_aA_b + A_a\overline{D_b} + A_b + A_b\overline{D_b}) \\ &= T\overline{D_a}\ \overline{D_b} + TA_a\overline{D_b} + TA_b \\ &= s_0\overline{D_a}\ \overline{D_b} + A_a\overline{D_b} + A_b \end{aligned}$$

Similarly, $s_2 = s_0\overline{D_b}\ \overline{D_a} + A_b\overline{D_a} + A_a$. 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 $s_0\overline{D_b}\ \overline{D_a}$ by K and have:

$$\begin{aligned} s_1 - s_2 &= (K + A_a\overline{D_b} + A_b)(\overline{K + A_b\overline{D_a} + A_a}) \\ &= A_b\overline{K}D_a = (\overline{s}_0 + D_a + D_b)A_bD_a \\ &= \overline{s}_0A_bD_a + A_bD_a + D_bA_bD_a = A_bD_a \end{aligned}$$

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 $A_aD_b = \emptyset$.

Now we prove the second part. Suppose we have $P_aD_b = P_bD_a = P_b\overline{s_0}A_a = A_bD_a = A_aD_b = \emptyset$, 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 $P_b\overline{s_0}\ \overline{A_a} = \emptyset$. Also, we have assumed that $P_b\overline{s_0}A_a = \emptyset$. Hence we have $P_b\overline{s_0} = P_b\overline{s_0}\overline{A_a} + P_b\overline{s_0}A_a = \emptyset$ and b is applicable at s_0. Further, we know that a is applicable after the execution of b since we have $P_a - (s_0 - D_b + A_b) = P_a\overline{s_0}\ \overline{A_b} + P_aD_b = \emptyset + \emptyset = \emptyset$. Thus, (b, a) is a valid path at s_0.

Last, since $s_1 - s_2 = A_bD_a = \emptyset$ and $s_2 - s_1 = A_aD_b = \emptyset$, (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 $P_aD_b = P_bD_a = P_bA_a = A_bD_a = A_aD_b = \emptyset$.*

Definition 4 *For a path $p = (a_1, \cdots, a_n)$, another path $q = (b_1, \cdots, b_n)$ 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 $p_1 = p, p_2, \cdots, p_k = q$ such that p_{j-1} is 1-swap semi-commutative with p_j for $j = 2, \cdots, k$. We denote the relation as $q \Rightarrow p$.*

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 as 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 $1_A : A \rightarrow A$, and each pair of morphisms $f : A \rightarrow B$ and $g : B \rightarrow C$ is assigned another morphism $g \circ f : A \rightarrow C$ as the composition of morphisms. All the morphisms should satisfy the identity laws (for $g : A \rightarrow B$, $g \circ 1_A = g$, $1_B \circ g = g$) 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 $\mathbf{pth}_\Sigma$ 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 $q \Rightarrow p$.

We see that $\mathbf{pth}_\Sigma$ 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 $r_B : B \rightarrow A_B$.*

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

- The objects include all subcategories of $\mathbf{pth}_\Sigma$;
- There is a morphism between $A \rightarrow B$ if and only if B is a representative subcategory of A.

We can verify that $\mathbf{sch}_\Sigma$ 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 Σ, a search is an object in $\mathbf{sch}_\Sigma$. 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 $A \rightarrow B$ in $\mathbf{sch}_\Sigma$.

Intuitively, a search B is a category of paths. If there is a morphism $B \rightarrow S$ in $\mathbf{sch}_\Sigma$, 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 $p \rightarrow q$. However, a B-morphism $p \rightarrow q$ 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** $dep(o)$ is the set of state variables that appear in the assignments in $pre(o)$.
- the **transition variable set** $trans(o)$ is the set of state variables that appear in both $pre(o)$ and *eff*(o).
- the **affected variable set** *aff*(o) is the set of state variables that appear in the assignments in *eff*(o).

