Optimally Relaxing Partial-Order Plans with MaxSAT

Christian Muise (cjmuise@cs.toronto.edu)
Sheila A. McIlraith (sheila@cs.toronto.edu)
J. Christopher Beck (jcb@mie.utoronto.ca)

University of Toronto

June 27, 2012

Motivation

- Partial-Order Plans (POPs) have an appealing *least commitment* nature.
- POP planners are not as effective as sequential ones.
- MaxSAT solvers have become increasingly powerful.

Motivation

- Partial-Order Plans (POPs) have an appealing *least commitment* nature.
- POP planners are not as effective as sequential ones.
- MaxSAT solvers have become increasingly powerful.

Goal

Can we use a sequential planner to generate a plan and then use a SAT solver to turn that plan into an "optimal" POP?

Approach

1. Generate a sequential plan (FF).

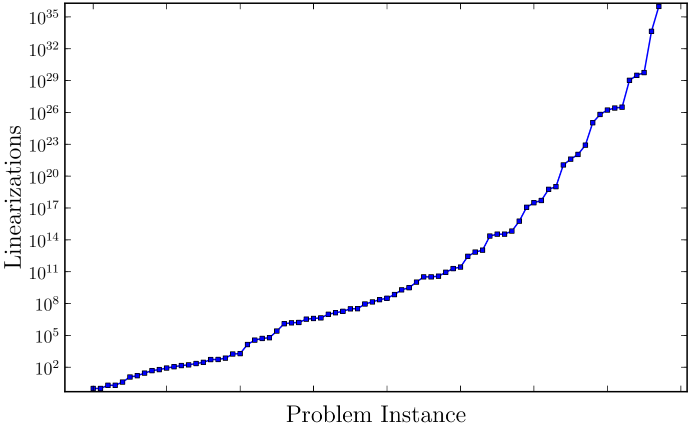
Approach

1. Generate a sequential plan (FF).
2. Encode the problem of finding a POP from the plan.

Approach

1. Generate a sequential plan (FF).
2. Encode the problem of finding a POP from the plan.
3. Use a MaxSAT solver to compute a POP (Sat4j).

The Result



10^{35}
10^{32}
10^{29}
10^{26}
10^{23}
10^{20}
10^{17}
10^{14}
10^{11}
10^{8}
10^{5}
10^{2}
Linearizations
Problem Instance

Outline

1. Background
2. Least Commitment Criteria
3. Encoding
4. Empirical Evaluation
5. Conclusion

Outline

Outline

1. Background
   - Propositional Planning
   - Partial Order Plans
   - Partial Weighted MAXSAT

Propositional Planning

Planning Problem

- STRIPS Planning problem $\Pi = \langle F, O, I, G \rangle$
- F: Finite set of fluents
- O: Finite set of operators
- I: Initial state ($I \subseteq F$)
- G: Goal state ($G \subseteq F$)

Propositional Planning

Planning Problem

- STRIPS Planning problem $\Pi = \langle F, O, I, G \rangle$
- F: Finite set of fluents
- O: Finite set of operators
- I: Initial state ($I \subseteq F$)
- G: Goal state ($G \subseteq F$)

State

A state $s \subseteq F$ is a subset of the fluents that currently hold. In a *complete state*, fluents not in s are presumed to be false. A *partial state* does not have this assumption.

Operators and Actions

For each $o \in O$

- $PRE(o) \subseteq F$: Precondition
- $ADD(o) \subseteq F$: Add effects
- $DEL(o) \subseteq F$: Delete effects

Operators and Actions

For each $o \in O$

- $PRE(o) \subseteq F$: Precondition
- $ADD(o) \subseteq F$: Add effects
- $DEL(o) \subseteq F$: Delete effects

Action

An instance of an operator is referred to as an *action*. There may be many actions that correspond to the same operator.

Plans

Action execution

- An action a is *executable* in state s iff $PRE(a) \subset s$.
- Executing an action a executable in state s causes the state to change to $(s \setminus DEL(a)) \cup ADD(a)$.
- Execution of a sequence of actions is the process of executing each action in turn. A sequence can only be executed if each individual action is executable in the corresponding state.

Plans

Action execution

- An action a is *executable* in state s iff $PRE(a) \subset s$.
- Executing an action a executable in state s causes the state to change to $(s \setminus DEL(a)) \cup ADD(a)$.
- Execution of a sequence of actions is the process of executing each action in turn. A sequence can only be executed if each individual action is executable in the corresponding state.

Sequential Plan

A *sequential plan* is a sequence of actions $\vec{a} = [a_1, a_2, \cdots, a_n]$ that can be executed in the initial state I, and achieves the goal G.

Outline

1. Background
   - Propositional Planning
   - Partial Order Plans
   - Partial Weighted MAXSAT

POP Example

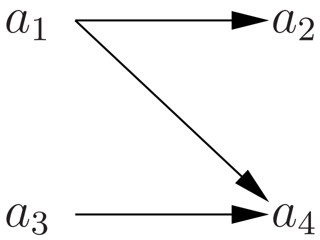
POP Example

a_1 a_2

a_3 a_4

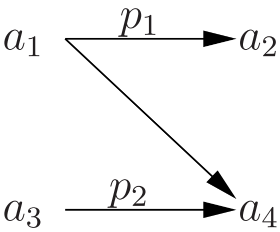
Actions

POP Example



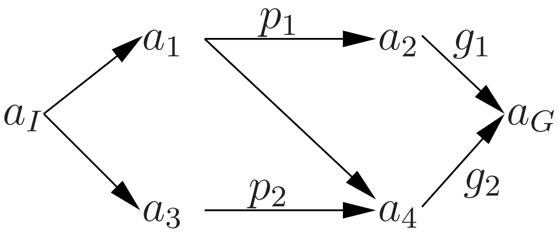
Ordering Constraints

POP Example



Causal Links

POP Example



Actions for *A* and *G*

Partial Order Plans

Partial Order Plan (POP)

- For a problem Π, a POP is a tuple $P = \langle \mathcal{A}, \mathcal{O}, \mathcal{C} \rangle$.

