Motivation

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

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

Example:

- wear-left-shoe
- wear-right-shoe
- wear-right-shoe
- wear-left-shoe
2 Preliminaries

- Setting
- Basic Definitions
- Operator Dependencies
- Active Operators
- Necessary Enabling Sets and Disjunctive Action
- Landmarks
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 $D_v$. Similarly, we assume that $\gamma$ 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

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

**Definition (Operators)**

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

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

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

**Assumption:** $\text{effvars}(o) \neq \emptyset$ for all $o \in O$. 
Domain Transition Graphs

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

- $\text{eff}(o)(v) = d'$, and
- $v \notin \text{prevars}(o)$ or $\text{pre}(o)(v) = d$.
Domain Transition Graphs

Example

\[
\begin{align*}
\text{move-a-b} & = \langle \text{pos} = a, \text{pos} := b \rangle \\
\text{move-b-c} & = \langle \text{pos} = b, \text{pos} := c \rangle \\
\text{move-c-d} & = \langle \text{pos} = c, \text{pos} := d \rangle \\
\text{reset} & = \langle \top, \text{pos} := a \land \text{othervar} := \text{otherval} \rangle
\end{align*}
\]

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

\[
\begin{array}{c}
da \\
\downarrow \\
b \\
\downarrow \\
c \\
\downarrow \\
d
\end{array}
\]

\[
\begin{array}{c}
a \\
\downarrow \\
b \\
\downarrow \\
c \\
\downarrow \\
d
\end{array}
\]

January 8th, 2016 B. Nebel, R. Mattmüller – AI Planning 11 / 42
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 $o$. 
Operator Dependencies

Example

wear-left = \langle pos = home \land left = f, left := t \rangle
wear-right = \langle pos = home \land right = f, right := t \rangle
go-to-uni = \langle left = t \land right = t, pos := uni \rangle
go-to-gym = \langle left = t \land right = t, pos := gym \rangle

Then:

- go-to-uni and go-to-gym disable wear-left and wear-right.
- wear-left and wear-right enable go-to-uni and go-to-gym.
- go-to-uni and go-to-gym conflict.
- wear-left and wear-right are commutative.
Active Operators

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 $\text{Act}(s) \subseteq O$ in $s$ is defined as the set of operators such that for all $o \in \text{Act}(s)$:

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

2. For every goal-related variable $v \in \text{effvars}(o)$, there is a path in $DTG(v)$ from $\text{eff}(o)(v)$ to the goal value $\gamma(v)$. 
Active Operators

Proposition

1. $\text{Act}(s)$ can be identified efficiently for a given state $s$ by considering paths in the projection of $\Pi$ onto $\nu$.

2. Operators not in $\text{Act}(s)$ 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 $\text{Act}(s)$.

2. Obvious.
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., supertsets of NESs are still NESs.)
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 $L$.

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 $s$. 
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 $s$. 
3 Stubborn Sets

- Strong Stubborn Sets
- Weak Stubborn Sets
- Algorithms
- Properties of Stubborn Sets
- Some Experiments
Motivation

Preliminaries

Stubborn Sets

Strong Stubborn Sets

Weak Stubborn Sets

Algorithms

Properties of Stubborn Sets

Some Experiments

Conclusion

Stubborn Sets

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.

One idea:

Identify operators that can be “postponed” since they are independent of all operators that are not “postponed”. E.g., wear-right-shoe could be postponed, since it is independent of wear-left-shoe (that is not postponed).

Second idea (roughly):

Identify operators that have to be applied and cannot be postponed because they are not independent of other operators also not postponed.
Strong Stubborn Sets

Following the second idea:

First attempt at a definition:

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

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

Following the second idea:

Improved attempt at a definition:

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 applicable in $s$, $T_s$ contains all operators that are active in $s$ and interfere with $o$, and
3. for all $o \in T_s$ that are not applicable in $s$, $T_s$ contains a necessary enabling set for $o$ and $s$.

