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**
2 Preliminaries

- Setting
- Basic Definitions
- Operator Dependencies
- Active Operators
- Necessary Enabling Sets and Disjunctive Action Landmarks

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 $\text{DTG}(v) = (\mathcal{D}, E)$ where $(d, d') \in E$ if there is an operator $o \in O$ with
- $\text{eff}(o)(v) = d'$, and
- $v \notin \text{prevars}(o)$ or $\text{pre}(o)(v) = d$.
Then:

- \( \text{go-to-uni} \) and \( \text{go-to-gym} \) disable \( \text{wear-left} \) and \( \text{wear-right} \).
- \( \text{wear-left} \) and \( \text{wear-right} \) enable \( \text{go-to-uni} \) and \( \text{go-to-gym} \).
- \( \text{go-to-uni} \) and \( \text{go-to-gym} \) conflict.
- \( \text{wear-left} \) and \( \text{wear-right} \) are commutative.

Definition (Operator dependencies)

Let \( \Pi = (V, O, I, \gamma) \) 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 \( o \).
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 \( v \).
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.

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 \( \langle V, I, O, \text{pre}(o) \rangle \) 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 \( \text{pre}(o) \) 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

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 \( T_s \).

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 \( T_s \).
**Strong Stubborn Sets**

**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 $\Pi = (V, I, O, \gamma)$ be a planning task with the following components:

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

**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 $\Pi = (V, I, O, \gamma)$ 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 $s$.

**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.
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.

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

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

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?

compute-interfering-operators: Compute interfering operators.

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

Precedure compute-interfering-operators (for weak SS)
def compute-interfering-operators($o$):
    disablees $\leftarrow \{o' \in O \mid o \text{ disables } o'\}$
    conflicting $\leftarrow \{o' \in O \mid o \text{ and } o' \text{ conflict}\}$
    return disablees $\cup$ 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):
    $T_s$ ← $compute-DAL(\gamma)$
    while no fixed-point of $T_s$ reached do
        for $o \in T_s$ applicable in $s$
            $T_s$ ← $T_s \cup compute-interfering-operators(o)$
        for $o \in T_s$ not applicable in $s$
            $T_s$ ← $T_s \cup compute-NES(o, s)$
    end while
    return $T_s$
```

Integration into A*

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 $T_{app(s)} := T_s \cap app(s)$ 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 $\pi = o_1, \ldots, o_n$ 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. $o_k \in app(s)$: 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. ...
Thus, we have found an optimal plan of length $n$. It has the same cost as $\Pi$, 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 $o_1, \ldots, o_{k-1}$ 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 $o_k \in T_{\text{app}(s)}$, completing the proof.

\[\square\]
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.