

Principles of AI Planning

14. Planning as search: Partial-Order Reduction

Albert-Ludwigs-Universität Freiburg



**UNI
FREIBURG**

Bernhard Nebel and Robert Mattmüller

January 13th, 2017



Motivation

Motivation

Preliminaries

Stubborn
Sets

Conclusion

- **Worst case:** Heuristic search may explore **exponentially** more states than necessary, even if heuristic is **almost perfect** (Helmert and Röger, 2008).
- **Example:** A* search in GRIPPER domain explores all permutations of ball transportations if heuristic is off only by a small constant.
- **Idea:** Complement heuristic search with **orthogonal technique(s)** to reduce size of explored state space.
- **Desired properties of this technique:** preservation of **completeness** and, if possible, **optimality**.

Motivation

Preliminaries

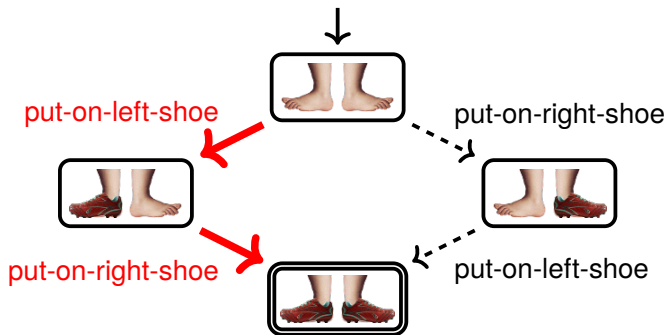
Stubborn
Sets

Conclusion

Idea:

- Enforce particular ordering among operators.
- Ignore all other orderings.

Example



Motivation

Preliminaries

Stubborn
Sets

Conclusion



Preliminaries

Motivation

Preliminaries

Setting

Operator

Dependencies

Necessary

Enabling Sets and

Disjunctive Action

Landmarks

Stubborn

Sets

Conclusion

Assumption: For the rest of the chapter, we assume that all planning tasks are SAS⁺ planning tasks $\Pi = (V, I, O, \gamma)$.

For convenience, we assume that operators have the form $o = \langle pre(o), eff(o) \rangle$, where $pre(o)$ and $eff(o)$ are both **partial states** over V , i.e., partial functions mapping variables v to values in \mathcal{D}_v . Similarly, we assume that γ is a partial state describing the goal.

Example

Operator $o = \langle pre(o), eff(o) \rangle$ with

- $pre(o) = \{v_1 \mapsto d_1, v_5 \mapsto d_5\}$ and
- $eff(o) = \{v_2 \mapsto d_2, v_3 \mapsto d_3\}$

corresponds to $o = \langle \chi, e \rangle$ with

$\chi = (v_1 = d_1 \wedge v_5 = d_5)$ and $e = (v_2 := d_2 \wedge v_3 := d_3)$.

Motivation

Preliminaries

Setting

Operator
Dependencies

Necessary
Enabling Sets and
Disjunctive Action
Landmarks

Stubborn
Sets

Conclusion

Definition (Operators)

Let $\Pi = (V, I, O, \gamma)$ be a SAS⁺ planning task and $o \in O$ an operator. Then

- $prevars(o) := vars(pre(o))$ are the variables that occur in the precondition of o .
- $effvars(o) := vars(eff(o))$ are the variables that occur in the effect of o .
- o **reads** $v \in V$ iff $v \in prevars(o)$.
- o **modifies** $v \in V$ iff $v \in effvars(o)$.

Variable $v \in V$ is **goal-related** iff $v \in vars(\gamma)$.

Assumption: $effvars(o) \neq \emptyset$ for all $o \in O$.

Motivation

Preliminaries

Setting

Operator

Dependencies

Necessary

Enabling Sets and

Disjunctive Action

Landmarks

Stubborn

Sets

Conclusion

Definition (Operator dependencies)

Let $\Pi = \langle V, O, I, \gamma \rangle$ be a planning task and $o, o' \in O$.

- 1 o **disables** o' iff there exists $v \in \text{effvars}(o) \cap \text{prevars}(o')$ such that $\text{eff}(o)(v) \neq \text{pre}(o')(v)$.
- 2 o **enables** o' iff there exists $v \in \text{effvars}(o) \cap \text{prevars}(o')$ such that $\text{eff}(o)(v) = \text{pre}(o')(v)$.
- 3 o and o' **conflict** iff there is $v \in \text{effvars}(o) \cap \text{effvars}(o')$ such that $\text{eff}(o)(v) \neq \text{eff}(o')(v)$.
- 4 o and o' **interfere** iff o disables o' , or o' disables o , or o and o' conflict.
- 5 o and o' are **commutative** iff o and o' do not interfere, and neither o enables o' , nor o' enables o .