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

Let $\Pi = \langle V, I, O, \gamma \rangle$ be a planning task with the following components:

- $V = \{u_1, u_2, v, w\}$
- $O = \{o_1, o_2, o_3\}$
- $pre(o_1) = \{u_1 \mapsto 0\}$, $eff(o_1) = \{u_1 \mapsto 1, w \mapsto 2\}$
- $pre(o_2) = \{u_2 \mapsto 0\}$, $eff(o_2) = \{u_2 \mapsto 1, w \mapsto 2\}$
- $pre(o_3) = \{u_1 \mapsto 0, u_2 \mapsto 0\}$, $eff(o_3) = \{v \mapsto 1, w \mapsto 1\}$
- $I = \{u_1 \mapsto 0, u_2 \mapsto 0, v \mapsto 0, w \mapsto 0\}$
- $\gamma = \{v \mapsto 0, u_1 \mapsto 1, u_2 \mapsto 1\}$
Example

- **Case 1** (first attempt at definition where non-active interfering operators are included in $T_s$):
  - Include $o_1$ (or $o_2$) in $T_s$ as disjunctive action landmark.
  - Include $o_3$ in $T_s$ since it interferes with $o_1$ (or $o_2$).
  - Include $o_2$ (or $o_1$) in $T_s$ since it interferes with $o_3$.

  $\Rightarrow$ all applicable operators included in $T_s$, no pruning.

- **Case 2** (improved attempt without non-active interfering operators):
  - $o_3$ is not active in any reachable state.
  - $T_s = \{o_1\}$ strong stubborn set in $I$.
  - Even active operator $o_2$ is not included in $T_s = \{o_1\}$.

  $\Rightarrow$ nice amount of pruning occurs.
Weak Stubborn Sets

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 applicable in $s$, $T_s$ contains the active operators in $s$ that have conflicting effects with $o$ or that are disabled by $o$, and
3. for all $o \in T_s$ that are not applicable in $s$, $T_s$ contains a necessary enabling set for $o$ and $s$. 
Weak Stubborn Sets

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.
compute-DAL: Compute a disjunctive action landmark.

**Procedure compute-DAL**

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

Selection of \( v \in \text{vars}(γ) \) arbitrary. Any variable will do.
Selection heuristics?
**Algorithms**

**compute-NES:** Compute a necessary enabling set.

**Procedure compute-NES**

```python
def compute-NES(o,s):
    select v ∈ prevars(o) with s(v) ≠ pre(o)(v)
    N ← {o' ∈ Act(s) | eff(o')(v) = pre(o)(v)}
    return N
```

Selection of v ∈ prevars(o) arbitrary. Any variable will do. Selection heuristics?
compute-interfering-operators: Compute interfering operators.

Procedure compute-interfering-operators (for strong SS)

def compute-interfering-operators(o):
    disablers ← \{ o' ∈ O | o' disables o \}
    disablees ← \{ o' ∈ O | o disables o' \}
    conflicting ← \{ o' ∈ O | o and o' conflict \}
    return disablers ∪ disablees ∪ conflicting

Procedure compute-interfering-operators (for weak SS)

def compute-interfering-operators(o):
    disablees ← \{ o' ∈ O | o disables o' \}
    conflicting ← \{ o' ∈ O | o and o' conflict \}
    return disablees ∪ conflicting
Algorithms

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

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

**Procedure compute-stubborn-set**

```python
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 $T_s$
```

January 8th, 2016 B. Nebel, R. Mattmüller – AI Planning
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 $T_{app(s)} := T_s \cap app(s)$.
Preservation of Completeness and Optimality

**Theorem**

*Weak stubborn sets are completeness and optimality preserving.*

**Proof**

Let \( T_{\text{app}}(s) := T_s \cap \text{app}(s) \) 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, \( T_{\text{app}}(s) \) contains an operator that starts such a plan.

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

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

...
Proof (ctd.)

As $T_s$ contains a disjunctive action landmark, $\pi$ must contain an operator from $T_s$.

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

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 $\pi$ to enable $o_k$, contradicting that $o_k$ was chosen with smallest index.

2. ...
Preservation of Completeness and Optimality

Proof (ctd.)

1. ...

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

Hence, we can move $o_k$ to the front:

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

It has the same cost as $\pi$ and is hence optimal.