Partial Order Plans

Partial Order Plan (POP)

- For a problem Π, a POP is a tuple $P = \langle \mathcal{A}, \mathcal{O}, \mathcal{C} \rangle$.
- $\mathcal{A}$: Set of actions in the plan corresponding to operators in Π.

Partial Order Plans

Partial Order Plan (POP)

- For a problem Π, a POP is a tuple $P = \langle \mathcal{A}, \mathcal{O}, \mathcal{C} \rangle$.
- $\mathcal{A}$: Set of actions in the plan corresponding to operators in Π.
- $\mathcal{O}$: Set of ordering constraints between the actions in $\mathcal{A}$.
    - E.g., $(a_1 \prec a_2) \in \mathcal{O}$ (can assume $\mathcal{O}$ is transitively closed)

Partial Order Plans

Partial Order Plan (POP)

- For a problem Π, a POP is a tuple $P = \langle \mathcal{A}, \mathcal{O}, \mathcal{C} \rangle$.
- $\mathcal{A}$: Set of actions in the plan corresponding to operators in Π.
- $\mathcal{O}$: Set of ordering constraints between the actions in $\mathcal{A}$.
  - E.g., $(a_1 \prec a_2) \in \mathcal{O}$ (can assume $\mathcal{O}$ is transitively closed)
- $\mathcal{C}$: Set of causal links between the actions in $\mathcal{A}$. A causal link is an annotated ordering constraint that is labelled with a fluent that represents why the link exists.
  - E.g., $(a_1 \overset{f}{\prec} a_2) \in \mathcal{C}$ (can assume $f \in ADD(a_1) \cap PRE(a_2)$)

POP Terminology

Linearizations

A *linearization* of the POP $P = \langle \mathcal{A}, \mathcal{O}, \mathcal{C} \rangle$ is a total ordering of actions in $\mathcal{A}$ that respects the ordering constraints of $\mathcal{O}$.

POP Terminology

Linearizations

A *linearization* of the POP $P = \langle \mathcal{A}, \mathcal{O}, \mathcal{C} \rangle$ is a total ordering of actions in $\mathcal{A}$ that respects the ordering constraints of $\mathcal{O}$.

Threats & Support

- For a causal link $(a_1 \stackrel{f}{\prec} a_2)$, we say that a_1 *supports* the precondition f of a_2, and the precondition is *supported*.

POP Terminology

Linearizations

A *linearization* of the POP $P = \langle \mathcal{A}, \mathcal{O}, \mathcal{C} \rangle$ is a total ordering of actions in $\mathcal{A}$ that respects the ordering constraints of $\mathcal{O}$.

Threats & Support

- For a causal link $(a_1 \overset{f}{\prec} a_2)$, we say that a_1 *supports* the precondition f of a_2, and the precondition is *supported*.
- Any precondition $f \in PRE(a)$ for some action $a \in \mathcal{A}$ is an *open precondition* if it is not supported.

POP Terminology

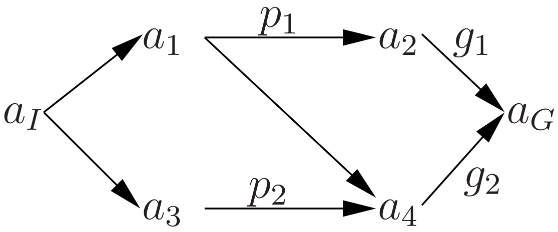
Linearizations

A *linearization* of the POP $P = \langle \mathcal{A}, \mathcal{O}, \mathcal{C} \rangle$ is a total ordering of actions in $\mathcal{A}$ that respects the ordering constraints of $\mathcal{O}$.

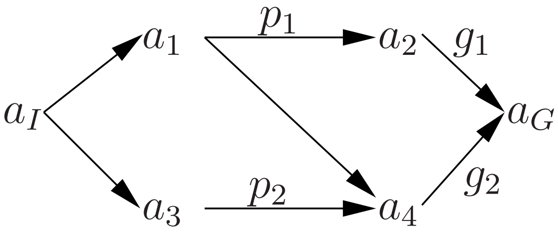
Threats & Support

- For a causal link $(a_1 \stackrel{f}{\prec} a_2)$, we say that a_1 *supports* the precondition f of a_2, and the precondition is *supported*.
- Any precondition $f \in PRE(a)$ for some action $a \in \mathcal{A}$ is an *open precondition* if it is not supported.
- A causal link $(a_1 \stackrel{f}{\prec} a_2)$ is *threatened* if there is some action a_3 such that $f \in DEL(a_3)$ and $\mathcal{O} \cap \{(a_3 \prec a_1), (a_2 \prec a_3)\} = \emptyset$.