Motivation

Preliminaries

Setting

Operator
Dependencies

Necessary
Enabling Sets and
Disjunctive Action
Landmarks

Stubborn
Sets

Conclusion

Example

put-on-left = $\langle \text{pos} = \text{home} \wedge \text{left} = \text{f}, \text{left} := \text{t} \rangle$

put-on-right = $\langle \text{pos} = \text{home} \wedge \text{right} = \text{f}, \text{right} := \text{t} \rangle$

go-to-uni = $\langle \text{left} = \text{t} \wedge \text{right} = \text{t}, \text{pos} := \text{uni} \rangle$

go-to-gym = $\langle \text{left} = \text{t} \wedge \text{right} = \text{t}, \text{pos} := \text{gym} \rangle$

Then:

- go-to-uni and go-to-gym disable put-on-left and put-on-right.
- put-on-left and put-on-right enable go-to-uni and go-to-gym.
- go-to-uni and go-to-gym conflict.
- put-on-left and put-on-right are commutative.

Motivation

Preliminaries

Setting

Operator

Dependencies

Necessary

Enabling Sets and

Disjunctive Action

Landmarks

Stubborn

Sets

Conclusion

Definition (Necessary enabling set)

Let $\Pi = \langle V, I, O, \gamma \rangle$ be a planning task, s a state, and $o \in O$ an operator that is not applicable in s . A set N of operators is a **necessary enabling set** (NES) for o in s if all operator sequences that lead from s to a goal state and include o contain an operator in N before the first occurrence of o .

Note: NESs not uniquely determined for given o and s .
(E.g., supersets of NESs are still NESs.)

Motivation

Preliminaries

Setting

Operator

Dependencies

Necessary

Enabling Sets and

Disjunctive Action

Landmarks

Stubborn

Sets

Conclusion

Necessary Enabling Sets and Disjunctive Action Landmarks



Definition (Disjunctive action landmark)

Let $\Pi = \langle V, I, O, \gamma \rangle$ be a planning task and s a state. A **disjunctive action landmark** (DAL) L in s is a set of operators such that all operator sequences that lead from s to a goal state contain some operator in L .

Note: This is really the same definition as in the chapter on LM-cut. Only the abbreviation “DAL” occurs here for the first time.

Observation

For state s and operator o that is not applicable in s , disjunctive action landmarks for task $\langle V, I, O, pre(o) \rangle$ are necessary enabling sets for o in s .

Motivation

Preliminaries

Setting

Operator

Dependencies

Necessary

Enabling Sets and
Disjunctive Action
Landmarks

Stubborn
Sets

Conclusion

Proof

Let L be such a disjunctive action landmark.

Then each operator sequence that leads from s to a state satisfying $pre(o)$ contains some operator in L .

Thus, each operator sequence that leads from s to a goal state and includes o contains an operator in L before the first occurrence of o .

Therefore, L is an NES for o in s .

Motivation

Preliminaries

Setting

Operator

Dependencies

Necessary

Enabling Sets and
Disjunctive Action
Landmarks

Stubborn
Sets

Conclusion



Stubborn Sets

Motivation

Preliminaries

**Stubborn
Sets**

Strong Stubborn
Sets

Active Operators

Weak Stubborn
Sets

Algorithms

Properties of
Stubborn Sets

Some Experiments

Conclusion

Back to the motivation:

If, in state s , some set of operators can be **applied in any order** and the order does not matter, we want to **commit to one such order** and **ignore all other orders**.

Idea:

Identify operators that can be “postponed” since they are independent of all operators that are not “postponed”.

E.g., put-on-right could be postponed, since it is independent of put-on-left (that is not postponed).

Motivation

Preliminaries

Stubborn Sets

Strong Stubborn Sets

Active Operators

Weak Stubborn Sets

Algorithms

Properties of Stubborn Sets

Some Experiments

Conclusion

Idea (more precisely): Identify operators that **should not** be postponed, and postpone the rest.