Thus, we have found an optimal plan of length $n$ started by an operator $o_k \in T_{app(s)}$, completing the proof.
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 $o_1, \ldots, o_{k-1}$ (and hence, $o_k$ is independent of $o_1, \ldots, o_{k-1}$), which is a stronger property than needed in the proof.

**Corollary**

*Strong stubborn sets are completeness and optimality preserving.*
## Some Experiments: Overview

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

<table>
<thead>
<tr>
<th>Domain (problems)</th>
<th>Coverage Nodes generated</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A*</td>
<td>+SSS</td>
</tr>
<tr>
<td><strong>PARCPRINTER-08 (30)</strong></td>
<td>18</td>
<td>+12</td>
</tr>
<tr>
<td><strong>PARCPRINTER-OPT11 (20)</strong></td>
<td>13</td>
<td>+7</td>
</tr>
<tr>
<td><strong>WOODWORKING-OPT08 (30)</strong></td>
<td>17</td>
<td>+10</td>
</tr>
<tr>
<td><strong>WOODWORKING-OPT11 (20)</strong></td>
<td>12</td>
<td>+7</td>
</tr>
<tr>
<td><strong>SATELLITE (36)</strong></td>
<td>7</td>
<td>+5</td>
</tr>
<tr>
<td><strong>ROVERS (40)</strong></td>
<td>7</td>
<td>+2</td>
</tr>
<tr>
<td><strong>AIRPORT (50)</strong></td>
<td>28</td>
<td>±0</td>
</tr>
<tr>
<td><strong>OPENSTACKS-OPT08 (30)</strong></td>
<td>19</td>
<td>+2</td>
</tr>
<tr>
<td><strong>OPENSTACKS-OPT11 (20)</strong></td>
<td>14</td>
<td>+2</td>
</tr>
<tr>
<td><strong>DRIVERLOG (20)</strong></td>
<td>13</td>
<td>+1</td>
</tr>
<tr>
<td><strong>SCANALYZER-08 (30)</strong></td>
<td>15</td>
<td>−3</td>
</tr>
<tr>
<td><strong>SCANALYZER-OPT11 (20)</strong></td>
<td>12</td>
<td>−3</td>
</tr>
<tr>
<td><strong>PARKING-OPT11 (20)</strong></td>
<td>3</td>
<td>−1</td>
</tr>
<tr>
<td><strong>SOKOBAN-OPT08 (30)</strong></td>
<td>30</td>
<td>−1</td>
</tr>
<tr>
<td><strong>VISITALL-OPT11 (20)</strong></td>
<td>11</td>
<td>−1</td>
</tr>
<tr>
<td><strong>REMAINING DOMAINS (980)</strong></td>
<td>544</td>
<td>±0</td>
</tr>
<tr>
<td><strong>SUM (1396)</strong></td>
<td>763</td>
<td>+39</td>
</tr>
</tbody>
</table>
Some Experiments
Weak compared to strong stubborn sets

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

<table>
<thead>
<tr>
<th>Domain (problems)</th>
<th>Coverage</th>
<th>Nodes generated</th>
<th># problems w. diff. gen.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SSS</td>
<td>SSS</td>
<td></td>
</tr>
<tr>
<td>OPENSTACKS-OPT08 (30)</td>
<td>21 ± 0</td>
<td>152711917</td>
<td>99.936%</td>
</tr>
<tr>
<td>OPENSTACKS-OPT11 (20)</td>
<td>16 ± 0</td>
<td>152642101</td>
<td>99.936%</td>
</tr>
<tr>
<td>PATHWAYS-NONEG (30)</td>
<td>5 ± 0</td>
<td>162347</td>
<td>99.702%</td>
</tr>
<tr>
<td>PSR-SMALL (50)</td>
<td>49 ± 0</td>
<td>18119489</td>
<td>99.998%</td>
</tr>
<tr>
<td>SATELLITE (36)</td>
<td>12 ± 0</td>
<td>70299721</td>
<td>92.804%</td>
</tr>
</tbody>
</table>

⇒ In practice (or, at least, in the standard benchmark problems) there is no significant difference between weak and strong stubborn sets.
4 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.