POP Example



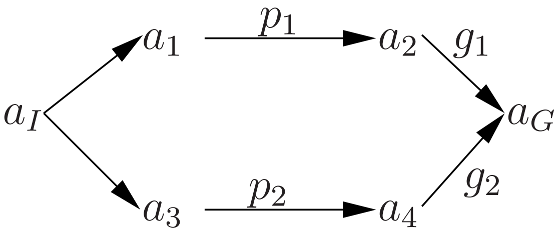
POP Example



- Linearizations:

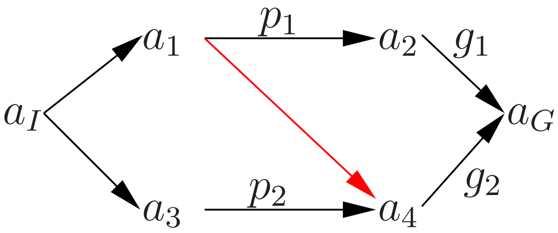
  $[a_1, a_2, a_3, a_4]$ $[a_1, a_3, a_2, a_4]$ $[a_1, a_3, a_4, a_2]$
  $[a_3, a_1, a_2, a_4]$ $[a_3, a_1, a_4, a_2]$

POP Example



Threats: $g_2 \in DEL(a_1)$

POP Example



Threats: $g_2 \in DEL(a_1)$

POP Validity

Intuition

A POP is *valid* if it achieves the goal from the initial state.

POP Validity

Intuition

A POP is *valid* if it achieves the goal from the initial state.

Linearization Validity

A POP $P = \langle \mathcal{A}, \mathcal{O}, \mathcal{C} \rangle$ is valid for Π iff every linearization of the POP is a plan for Π.

POP Validity

Intuition

A POP is *valid* if it achieves the goal from the initial state.

Linearization Validity

A POP $P = \langle \mathcal{A}, \mathcal{O}, \mathcal{C} \rangle$ is valid for Π iff every linearization of the POP is a plan for Π.

Threat & Support Validity

A POP $P = \langle \mathcal{A}, \mathcal{O}, \mathcal{C} \rangle$ is valid for Π iff the following holds:

1. There are no open preconditions in P.
2. No causal link in $\mathcal{C}$ is threatened.
3. $\mathcal{A}$ contains the dummy actions a_I and a_G.

Outline

(Maximum) Satisfiability

Satisfiability (SAT)

$$(x \vee y) \wedge (\neg x \vee z)$$

(Maximum) Satisfiability

Satisfiability (SAT)

$$(x \vee y) \wedge (\neg x \vee z)$$

(Maximum) Satisfiability

Satisfiability (SAT)

$$(x \vee y) \wedge (\neg x \vee z)$$

Maximum Satisfiability (MAXSAT)

$$(x \vee y \vee \neg z) \wedge (x \vee z) \wedge (y \vee z) \wedge$$
$$(\neg x \vee \neg y) \wedge (\neg z \vee \neg x) \wedge (\neg z \vee \neg y)$$

(Maximum) Satisfiability

Satisfiability (SAT)

$$(x \lor y) \land (\neg x \lor z)$$

Maximum Satisfiability (MAXSAT)

$$(x \lor y \lor \neg z) \land (x \lor z) \land (y \lor z) \land$$
$$(\neg x \lor \neg y) \land (\neg z \lor \neg x) \land (\neg z \lor \neg y)$$

(Partial) Weighted MAXSAT

Weighted MAXSAT

$$\overset{3}{(x)} \wedge \overset{1}{(\neg x \vee \neg y)} \wedge \overset{1}{(\neg x \vee \neg z)} \wedge \overset{1}{(y)} \wedge \overset{1}{(z)}$$

(Partial) Weighted MAXSAT

Weighted MAXSAT

$$\overset{3}{(x)} \wedge \overset{1}{(\neg x \vee \neg y)} \wedge \overset{1}{(\neg x \vee \neg z)} \wedge \overset{1}{(y)} \wedge \overset{1}{(z)}$$

(Partial) Weighted MAXSAT

Weighted MAXSAT

$$\overset{3}{(x)} \wedge \overset{1}{(\neg x \vee \neg y)} \wedge \overset{1}{(\neg x \vee \neg z)} \wedge \overset{1}{(y)} \wedge \overset{1}{(z)}$$

(Partial) Weighted MAXSAT

Weighted MAXSAT

$$\overset{3}{(x)} \wedge \overset{1}{(\neg x \vee \neg y)} \wedge \overset{1}{(\neg x \vee \neg z)} \wedge \overset{1}{(y)} \wedge \overset{1}{(z)}$$

Partial Weighted MAXSAT

$$\overset{1}{(x)} \wedge \overset{2}{(y)} \wedge \overset{3}{(z)} \wedge \overset{\bullet}{(\neg x \vee \neg z)} \wedge \overset{\bullet}{(\neg y \vee \neg z)} \wedge \overset{\bullet}{(\neg x \vee \neg y)}$$

(Partial) Weighted MAXSAT

Weighted MAXSAT

$$\overset{3}{(x)} \wedge (\neg x \overset{1}{\vee} \neg y) \wedge (\neg x \overset{1}{\vee} \neg z) \wedge \overset{1}{(y)} \wedge \overset{1}{(z)}$$

Partial Weighted MAXSAT

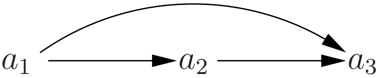
$$\overset{1}{(x)} \wedge \overset{2}{(y)} \wedge \overset{3}{(z)} \wedge (\neg x \overset{\bullet}{\vee} \neg z) \wedge (\neg y \overset{\bullet}{\vee} \neg z) \wedge (\neg x \overset{\bullet}{\vee} \neg y)$$