Definition 12 *Given a SAS+ planning task Π with state variable set X, its **causal graph** (CG) is a directed graph $CG(\Pi) = (X, E)$ 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 $x \in trans(o)$ and $x' \in dep(o)$, or, $x \in$ aff(o) and $x' \in trans(o)$.*

SP uses a **stratification** of the CG. A stratification of $CG(\Pi) = (X, E)$ is a partition of the set X: $X = (X_1, \cdots, X_k)$ 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 $trans(o)$. Note that all state variables in a same $trans(o)$ 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 $a \rhd b$) if aff$(a) \cap dep(b) \neq \emptyset$ or aff$(a) \cap$ aff$(b) \neq \emptyset$.*

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 $\mathbf{sch}_\Sigma$. 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 $\mathbf{sch}_\Sigma$.

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 $P_b D_a = \emptyset$ 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 $P_a D_b = D_a A_b = \emptyset$. Thus, we proved that $P_a D_b = P_b D_a = P_b A_a = A_b D_a = A_a D_b = \emptyset$. According to Corollary 1, we have $b \Rightarrow a$. ■

Theorem 2 *For any search A in $\mathbf{sch}_\Sigma$, 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 $\mathbf{pth}_\Sigma$. 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 $p_2 = (a_1^1, \ldots, a_k^1)$ such that $p_1 \rightarrow p_2$ is a morphism in $\mathbf{pth}_\Sigma$.

Now we consider a new path $p^1 = (a_1^1, \ldots, a_k^1, a_{k+1})$. If a_{k+1} is SP-expandable after a_k^1, then $p^0 \rightarrow p^1$ is a morphism in $\mathbf{pth}_\Sigma$.

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

Let $p_3 = (a_1^1, \ldots, a_{k-1}^1, a_{k+1})$, we know that there is a path $p_4 = (a_1^2, \ldots, a_k^2)$ such that $p_3 \rightarrow p_4$ is a morphism in $\mathbf{pth}_\Sigma$. Define $p^3 = (a_1^2, \ldots, a_k^2, a_k^1)$.

Comparing p^2 and p^3, we know that $L(a_{k+1}) > L(a_k^1)$, 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, $p^5, \cdots$, as long as $p^0 \rightarrow p^j$ is not a morphism in $\mathbf{pth}_\Sigma$.

Since we know that the level of the last action in p^j is monotonically decreasing as j increases, such a process must stop in a finite number of iterations and yield a path p^j such that $p^0 \rightarrow p^j$ is a morphism in $\mathbf{pth}_\Sigma$. ■

The above proof explains SP as a reduction that reduces each search in $\mathbf{sch}_\Sigma$ 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 $X_i, i = 1, \cdots, N$ 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 $o \vdash e_i$).

Definition 14 *An action o is **associated** with a DTG G_i (denoted as $o \vdash G_i$) if o is associated with any edge 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 $v \lhd e$) 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 $v \lhd w$) 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 $s = (s_1, \cdots, s_N)$, for any $1 \leq i, j \leq N, i \neq j$, s_i is a **potential precondition** of the DTG G_j if there exist $o \in O$ and $e_j \in E(G_j)$ such that*

$$s_j \lhd e_j,\ o \vdash e_j,\ \text{and } s_i \in pre(o) \tag{1}$$

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*

$$s_j \lhd w_j, o \vdash e_j, w_i \in pre(o)$$

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 $v \in C$ and $w \in V(H) - C$ 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 $PDG(s)\backslash C$, 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 $P_b P_a = \emptyset$, which leads to $P_b D_a = \emptyset$ 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 $P_b D_a = \emptyset$ and $P_b A_a = \emptyset$. Finally, since a and b do not associate with a same DTG, we have $A_b D_a = A_a D_b = \emptyset$. 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 $b_1, \cdots, b_k \notin T(s)$, if $(b_1, \cdots, b_k, b)$ is a prefix of a path from s to a goal state, then $(b, b_1, \cdots, b_k)$ is a valid path from s and leads to the same state as $(b_1, \cdots, b_k, b)$ does.