Question: When should we consider an operator o **too important to be tentatively ignored?**

Answer:

- 1 Base case:** If o may be immediately relevant to reaching (part of) the goal, or
- 2 Inductive case I:** If o may be immediately relevant to contributing to making an operator applicable that we consider too important to be tentatively ignored, or
- 3 Inductive case II:** If o might not be applicable any more if we postponed it, or if its effect might conflict with the effect of an operator that we considered too important to be tentatively ignored ($\approx o$ interferes with such an operator).

Motivation

Preliminaries

Stubborn Sets

Strong Stubborn Sets

Active Operators

Weak Stubborn Sets

Algorithms

Properties of Stubborn Sets

Some Experiments

Conclusion

Let's formalize the above answer:

Definition (Strong stubborn set)

Let $\Pi = \langle V, I, O, \gamma \rangle$ be a planning task and s a state. A set $T_s \subseteq O$ is a **strong stubborn set in s** if

- 1 T_s contains a disjunctive action landmark in s , and
- 2 for all $o \in T_s$ that are not applicable in s , T_s contains a necessary enabling set for o and s , and
- 3 for all $o \in T_s$ that are applicable in s , T_s contains all operators that interfere with o .

Instead of applying all applicable operators in s only apply those that are applicable and contained in T_s .

Motivation

Preliminaries

Stubborn
Sets

Strong Stubborn
Sets

Active Operators

Weak Stubborn
Sets

Algorithms

Properties of
Stubborn Sets

Some Experiments

Conclusion

Example

$$I = \{\text{pos} \mapsto \text{home}, \text{left} \mapsto \text{f}, \text{right} \mapsto \text{f}\}, \quad \gamma = \{\text{pos} \mapsto \text{uni}\}$$

$$\text{put-on-left} = \langle \text{pos} = \text{home} \wedge \text{left} = \text{f}, \text{left} := \text{t} \rangle$$

$$\text{put-on-right} = \langle \text{pos} = \text{home} \wedge \text{right} = \text{f}, \text{right} := \text{t} \rangle$$

$$\text{go-to-uni} = \langle \text{left} = \text{t} \wedge \text{right} = \text{t}, \text{pos} := \text{uni} \rangle$$

- Step 1: DAL in I is $\{\text{go-to-uni}\} \rightsquigarrow T_S := \{\text{go-to-uni}\}$.
- Step 2: go-to-uni not applicable in I . One possible NES for go-to-uni in I is $\{\text{put-on-left}\} \rightsquigarrow T_S := T_S \cup \{\text{put-on-left}\}$.
- Step 3: put-on-left is applicable in I . The only operator that interferes with it, go-to-uni , is already in T_S .
- Hence, $T_S = \{\text{go-to-uni}, \text{put-on-left}\}$, and T_S restricted to the applicable operators is $\{\text{put-on-left}\}$. During search, only apply put-on-left (not put-on-right).

Motivation

Preliminaries

Stubborn
Sets

Strong Stubborn
Sets

Active Operators

Weak Stubborn
Sets

Algorithms

Properties of
Stubborn Sets

Some Experiments

Conclusion

Example

Let $V = \{u_1, u_2, v, w\}$, $I = \{u_1 \mapsto 0, u_2 \mapsto 0, v \mapsto 0, w \mapsto 0\}$,
 $\gamma = \{v \mapsto 0, u_1 \mapsto 1, u_2 \mapsto 1\}$, and $O = \{o_1, o_2, o_3\}$, where:

- $o_1 = \langle u_1 = 0, u_1 := 1 \wedge w := 2 \rangle$,
- $o_2 = \langle u_2 = 0, u_2 := 1 \wedge w := 2 \rangle$,
- $o_3 = \langle u_1 = 0 \wedge u_2 = 0, v := 1 \wedge w := 1 \rangle$.

Strong stubborn set:

- Step 1: Include o_1 (or o_2) in T_s as DAL.
- Step 2: Include o_3 in T_s since it interferes with o_1 (or o_2).
- Step 3: Include o_2 (or o_1) in T_s since it interferes with o_3 .

\rightsquigarrow all applicable operators included in T_s , no pruning.

Question: Can we do better than that in this example?