Outline

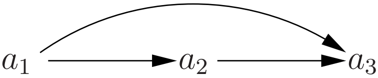
Outline

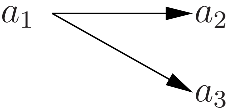
2 Least Commitment Criteria
- Deordering & Reordering
- Least Commitment POP

Deordering & Reordering



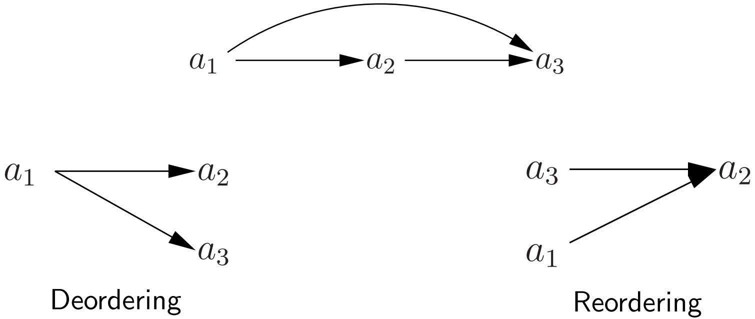
Deordering & Reordering



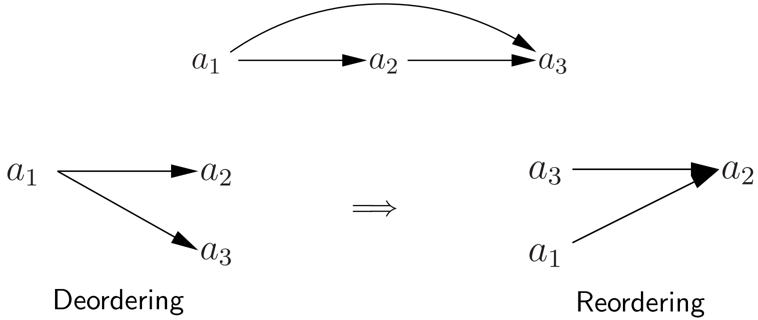


Deordering

Deordering & Reordering



Deordering & Reordering



Outline

2. Least Commitment Criteria
   - Deordering & Reordering
   - Least Commitment POP

Least Commitment POP

- Would like a notion that includes the number of actions.

Least Commitment POP

- Would like a notion that includes the number of actions.
- Prefer to first minimize the number of actions.

Least Commitment POP

- Would like a notion that includes the number of actions.
- Prefer to first minimize the number of actions.
- When actions are minimal, minimize the number of orderings.

Least Commitment POP

- Would like a notion that includes the number of actions.
- Prefer to first minimize the number of actions.
- When actions are minimal, minimize the number of orderings.

Least Commitment POP (LCP)

Let $P = \langle \mathcal{A}, \mathcal{O} \rangle$ and $Q = \langle \mathcal{A}^{'}, \mathcal{O}^{'} \rangle$ be two POPs valid for Π. Q is a *least commitment POP* (LCP) of P iff Q is the minimum reordering of itself and there is no valid POP $\langle \mathcal{A}^{''}, \mathcal{O}^{''} \rangle$ for Π such that $\mathcal{A}'' \subseteq \mathcal{A}$ and $|\mathcal{A}''| < |\mathcal{A}'|$.

Outline

Outline

3. Encoding
   - Core Encoding
   - Extensions
   - Approach

Core Encoding

Action Variables and Ordering Variables

- $\forall a \in \vec{a}, \quad x_a$: True iff a is in the final POP.
- $\forall a_i, a_j \in \vec{a}, \quad \kappa(a_i, a_j)$: True iff $(a_i \prec a_j)$ is in the final POP.
- $\Upsilon(a_i, a_j, p)$: a_i supports a_j with p.

Basic Clauses

- No self loops.
- Include a_I and a_G.
- If an ordering is used, include the actions.
- If we include an action, order it after (before) a_I (a_G).
- Enforce the transitive closure.

Core Encoding

Action Variables and Ordering Variables

- $\forall a \in \vec{a}, \quad x_a$: True iff a is in the final POP.
- $\forall a_i, a_j \in \vec{a}, \quad \kappa(a_i, a_j)$: True iff $(a_i \prec a_j)$ is in the final POP.
- $\Upsilon(a_i, a_j, p)$: a_i supports a_j with p.

Basic Clauses

- No self loops.
- Include a_I and a_G.
- If an ordering is used, include the actions.
- If we include an action, order it after (before) a_I (a_G).
- Enforce the transitive closure.

Core Encoding

Action Variables and Ordering Variables

- $\forall a \in \vec{a}, \quad x_a$: True iff a is in the final POP.
- $\forall a_i, a_j \in \vec{a}, \quad \kappa(a_i, a_j)$: True iff $(a_i \prec a_j)$ is in the final POP.
- $\Upsilon(a_i, a_j, p)$: a_i supports a_j with p.

Basic Clauses

- $\neg\kappa(a, a)$ No self loops.
- Include a_I and a_G.
- If an ordering is used, include the actions.
- If we include an action, order it after (before) a_I (a_G).
- Enforce the transitive closure.

Core Encoding

Action Variables and Ordering Variables

- $\forall a \in \vec{a}, \quad x_a$: True iff a is in the final POP.
- $\forall a_i, a_j \in \vec{a}, \quad \kappa(a_i, a_j)$: True iff $(a_i \prec a_j)$ is in the final POP.
- $\Upsilon(a_i, a_j, p)$: a_i supports a_j with p.