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

A valid path $(a_1, \cdots, a_n)$ is **stubborn-set conforming** at a state s_1 if $a_i \in T(s_i)$ for $i = 1, \cdots n$ where $s_{i+1} = apply(s_i, a_i)$. For any search A in $\mathbf{sch}_\Sigma$, the stubborn-set reduced search of A, A_{SS}, is the subset of A that includes all stubborn-set conforming paths.

Theorem 3 *For any goal search A in $\boldsymbol{sch}_\Sigma$, there is a morphism $A \rightarrow A_{SS}$ in $\boldsymbol{sch}_\Sigma$.*

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 $p = (a_1, \cdots, a_n)$ 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 $q = (a_i, a_1, \cdots, a_{i-1}, a_{i+1}, \cdots, a_n)$ 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 A_{SS} 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 $C \in PDG(s)$. For any action b expanded by EC, and actions $b_1, \cdots, b_k$ that do not associate with a DTG in C, if $(b_1, \cdots, b_k, b)$ is a prefix of a path to goal, then we know $b \Rightarrow b_i$ for $i = 1, \cdots, k$ from Lemma 2. Therefore, $(b, b_1, \cdots, b_k)$ 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 $\boldsymbol{sch}_\Sigma$, 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-commutativness 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 $a_1, \cdots, a_n$, SP will expand each of $a_i, i = 1, \cdots, n$ and for each a_i may prune some actions b_i if $b_i \Rightarrow a_i$. 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 $P_aD_b \neq \emptyset$ or $P_bD_a \neq \emptyset$ or $P_bA_a \neq \emptyset$ or $A_bD_a \neq \emptyset$ or $A_aD_b \neq \emptyset$.*

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: (SCC_1, $\cdots$, SCC_N), where 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 $P_bA_a \neq \emptyset$.*

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, $P_aD_b = P_bD_a = P_bA_a = A_bD_a = A_aD_b = \emptyset$. 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 $b_1, \cdots, b_k \notin T(s)$, if $(b_1, \cdots, b_k, b)$ is a prefix of a path to goal, then $(b, b_1, \cdots, b_k)$ is a valid path from s that leads to the same state as $(b_1, \cdots, b_k, b)$.

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 $\mathbf{sch}_\Sigma$, there is a morphism $A \rightarrow A_{OSS}$ in $\mathbf{sch}_\Sigma$, where A_{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 $b_1, \cdots, b_k$ that are not in T. If $(b_1, \cdots, b_k, b)$ is the prefix of an optimal path, then we know $b \Rightarrow b_i$ for $i = 1, \cdots, k$ from Lemma 4. Therefore, $(b, b_1, \cdots, b_k)$ 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 $\mathbf{sch}_\Sigma$, its AC-reduced search A_{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_{AC} and then to A_{AC^+}, all action-preserving.

Theorem 7 *For any optimal search A in $\mathbf{sch}_\Sigma$, its AC$^+$-reduced search A_{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-depended.

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 $\mathbf{sch}_\Sigma$.