Motivation

Preliminaries

Stubborn
Sets

Strong Stubborn
Sets

Active Operators

Weak Stubborn
Sets

Algorithms

Properties of
Stubborn Sets

Some Experiments

Conclusion

Definition (Domain transition graph)

Let $\Pi = (V, I, O, \gamma)$ be a SAS⁺ planning task and $v \in V$. The

domain transition graph for v is the directed graph

$DTG(v) = \langle \mathcal{D}_v, E \rangle$ where $(d, d') \in E$ iff there is an operator

$o \in O$ with

- $eff(o)(v) = d'$, and
- $v \notin prevars(o)$ or $pre(o)(v) = d$.

Motivation

Preliminaries

Stubborn
Sets

Strong Stubborn
Sets

Active Operators

Weak Stubborn
Sets

Algorithms

Properties of
Stubborn Sets

Some Experiments

Conclusion

Example

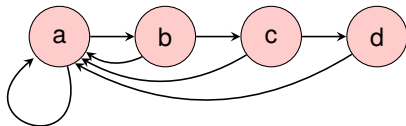
move-a-b = $\langle \text{pos} = \text{a}, \text{pos} := \text{b} \rangle$

move-b-c = $\langle \text{pos} = \text{b}, \text{pos} := \text{c} \rangle$

move-c-d = $\langle \text{pos} = \text{c}, \text{pos} := \text{d} \rangle$

reset = $\langle \top, \text{pos} := \text{a} \wedge \text{othersvar} := \text{othersvar} \rangle$

Then $DTG(\text{pos})$:



Motivation

Preliminaries

Stubborn
Sets

Strong Stubborn
Sets

Active Operators
Weak Stubborn
Sets

Algorithms

Properties of
Stubborn Sets

Some Experiments

Conclusion

Definition (Active operators)

Let $\Pi = \langle V, I, O, \gamma \rangle$ be a planning task and let s be a state. The set of **active operators** $Act(s) \subseteq O$ in s is defined as the set of operators such that for all $o \in Act(s)$:

- For every variable $v \in prevars(o)$, there is a path in $DTG(v)$ from $s(v)$ to $pre(o)(v)$. If v is goal-related, then there is also a path from $pre(o)(v)$ to the goal value $\gamma(v)$.
- For every goal-related variable $v \in effvars(o)$, there is a path in $DTG(v)$ from $eff(o)(v)$ to the goal value $\gamma(v)$.

Motivation

Preliminaries

Stubborn
Sets

Strong Stubborn
Sets

Active Operators
Weak Stubborn
Sets

Algorithms

Properties of
Stubborn Sets

Some Experiments

Conclusion

Proposition

- 1 $Act(s)$ can be identified efficiently for a given state s by considering paths in the projection of Π onto v .
- 2 Operators not in $Act(s)$ can be treated as nonexistent when reasoning about s because they are not applicable in all states reachable from s , or they lead to a dead-end from s .

Proof

- 1 Homework: Specify efficient algorithm for identification of $Act(s)$.
- 2 Obvious. □

Motivation

Preliminaries

Stubborn
Sets

Strong Stubborn
Sets

Active Operators
Weak Stubborn
Sets

Algorithms

Properties of
Stubborn Sets

Some Experiments

Conclusion

Motivation

Preliminaries

Stubborn
Sets

Strong Stubborn
Sets

Active Operators

Weak Stubborn
Sets

Algorithms

Properties of
Stubborn Sets

Some Experiments

Conclusion

Remark 1: Even when excluding **inactive** operators, this preserves completeness and even optimality of a search algorithm (see proof below).

Remark 2: Excluding **inactive** operators can “cascade” in the sense that additional **active** operators need not be considered.

Definition (Strong stubborn set with active operator pruning)

Let $\Pi = \langle V, I, O, \gamma \rangle$ be a planning task and s a state. A set $T_s \subseteq O$ is a **strong stubborn set in s** if

- 1 T_s contains a disjunctive action landmark in s , and
- 2 for all $o \in T_s$ that are not applicable in s , T_s contains a necessary enabling set for o and s , and
- 3 for all $o \in T_s$ that are applicable in s , T_s contains all operators that **are active in s and** interfere with o .

Instead of applying all applicable operators in s only apply those that are applicable and contained in T_s .

Motivation

Preliminaries

Stubborn Sets

Strong Stubborn Sets

Active Operators

Weak Stubborn Sets

Algorithms

Properties of Stubborn Sets

Some Experiments

Conclusion

Recall the previous example where strong stubborn sets without active operator pruning were useless.