Basic Clauses

- $\neg\kappa(a, a)$ No self loops.
- Include a_I and a_G.
- If an ordering is used, include the actions.
- If we include an action, order it after (before) a_I (a_G).
- Enforce the transitive closure.

Core Encoding

Action Variables and Ordering Variables

- $\forall a \in \vec{a}, \quad x_a$: True iff a is in the final POP.
- $\forall a_i, a_j \in \vec{a}, \quad \kappa(a_i, a_j)$: True iff $(a_i \prec a_j)$ is in the final POP.
- $\Upsilon(a_i, a_j, p)$: a_i supports a_j with p.

Basic Clauses

- $\neg\kappa(a, a)$ — No self loops.
- $(x_{a_I}) \wedge (x_{a_G})$ — Include a_I and a_G.
- If an ordering is used, include the actions.
- If we include an action, order it after (before) a_I (a_G).
- Enforce the transitive closure.

Core Encoding

Action Variables and Ordering Variables

- $\forall a \in \vec{a}, \quad x_a$: True iff a is in the final POP.
- $\forall a_i, a_j \in \vec{a}, \quad \kappa(a_i, a_j)$: True iff $(a_i \prec a_j)$ is in the final POP.
- $\Upsilon(a_i, a_j, p)$: a_i supports a_j with p.

Basic Clauses

- $\neg\kappa(a, a)$ No self loops.
- $(x_{a_I}) \wedge (x_{a_G})$ Include a_I and a_G.
- If an ordering is used, include the actions.
- If we include an action, order it after (before) a_I (a_G).
- Enforce the transitive closure.

Core Encoding

Action Variables and Ordering Variables

- $\forall a \in \vec{a}, \quad x_a$: True iff a is in the final POP.
- $\forall a_i, a_j \in \vec{a}, \quad \kappa(a_i, a_j)$: True iff $(a_i \prec a_j)$ is in the final POP.
- $\Upsilon(a_i, a_j, p)$: a_i supports a_j with p.

Basic Clauses

- $\neg\kappa(a, a)$ — No self loops.
- $(x_{a_I}) \wedge (x_{a_G})$ — Include a_I and a_G.
- $\kappa(a_i, a_j) \rightarrow x_{a_i} \wedge x_{a_j}$ — Ordering implies actions.
- If we include an action, order it after (before) a_I (a_G).
- Enforce the transitive closure.

Core Encoding

Action Variables and Ordering Variables

- $\forall a \in \vec{a}, \quad x_a$: True iff a is in the final POP.
- $\forall a_i, a_j \in \vec{a}, \quad \kappa(a_i, a_j)$: True iff $(a_i \prec a_j)$ is in the final POP.
- $\Upsilon(a_i, a_j, p)$: a_i supports a_j with p.

Basic Clauses

- $\neg\kappa(a, a)$ — No self loops.
- $(x_{a_I}) \wedge (x_{a_G})$ — Include a_I and a_G.
- $\kappa(a_i, a_j) \rightarrow x_{a_i} \wedge x_{a_j}$ — Ordering implies actions.
- If we include an action, order it after (before) a_I (a_G).
- Enforce the transitive closure.

Core Encoding

Action Variables and Ordering Variables

- $\forall a \in \vec{a}, \quad x_a$: True iff a is in the final POP.
- $\forall a_i, a_j \in \vec{a}, \quad \kappa(a_i, a_j)$: True iff $(a_i \prec a_j)$ is in the final POP.
- $\Upsilon(a_i, a_j, p)$: a_i supports a_j with p.

Basic Clauses

- $\neg\kappa(a, a)$ — No self loops.
- $(x_{a_I}) \wedge (x_{a_G})$ — Include a_I and a_G.
- $\kappa(a_i, a_j) \rightarrow x_{a_i} \wedge x_{a_j}$ — Ordering implies actions.
- $x_{a_i} \rightarrow \kappa(a_I, a_i) \wedge \kappa(a_i, a_G)$ — Order actions with a_I and a_G.
- Enforce the transitive closure.

Core Encoding

Action Variables and Ordering Variables

- $\forall a \in \vec{a}, \quad x_a$: True iff a is in the final POP.
- $\forall a_i, a_j \in \vec{a}, \quad \kappa(a_i, a_j)$: True iff $(a_i \prec a_j)$ is in the final POP.
- $\Upsilon(a_i, a_j, p)$: a_i supports a_j with p.

Basic Clauses

- $\neg\kappa(a, a)$ — No self loops.
- $(x_{a_I}) \wedge (x_{a_G})$ — Include a_I and a_G.
- $\kappa(a_i, a_j) \rightarrow x_{a_i} \wedge x_{a_j}$ — Ordering implies actions.
- $x_{a_i} \rightarrow \kappa(a_I, a_i) \wedge \kappa(a_i, a_G)$ — Order actions with a_I and a_G.
- Enforce the transitive closure.

Core Encoding

Action Variables and Ordering Variables

- $\forall a \in \vec{a}, \quad x_a$: True iff a is in the final POP.
- $\forall a_i, a_j \in \vec{a}, \quad \kappa(a_i, a_j)$: True iff $(a_i \prec a_j)$ is in the final POP.
- $\Upsilon(a_i, a_j, p)$: a_i supports a_j with p.