Domains	Fast Downward			AC$^+$		
	Expanded	Generated	Time	Expanded	Generated	Time
free1	32	190	0.07	30	322	0.16
free2	42	262	0.06	56	108	0.58
free3	53	397	0.28	72	693	0.36
free4	116	533	0.24	108	542	0.16
free5	796	4191	1.67	1808	14858	3.44
free6	390	2825	1.68	475	4608	3.1
free7	535	3281	2.42	534	4246	3.34
free8	2379	10110	10.72	532	6818	2.79
free9	5754	53638	23.83	428	4218	2.23
free10	2052	14510	15.69	902	11410	12.88
free11	2406	9001	16.7	721	6961	11.83
free12	1362	8013	9.52	634	6396	4.21
free13	12083	77311	138.41	12849	125532	111.94
free14	4431	40529	46.83	605	7558	6.13
free15	35329	307397	463.72	2841	32298	48.39
free16	-	-	-	11757	146286	171.75
free17	657	4870	12.3	330	3104	7.06
pipe1	23	115	0.01	18	138	0.04
pipe2	158	709	0.02	139	870	0.06
pipe3	184	2666	0.31	70	1026	0.3
pipe4	202	2712	0.1	139	2420	0.34
pipe5	47	701	0.23	40	906	0.36
pipe6	64	930	0.13	68	1290	0.14
pipe7	358	15371	1.69	522	24219	1.38
pipe8	1781	70706	3.33	760	25576	1.64
pipe9	1373	43171	4.23	1478	64516	3.51
pipe10	476729	13794006	1499.7	646	25811	2.95
pipe11	303622	1493750	235.9	1706	9042	1.13
pipe12	228593	1783627	350.94	680	22778	3.01
pipe13	62797	555162	117.6	474	6265	2.19
pipe14	177670	972962	193.52	93385	452848	80.24
pipe15	260672	1306367	254.71	27651	130042	20.17
pipe18	7807	98160	46.31	6547	96915	31.16
pipe19	218224	2290321	396.64	5344	44078	34.35

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

Barr, M., and Wells, C. 1990. *Category Theory for Computer Science*. Les Publications CRM.

Chen, Y., and Guo, Y. 2009. Completeness and optimality preserving reduction for planning. In *Proc. IJCAI*.

Chen, Y.; Xu, Y.; and Guo, Y. 2009. Stratified planning. In *Proc. IJCAI*.

Helmert, M., and Röger, G. 2008. How good is almost perfect. In *Proc. AAAI*.

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

Valmari, A. 1989. Stubborn sets for reduced state space generation. In *Proceedings of the 10th International Conference on Applications and Theory of Petri Nets*.