Example

- $I = \{u_1 \mapsto 0, u_2 \mapsto 0, v \mapsto 0, w \mapsto 0\}$,
 $\gamma = \{v \mapsto 0, u_1 \mapsto 1, u_2 \mapsto 1\}$
- $o_1 = \langle u_1 = 0, u_1 := 1 \wedge w := 2 \rangle$
- $o_2 = \langle u_2 = 0, u_2 := 1 \wedge w := 2 \rangle$
- $o_3 = \langle u_1 = 0 \wedge u_2 = 0, v := 1 \wedge w := 1 \rangle$

Now, **with** active operator pruning:

- Step 1: Include o_1 (or o_2) in T_s as DAL.
- Step 2: Operator o_3 is not active in any reachable state.
 $\rightsquigarrow o_3$ not in T_s , although it interferes with o_1 (or o_2).

Motivation

Preliminaries

Stubborn
Sets

Strong Stubborn
Sets

Active Operators
Weak Stubborn
Sets

Algorithms

Properties of
Stubborn Sets

Some Experiments

Conclusion

Example (Example, ctd.)

Now, **with** active operator pruning:

- Step 1: Include o_1 (or o_2) in T_s as DAL.
- Step 2: Operator o_3 is not active in any reachable state.
 $\rightsquigarrow o_3$ not in T_s , although it interferes with o_1 (or o_2).
- Hence, e. g., $T_s = \{o_1\}$ strong stubborn set (with active operator pruning) in I .
- Even **active** operator o_2 is not included in $T_s = \{o_1\}$.

\rightsquigarrow some pruning occurs.

Motivation

Preliminaries

Stubborn
Sets

Strong Stubborn
Sets

Active Operators
Weak Stubborn
Sets

Algorithms

Properties of
Stubborn Sets

Some Experiments

Conclusion

With **weak** stubborn sets, some operators that disable an operator in T_s need not be included in T_s .

Therefore, weak stubborn sets potentially allow more pruning than strong stubborn sets.

Definition (Weak stubborn set)

Let $\Pi = \langle V, I, O, \gamma \rangle$ be a planning task and s a state. A set $T_s \subseteq O$ is a **weak stubborn set in s** if

- 1 T_s contains a disjunctive action landmark in s , and
- 2 for all $o \in T_s$ that are not applicable in s , T_s contains a necessary enabling set for o and s , and
- 3 for all $o \in T_s$ that are applicable in s , T_s contains the active operators in s that have conflicting effects with o or that are disabled by o .

Motivation

Preliminaries

Stubborn Sets

Strong Stubborn Sets

Active Operators

Weak Stubborn Sets

Algorithms

Properties of Stubborn Sets

Some Experiments

Conclusion



For weak stubborn sets, it suffices to include active operators o' that **are disabled** or **conflict** with applicable operators $o \in T_S$. However, o' **does not need to be included** if o' **disables** an applicable operator $o \in T_S$.

No computational overhead of computing weak stubborn sets over computing strong stubborn sets.

Theorem

In the best case, weak stubborn sets admit **exponentially more pruning** than strong stubborn sets.

Proof

Homework. □

Motivation

Preliminaries

Stubborn
Sets

Strong Stubborn
Sets

Active Operators

**Weak Stubborn
Sets**

Algorithms

Properties of
Stubborn Sets

Some Experiments

Conclusion

compute-DAL: Compute a disjunctive action landmark.

Procedure compute-DAL

```
def compute-DAL( $\gamma$ ):  
  select  $v \in vars(\gamma)$  with  $s(v) \neq \gamma(v)$   
   $L \leftarrow \{o' \in Act(s) \mid eff(o')(v) = \gamma(v)\}$   
  return  $L$ 
```

Selection of $v \in vars(\gamma)$ arbitrary. Any variable will do.
Selection heuristics?

Motivation

Preliminaries

Stubborn
Sets

Strong Stubborn
Sets

Active Operators

Weak Stubborn
Sets

Algorithms

Properties of
Stubborn Sets

Some Experiments

Conclusion

compute-NES: Compute a necessary enabling set.

Precedure compute-NES

def compute-NES(o, s):

 select $v \in \text{prevars}(o)$ with $s(v) \neq \text{pre}(o)(v)$

$N \leftarrow \{o' \in \text{Act}(s) \mid \text{eff}(o')(v) = \text{pre}(o)(v)\}$

return N

Selection of $v \in \text{prevars}(o)$ arbitrary. Any variable will do.

Selection heuristics?