Basic Clauses

- $\neg\kappa(a, a)$ — No self loops.
- $(x_{a_I}) \wedge (x_{a_G})$ — Include a_I and a_G.
- $\kappa(a_i, a_j) \rightarrow x_{a_i} \wedge x_{a_j}$ — Ordering implies actions.
- $x_{a_i} \rightarrow \kappa(a_I, a_i) \wedge \kappa(a_i, a_G)$ — Order actions with a_I and a_G.
- $\kappa(a_i, a_j) \wedge \kappa(a_j, a_k) \rightarrow \kappa(a_i, a_k)$ — Transitive closure.

Core Encoding Cont.

POP Viability Clauses

- Ensure that if we include action a_j, then every precondition p of a_j must be satisfied by at least one achiever a_i.
- Ensure that if a_i achieves precondition p for action a_j, then no deleter of p will be allowed to occur between a_i and a_j.

Core Encoding Cont.

POP Viability Clauses

- Ensure that if we include action a_j, then every precondition p of a_j must be satisfied by at least one achiever a_i.

$$x_{a_j} \rightarrow \bigwedge_{p \in PRE(a_j)} \bigvee_{a_i \in \textbf{adders}(p)} \left[\, \kappa(a_i, a_j) \wedge \Upsilon(a_i, a_j, p) \, \right]$$

- Ensure that if a_i achieves precondition p for action a_j, then no deleter of p will be allowed to occur between a_i and a_j.

Core Encoding Cont.

POP Viability Clauses

- Ensure that if we include action a_j, then every precondition p of a_j must be satisfied by at least one achiever a_i.

$$x_{a_j} \rightarrow \bigwedge_{p \in PRE(a_j)} \bigvee_{a_i \in \textbf{adders}(p)} \left[\, \kappa(a_i, a_j) \wedge \Upsilon(a_i, a_j, p) \, \right]$$

- Ensure that if a_i achieves precondition p for action a_j, then no deleter of p will be allowed to occur between a_i and a_j.

$$\Upsilon(a_i, a_j, p) \rightarrow \left[\, \bigwedge_{a_k \in \textbf{deleters}(p)} x_{a_k} \rightarrow \kappa(a_k, a_i) \vee \kappa(a_j, a_k) \, \right]$$

Core Encoding Cont.

POP Viability Clauses

- Ensure that if we include action a_j, then every precondition p of a_j must be satisfied by at least one achiever a_i.

$$x_{a_j} \rightarrow \bigwedge_{p \in PRE(a_j)} \bigvee_{a_i \in \textbf{adders}(p)} \left[\kappa(a_i, a_j) \wedge \Upsilon(a_i, a_j, p) \right]$$

- Ensure that if a_i achieves precondition p for action a_j, then no deleter of p will be allowed to occur between a_i and a_j.

$$\Upsilon(a_i, a_j, p) \rightarrow \left[\bigwedge_{a_k \in \textbf{deleters}(p)} x_{a_k} \rightarrow \kappa(a_k, a_i) \vee \kappa(a_j, a_k) \right]$$

Soft Clauses

- $w(\neg\kappa(a_i, a_j)) = 1, \ \forall a_i, a_j \in \mathcal{A}$
- $w(\neg x_a) = 1 + |\mathcal{A}|^2, \ \forall a \in \mathcal{A} \setminus \{a_I, a_G\}$

Outline

3. Encoding
   - Core Encoding
   - Extensions
   - Approach

Extensions

All Actions (AA)

$$(x_a), \quad \forall a \in \mathcal{A}$$

Deordering (DO)

$$(\neg\kappa(a_j, a_i)), \qquad [a_1, \ldots, a_i, \ldots, a_j, \ldots, a_n]$$

Variants

- AA,DO: Minimum Deordering (MD)
- AA,$\neg$DO: Minimum Reordering (MR)
- $\neg$AA,$\neg$DO: Least Commitment POP (LCP)

Outline

3 Encoding
- Core Encoding
- Extensions
- Approach

Approach

1. Generate a sequential plan (FF).
2. Encode the problem of finding a POP (MD, MR, or LCP).
3. Use a MAXSAT solver to compute the POP (Sat4j).

Outline

Outline

4. Empirical Evaluation
   - Relaxer Algorithm
   - Encoding Difficulty
   - POP Quality
   - Reordering Flexibility

Relaxer Algorithm (KK)

Introduced by Kambhampati and Kedar (1994), the algorithm computes a deordering of a plan by removing redundant edges.

Algorithm 1: Relaxer Algorithm

Input: Sequential plan, $\vec{a}$, including a_I and a_G
Output: Partial-order plan, $\langle \mathcal{A}, \mathcal{O}, \mathcal{C} \rangle$

foreach $a_j \in \mathcal{A}$ **do**
  foreach $f \in PRE(a_j)$ **do**
    Let a_i be the first action in $\vec{a}$ such that $i < j$, $f \in ADD(a_i)$, and $\forall k, i < k < j \Rightarrow f \notin DEL(a_k)$.
    Create a causal link between a_i and a_j.
    Add necessary ordering constraints so f isn't threatened.