Domains	Fast Downward			Stratified Planning			AC			AC^+		
	Expanded	Generated	Time	Expanded	Generated	Time	Expanded	Generated	Time	Expanded	Generated	Time
driverlog11	280	2858	0.07	215	998	0.04	254	4240	0.07	173	1842	0.06
driverlog12	1810	21582	0.11	2380	8719	0.26	1150	18808	0.22	326	3484	0.13
driverlog13	599	7155	0.18	402	2126	0.09	635	11634	0.2	324	3984	0.15
driverlog14	527	6173	0.18	370	1723	0.1	555	12568	0.21	271	3136	0.11
driverlog15	1288	18823	0.45	972	6202	0.35	2393	74680	1.23	383	5928	0.19
driverlog16	439226	8831575	105.71	192324	1400388	70.93	379769	12644130	72.91	-	-	-
driverlog17	9211	303992	5.44	5438	63710	4.48	3765	190376	2.57	-	-	-
driverlog18	13524	353873	17.2	21620	163436	24.94	38682	1704738	37.2	2867	41254	2.96
truck6	339	5071	0.07	339	2506	0.09	256	6414	0.08	296	5454	0.23
truck7	38532	165934	2.23	38532	81317	2.43	39782	180080	1.13	220179	1194652	7.28
truck8	1966	11558	0.37	1970	5749	0.2	537	7578	0.25	123793	770216	6.01
truck9	236058	2023106	17.29	236058	1002496	28.34	19809	102328	1.9	2584060	15899640	241.84
truck10	325002	3064955	29.82	325002	1519147	47.03	215737	1039650	16.58	468971	2091052	33.98
truck11	99902	1542311	10.45	99902	766989	16.85	77034	449580	6.5	422792	2518740	35.2
depot1	23	91	0.01	18	28	0	44	392	0	-	-	-
depot2	65	485	0.02	84	173	0.01	85	1022	0.05	-	-	-
depot3	6121	48336	1.19	819	2097	0.17	7277	75322	1.62	1762	11454	0.19
depot4	9291	78046	2.64	10366	25840	2.92	8004	87924	1.94	4786	30894	0.76
depot5	343364	2884118	103.36	30538	72874	13.98	22871	241254	6.29	14401	93618	4.07
depot7	28204	261112	3.83	40997	109669	8.45	18740	217698	2.83	8000	55848	0.85
depot8	162784	1674483	55.54	108157	349662	45.67	569532	8150860	137.4	33672	290238	7.9
depot9	-	-	-	165286	572968	226.26	192483	2877660	138.3	28454	216286	19.7
depot10	51542	726134	14.06	35314	121058	13.78	31735	552866	6.79	482	5162	0.18
depot11	215106	3316774	198.47	119489	456139	135.1	205746	3677638	122.57	1577	18828	1.23
depot13	257	4045	0.14	179	730	0.17	265	5480	0.46	5943	62406	2.23
depot14	-	-	-	-	-	-	274978	2832700	217.26	-	-	-
depot15	-	-	-	-	-	-	46357	478958	106.83	-	-	-
depot16	-	-	-	393419	2176390	257.24	653345	15021200	249.81	65774	667106	21.28
storage1	4	7	0	-	-	-	4	14	0	4	14	0
storage2	4	9	0	4	3	0	4	18	0	4	18	0
storage3	4	11	0	4	5	0	4	22	0.01	4	22	0.02
storage4	32	85	0.02	35	34	0	32	170	0.02	32	170	0.02
storage5	20	94	0.01	21	43	0.02	20	188	0.03	19	130	0.02
storage6	31	176	0.04	30	75	0.02	31	352	0.04	31	260	0.05
storage7	235	634	0.02	240	248	0.04	233	1250	0.05	227	1216	0.05
storage8	95	480	0.07	90	200	0.04	109	1064	0.08	189	1108	0.09
storage9	93	744	0.04	91	355	0.04	93	1488	0.1	159	1342	0.12
storage10	1521	4364	0.23	1372	1479	0.2	1494	8620	0.18	1516	8700	0.18
storage11	297	1774	0.18	326	893	0.1	293	3566	0.1	2066	11398	0.68
storage12	1496	11550	0.77	357	1511	0.15	1323	20094	0.55	1752	10583	0.31
storage13	4930	17046	1.31	5697	7565	1.01	5414	37662	0.71	8160	55656	1.33
storage14	2668	18730	1.11	2459	8029	0.71	2837	39492	0.86	1263	8164	0.55
storage15	325	2673	0.36	355	1267	0.2	308	5048	0.38	2583	20806	0.52
storage16	276	3385	0.64	273	1591	0.27	289	1542	0.23	259	2974	0.34
tpp1	6	8	0	6	3	0	6	16	0	6	16	0
tpp2	9	17	0	9	6	0	11	34	0	11	34	0
tpp3	12	29	0	12	12	0	16	54	0	16	54	0
tpp4	15	44	0	15	18	0	19	72	0	22	78	0
tpp5	22	92	0	22	33	0	88	452	0	122	454	0.01
tpp6	664	3641	0.06	617	1229	0.04	261	1882	0.03	94	444	0.02
tpp7	1591	9403	0.17	2199	5840	0.14	1250	8394	0.03	431	2410	0.04
tpp8	4685	37683	0.29	4181	13904	0.31	932	6670	0.07	486	2642	0.05
tpp9	3630	25924	0.59	4044	9734	0.39	1675	13262	0.18	1177	7460	0.04
tpp10	12242	110251	1.66	9634	32313	1.07	6685	63868	0.22	2339	14382	0.27
tpp11	13148	126912	2.08	24193	95847	3.82	5973	51660	1.11	1621	9614	0.26
tpp12	36690	364082	4.48	23754	71293	3.51	18366	154012	1.41	5195	47802	0.87
tpp13	24066	295068	4.35	32150	156888	8.34	26175	412436	3.44	16375	292526	2.07
tpp14	68494	894865	14.74	52963	209880	19.1	42991	643584	6.61	16982	175226	2.23
tpp15	37145	477808	8.71	54522	252323	20.85	28275	403406	4.54	16506	171188	2.23

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.