Motivation

Preliminaries

Stubborn
Sets

Strong Stubborn
Sets

Active Operators

Weak Stubborn
Sets

Algorithms

Properties of
Stubborn Sets

Some Experiments

Conclusion



compute-interfering-operators: Compute interfering operators.

Procedure compute-interfering-operators (for strong SS)

```
def compute-interfering-operators( $o$ ):  
   $\text{disablers} \leftarrow \{o' \in O \mid o' \text{ disables } o\}$   
   $\text{disablees} \leftarrow \{o' \in O \mid o \text{ disables } o'\}$   
   $\text{conflicting} \leftarrow \{o' \in O \mid o \text{ and } o' \text{ conflict}\}$   
  return  $\text{disablers} \cup \text{disablees} \cup \text{conflicting}$ 
```

Procedure compute-interfering-operators (for weak SS)

```
def compute-interfering-operators( $o$ ):  
   $\text{disablees} \leftarrow \{o' \in O \mid o \text{ disables } o'\}$   
   $\text{conflicting} \leftarrow \{o' \in O \mid o \text{ and } o' \text{ conflict}\}$   
  return  $\text{disablees} \cup \text{conflicting}$ 
```

Motivation

Preliminaries

Stubborn
Sets

Strong Stubborn
Sets

Active Operators

Weak Stubborn
Sets

Algorithms

Properties of
Stubborn Sets

Some Experiments

Conclusion

Computing (strong and weak) stubborn sets for planning can be achieved with a **fixpoint iteration** until the constraints of T_s are satisfied:

compute-stubborn-set: Compute (strong or weak) stubborn set.

Precedure **compute-stubborn-set**

def compute-stubborn-set(s):

$T_s \leftarrow \text{compute-DAL}(\gamma)$

while no fixed-point of T_s reached **do**

for $o \in T_s$ applicable in s:

$T_s \leftarrow T_s \cup \text{compute-interfering-operators}(o)$

for $o \in T_s$ not applicable in s:

$T_s \leftarrow T_s \cup \text{compute-NES}(o, s)$

end while

return T_s

Motivation

Preliminaries

Stubborn
Sets

Strong Stubborn
Sets

Active Operators

Weak Stubborn
Sets

Algorithms

Properties of
Stubborn Sets

Some Experiments

Conclusion

Observation: stubborn sets are state-dependent, but not path-dependent.

This allows filtering the applicable operators in s in graph search algorithms like A* that perform duplicate detection, too.

Instead of applying all applicable operators $app(s)$ in s , only apply operators in $T_{app(s)} := T_s \cap app(s)$.

Motivation

Preliminaries

Stubborn Sets

Strong Stubborn Sets

Active Operators

Weak Stubborn Sets

Algorithms

Properties of Stubborn Sets

Some Experiments

Conclusion



Theorem

Weak stubborn sets are completeness and optimality preserving.

Proof

Let $T_{app(s)} := T_s \cap app(s)$ for a weak stubborn set T_s .

We show that for all states s from which an optimal plan consisting of $n > 0$ operators exists, $T_{app(s)}$ contains an operator that starts such a plan.

We show by induction that A^* restricting successor generation to $T_{app(s)}$ is optimal.

Let T_s be a weak stubborn set and $\pi = o_1, \dots, o_n$ be an optimal plan that starts in s .

...

Motivation

Preliminaries

Stubborn Sets

Strong Stubborn Sets

Active Operators

Weak Stubborn Sets

Algorithms

Properties of Stubborn Sets

Some Experiments

Conclusion



Proof (ctd.)

As T_S contains a disjunctive action landmark, π must contain an operator from T_S .

Let o_k be the operator with smallest index in π that is also contained in T_S , i.e., $o_k \in T_S$ and $\{o_1, \dots, o_{k-1}\} \cap T_S = \emptyset$.

We observe:

1. $o_k \in \text{app}(s)$: otherwise by definition of weak stubborn sets, a necessary enabling set N for o_k in s would have to be contained in T_S , and at least one operator from N would have to occur before o_k in π to enable o_k , contradicting that o_k was chosen with smallest index.
2. ...

Motivation

Preliminaries

Stubborn
Sets

Strong Stubborn
Sets

Active Operators
Weak Stubborn
Sets

Algorithms

Properties of
Stubborn Sets

Some Experiments

Conclusion



Proof (ctd.)