Outline

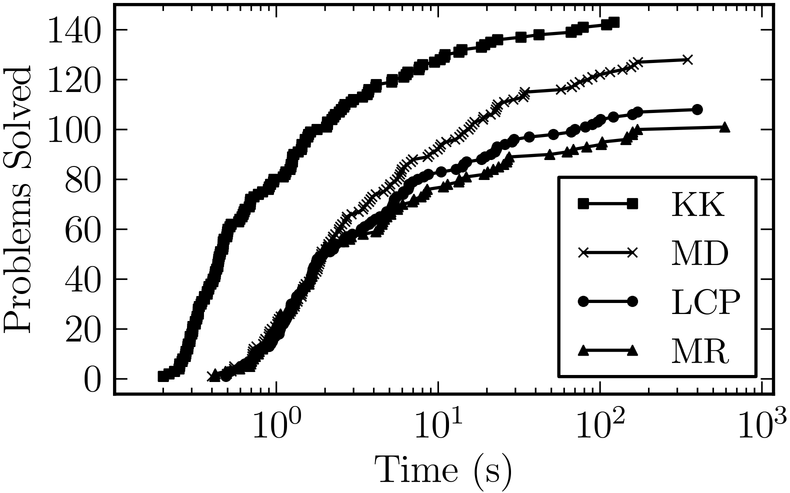
4. Empirical Evaluation
   - Relaxer Algorithm
   - Encoding Difficulty
   - POP Quality
   - Reordering Flexibility

Successfully Encoded

Domain	Num Probs	FF Solved	Successfully Encoded
Depots	22	22	22
Driverlog	20	16	16
Logistics	35	35	33
TPP	30	30	20
Rovers	20	20	20
Zeno	20	20	18
ALL	147	143	129

- Time / memory limit of 30min / 2GB.
- Encoding failure due to CNF conversion.

Solve Time


Problems Solved
0
20
40
60
80
100
120
140
Time (s)
10^0
10^1
10^2
10^3
KK
MD
LCP
MR

Outline

4. Empirical Evaluation
   - Relaxer Algorithm
   - Encoding Difficulty
   - POP Quality
   - Reordering Flexibility

POP Quality

Domain	# Actions KK	# Actions LCP	# Ordering Constraints RX	# Ordering Constraints MD	# Ordering Constraints MR	# Ordering Constraints LCP
Depots (14)	34.9	31.0	473.4	473.4	430.9	341.5
Driverlog (15)	27.5	26.5	332.6	332.6	326.9	297.3
Logistics (30)	78.1	77.4	1490.6	1490.6	1462.5	1470.4
TPP (5)	13.4	13.4	74.8	74.8	74.8	74.8
Rovers (18)	31.1	30.3	223.2	223.2	217.6	204.2
Zeno (16)	29.2	29.2	404.3	404.3	403.5	403.5
ALL (98)	44.3	43.2	685.7	685.7	669.0	651.6

Mean number of actions and ordering constraints. Number of actions for KK, MD, and MR are all equal.

POP Quality

Domain	# Actions		# Ordering Constraints			
	KK	LCP	RX	MD	MR	LCP
Depots (14)	34.9	31.0	473.4	473.4	430.9	341.5
Driverlog (15)	27.5	26.5	332.6	332.6	326.9	297.3
Logistics (30)	78.1	77.4	1490.6	1490.6	1462.5	1470.4
TPP (5)	13.4	13.4	74.8	74.8	74.8	74.8
Rovers (18)	31.1	30.3	223.2	223.2	217.6	204.2
Zeno (16)	29.2	29.2	404.3	404.3	403.5	403.5
ALL (98)	44.3	43.2	685.7	685.7	669.0	651.6

Four of the domains had problems with a reduction of actions.

POP Quality

Domain	# Actions		# Ordering Constraints			
	KK	LCP	RX	MD	MR	LCP
Depots (14)	34.9	31.0	473.4	473.4	430.9	341.5
Driverlog (15)	27.5	26.5	332.6	332.6	326.9	297.3
Logistics (30)	78.1	77.4	1490.6	1490.6	1462.5	1470.4
TPP (5)	13.4	13.4	74.8	74.8	74.8	74.8
Rovers (18)	31.1	30.3	223.2	223.2	217.6	204.2
Zeno (16)	29.2	29.2	404.3	404.3	403.5	403.5
ALL (98)	44.3	43.2	685.7	685.7	669.0	651.6

The Relaxer algorithm *always* computed the minimum deordering.

POP Quality

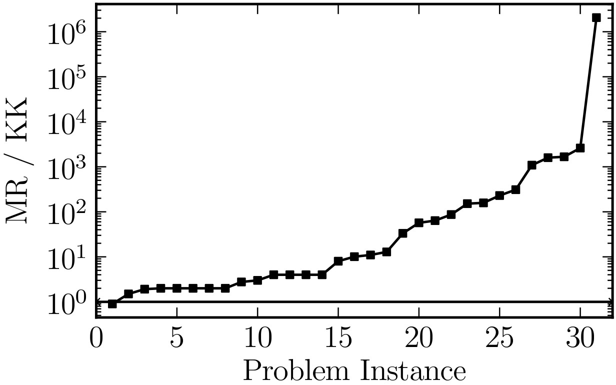
Domain	# Actions KK	# Actions LCP	# Ordering Constraints RX	# Ordering Constraints MD	# Ordering Constraints MR	# Ordering Constraints LCP
Depots (14)	34.9	31.0	473.4	473.4	430.9	341.5
Driverlog (15)	27.5	26.5	332.6	332.6	326.9	297.3
Logistics (30)	78.1	77.4	1490.6	1490.6	1462.5	1470.4
TPP (5)	13.4	13.4	74.8	74.8	74.8	74.8
Rovers (18)	31.1	30.3	223.2	223.2	217.6	204.2
Zeno (16)	29.2	29.2	404.3	404.3	403.5	403.5
ALL (98)	44.3	43.2	685.7	685.7	669.0	651.6

Fewer actions may require an increase in ordering constraints.

Outline

4. Empirical Evaluation
   - Relaxer Algorithm
   - Encoding Difficulty
   - POP Quality
   - Reordering Flexibility

Reordering Flexibility



MR (resp. KK): The number of linearizations of the POP for the minimum reordering (resp. the POP generated by Relaxer).

Outline

Conclusion

- Introduced a practical method for computing the optimal deording and reordering of a plan.

Conclusion

- Introduced a practical method for computing the optimal deording and reordering of a plan.
- Proposed an extension to least commitment planning that includes the number of actions in a solution.

Conclusion

- Introduced a practical method for computing the optimal deording and reordering of a plan.
- Proposed an extension to least commitment planning that includes the number of actions in a solution.
- Discovered that the Relaxer algorithm is extremely efficient at computing optimal deorderings.

Conclusion

- Introduced a practical method for computing the optimal deording and reordering of a plan.
- Proposed an extension to least commitment planning that includes the number of actions in a solution.
- Discovered that the Relaxer algorithm is extremely efficient at computing optimal deorderings.
- Found that greater flexibility can be achieved when using reorderings or the introduced least commitment criterion.

Future Work

- Try other forms of optimization techniques (MIP, CSP, etc.).

Future Work

- Try other forms of optimization techniques (MIP, CSP, etc.).
- Use external reasoning for handling the transitive closure.

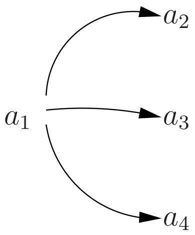
Future Work

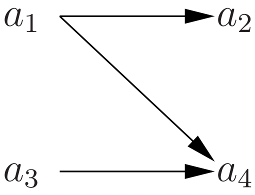
- Try other forms of optimization techniques (MIP, CSP, etc.).
- Use external reasoning for handling the transitive closure.
- Incorporate preferences into the optimization function.

Thanks

http://www.haz.ca/research/popgen/

Linearization Corner Case





Boolean Satisfiability

- Boolean variables $x_1, x_2, \cdots$ that can be either True or False.
- Unary operator $\neg$, and binary operators $\vee$ and $\wedge$.
- Well formed formula built by using variables, $\neg$, $\vee$, and $\wedge$.
- Typically given in Conjunctive Normal Form (CNF).

Boolean Satisfiability

- Boolean variables $x_1, x_2, \cdots$ that can be either True or False.
- Unary operator $\neg$, and binary operators $\vee$ and $\wedge$.
- Well formed formula built by using variables, $\neg$, $\vee$, and $\wedge$.
- Typically given in Conjunctive Normal Form (CNF).

SAT Problem

Given a well formed formula, find a True / False setting to the variables such that the formula evaluates to True.

Partial Weighted MAXSAT

MAXSAT

Given a CNF, find an assignment that satisfies as many of the clauses as possible.

Partial Weighted MAXSAT

MAXSAT

Given a CNF, find an assignment that satisfies as many of the clauses as possible.

Weighted MAXSAT

Given a CNF with weights on the clauses, find an assignment that maximizes the sum of the weights on the satisfied clauses.

Partial Weighted MAXSAT

MAXSAT

Given a CNF, find an assignment that satisfies as many of the clauses as possible.

Weighted MAXSAT

Given a CNF with weights on the clauses, find an assignment that maximizes the sum of the weights on the satisfied clauses.

Partial Weighted MAXSAT

Given a CNF with weights on *soft* clauses, find an assignment that satisfy all of the *hard* clauses and maximizes the sum of the weights on the satisfied clauses.

Deordering

Deordering

Let $P = \langle \mathcal{A}, \mathcal{O} \rangle$ and $Q = \langle \mathcal{A}', \mathcal{O}' \rangle$ be two POPs, and Π a planning problem. Q is a deordering of P wrt. Π iff P and Q are valid POPs for Π, $\mathcal{A} = \mathcal{A}'$, and $\mathcal{O}' \subseteq \mathcal{O}$.

Optimal Deordering

Let $P = \langle \mathcal{A}, \mathcal{O} \rangle$ and $Q = \langle \mathcal{A}', \mathcal{O}' \rangle$ be two POPs, and Π a planning problem. Q is a minimum deordering of P wrt. Π iff

1. Q is a deordering of P wrt. Π, and
2. There is no deordering $\langle \mathcal{A}'', \mathcal{O}'' \rangle$ of P wrt. Π s.t. $|\mathcal{O}''| < |\mathcal{O}'|$

Reordering

Reordering

Let $P = \langle \mathcal{A}, \mathcal{O} \rangle$ and $Q = \langle \mathcal{A}^{'}, \mathcal{O}^{'} \rangle$ be two POPs, and Π a planning problem. Q is a reordering of P wrt. Π iff P and Q are valid POPs for Π, and $\mathcal{A} = \mathcal{A}'$.

Optimal Reordering

Let $P = \langle \mathcal{A}, \mathcal{O} \rangle$ and $Q = \langle \mathcal{A}^{'}, \mathcal{O}^{'} \rangle$ be two POPs, and Π a planning problem. Q is a minimum reordering of P wrt. Π iff

1. Q is a reordering of P wrt. Π, and
2. There is no reordering $\langle \mathcal{A}^{''}, \mathcal{O}^{''} \rangle$ of P wrt. Π s.t. $|\mathcal{O}''| < |\mathcal{O}'|$