1. ...
2. o_k does not disable any of the operators o_1, \dots, o_{k-1} , and all these operators have non-conflicting effects with o_k : otherwise, as $o_k \in \text{app}(s)$, and by definition of weak stubborn sets, at least one of o_1, \dots, o_{k-1} would have to be contained in T_s , again contradicting the assumption.

Hence, we can move o_k to the front:

$o_k, o_1, \dots, o_{k-1}, o_{k+1}, \dots, o_n$ is also a plan for Π .

It has the same cost as π and is hence optimal.

Thus, we have found an optimal plan of length n started by an operator $o_k \in T_{\text{app}(s)}$, completing the proof. \square

Motivation

Preliminaries

Stubborn
Sets

Strong Stubborn
Sets

Active Operators

Weak Stubborn
Sets

Algorithms

Properties of
Stubborn Sets

Some Experiments

Conclusion



Remark: The argument to move o_k to the front also holds for strong stubborn sets: in this case, o_k is not even disabled by any of o_1, \dots, o_{k-1} (and hence, o_k is independent of o_1, \dots, o_{k-1}), which is a stronger property than needed in the proof.

Corollary

Strong stubborn sets are completeness and optimality preserving.



Motivation

Preliminaries

Stubborn
Sets

Strong Stubborn
Sets

Active Operators

Weak Stubborn
Sets

Algorithms

Properties of
Stubborn Sets

Some Experiments

Conclusion

Some Experiments: Overview

Optimal Planning, A* with LM-cut Heuristic, Selected Domains



UNI
FREIBURG

Domain (problems)	Coverage		Nodes generated	
	A*	+SSS	A*	+SSS
PARCPRINTER-08 (30)	18	+12	2455181	<1%
PARCPRINTER-OPT11 (20)	13	+7	2454533	<1%
WOODWORKING-OPT08 (30)	17	+10	26796212	<1%
WOODWORKING-OPT11 (20)	12	+7	26795517	<1%
SATELLITE (36)	7	+5	5116312	2%
ROVERS (40)	7	+2	1900691	22%
AIRPORT (50)	28	±0	545072	93%
OPENSTACKS-OPT08 (30)	19	+2	56584063	51%
OPENSTACKS-OPT11 (20)	14	+2	56456969	51%
DRIVERLOG (20)	13	+1	3679376	82%
SCANALYZER-08 (30)	15	-3	14203012	100%
SCANALYZER-OPT11 (20)	12	-3	14202884	100%
PARKING-OPT11 (20)	3	-1	560914	100%
SOKOBAN-OPT08 (30)	30	-1	20519270	100%
VISITALL-OPT11 (20)	11	-1	1991169	100%
REMAINING DOMAINS (980)	544	±0	436017004	93%
SUM (1396)	763	+39	670278179	77%

Motivation

Preliminaries

Stubborn
Sets

Strong Stubborn
Sets

Active Operators

Weak Stubborn
Sets

Algorithms

Properties of
Stubborn Sets

Some Experiments

Conclusion

Some Experiments

Weak compared to strong stubborn sets



Motivation

Preliminaries

Stubborn
Sets

Strong Stubborn
Sets

Active Operators

Weak Stubborn
Sets

Algorithms

Properties of
Stubborn Sets

Some Experiments

Conclusion

Domain (problems)	Coverage		Nodes generated		# problems w. diff. gen.
	SSS	WSS	SSS	WSS	
OPENSTACKS-OPT08 (30)	21	± 0	152711917	99.936%	18
OPENSTACKS-OPT11 (20)	16	± 0	152642101	99.936%	16
PATHWAYS-NONEG (30)	5	± 0	162347	99.702%	2
PSR-SMALL (50)	49	± 0	18119489	99.998%	6
SATELLITE (36)	12	± 0	70299721	92.804%	12

⇒ In practice (or, at least, in the standard benchmark problems) there is no significant difference between weak and strong stubborn sets.



Motivation

Preliminaries

Stubborn
Sets

Conclusion

Conclusion

- Need for **techniques orthogonal to heuristic search**, complementing heuristics.
- One idea: **Commit to one order of operators** if they are independent. Prune other orders.
- Class of such techniques: **partial-order reduction** (POR)
- One such technique: **strong/weak stubborn sets**
- Can lead to **substantial pruning** compared to plain A^* .
- Many other POR techniques exist.
- Other pruning techniques exist as well, e.g., symmetry reduction.

Motivation

Preliminaries

Stubborn
Sets

Conclusion