Change the World – How Hard Can That Be?
On the Complexity of Fixing Planning Models

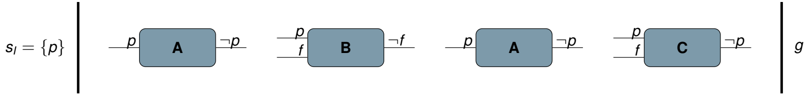
Songtuan Lin, Pascal Bercher

The Australian National University

May 20, 2021

Introduction

Motivation



An infeasible plan in which the action A deletes the fact p that is required by the actions A, B and C, and the environment does not have the fact f that is required by B and C as well.

- Counter-factual Explanations. (How to make the plan executable?)

	A	**B**	**C**
p	delete $\neg p$	N/A	N/A
	N/A	delete p	delete p
f	add f	N/A	N/A
	N/A	delete f	delete f

- Modeling assistance.

Scenario

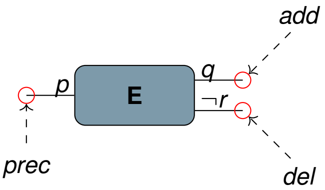
Given a plan that is supposed to be a solution to a planning problem, but it is actually not, we want to change the planning model so that it can be.

- Considering the problem in the context of hierarchical (HTN) & non-hierarchical planning.
- Complexity Study.

Changing Planning Models in Classical Planning

Basic Terminologies

- A state s is a set of proposition variables
- An action is a tuple $(prec, add, del)$
  - Preconditions $prec$: $prec \subseteq s$.
  - Effects add & del: $(s \setminus del) \cup add$.



An example of the action $\mathbf{E} = (\underbrace{\{p\}}_{prec}, \underbrace{\{q\}}_{add}, \underbrace{\{r\}}_{del})$.

Allowed Changes

Given an action sequence, we want to change actions' preconditions and effects to make the sequence executable.

Allowed Changes

Given an action sequence, we want to change actions' preconditions and effects to make the sequence executable.

- FIX-PREC: Removing a variable from an action's precondition.

Allowed Changes

Given an action sequence, we want to change actions' preconditions and effects to make the sequence executable.

- FIX-PREC: Removing a variable from an action's precondition.
- FIX-ADD: Adding a variable to an action's add list.

Allowed Changes

Given an action sequence, we want to change actions' preconditions and effects to make the sequence executable.

- FIX-PREC: Removing a variable from an action's precondition.
- FIX-ADD: Adding a variable to an action's add list.
- FIX-DEL: Removing a variable from an action's delete list.

Problem Definition: Definition

Definition (Fix-Action^k_X, $X \subseteq \{\textsc{Prec}, \textsc{Add}, \textsc{Del}\}$ and $|X| \geq 1$)

Given an action sequence $\overline{a}$, is there a way to make $\overline{a}$ executable by using the respective changes according to the value of X at most k times.

- If $\textsc{Prec} \in X$, Fix-Prec is allowed.
- If $\textsc{Add} \in X$, Fix-Add is allowed.
- If $\textsc{Del} \in X$, Fix-Del is allowed.

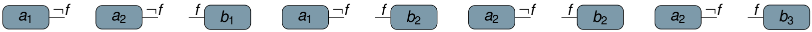
Results: FIX-PREC & FIX-DEL

Theorem

FIX-ACTION$_{\text{PREC,DEL}}$ *is in P.*

Proof.

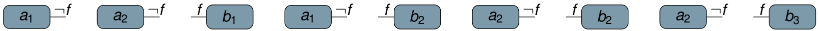
Reducing to the vertex cover problem with a bipartite graph as the input. □



An example of input action sequence in which $a_1 = a_2 = (\emptyset, \emptyset, \{f\})$ and $b_1 = b_2 = b_3 = (\{f\}, \emptyset, \emptyset)$.

*More complicated cases where $prec \cap add \cap del \neq \emptyset$ for some actions are considered in the proof presented in our paper.

Results: Reducing to the Vertex Covering Problem

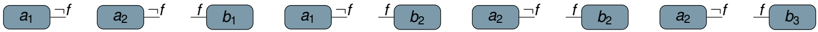


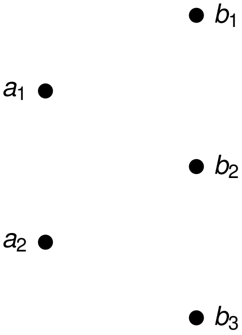
Definition (Vertex Cover)

Given a graph, finding the minimum subset of the vertices such that for every edge, at least one of its endpoints is in the set.

The minimal vertex cover is $\{a_1, a_2\}$. $\Longrightarrow$
The optimal change is removing f from the actions a_1 and a_2.

Results: Reducing to the Vertex Covering Problem



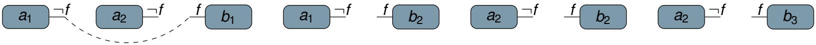


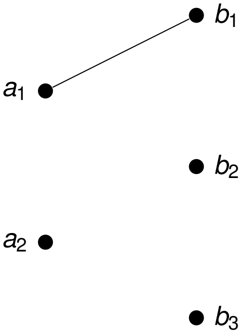
Definition (Vertex Cover)

Given a graph, finding the minimum subset of the vertices such that for every edge, at least one of its endpoints is in the set.

The minimal vertex cover is $\{a_1, a_2\}$. $\Longrightarrow$
The optimal change is removing f from the actions a_1 and a_2.

Results: Reducing to the Vertex Covering Problem



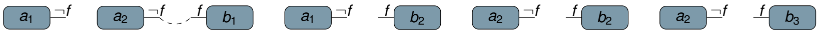


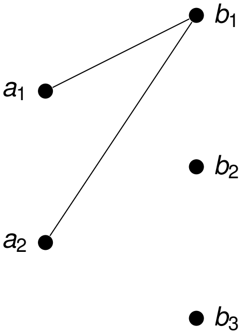
Definition (Vertex Cover)

Given a graph, finding the minimum subset of the vertices such that for every edge, at least one of its endpoints is in the set.

The minimal vertex cover is $\{a_1, a_2\}$. $\Longrightarrow$
The optimal change is removing f from the actions a_1 and a_2.

Results: Reducing to the Vertex Covering Problem



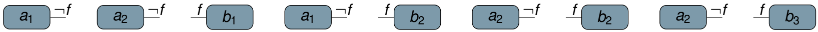


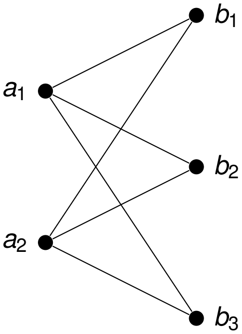
Definition (Vertex Cover)

Given a graph, finding the minimum subset of the vertices such that for every edge, at least one of its endpoints is in the set.

The minimal vertex cover is $\{a_1, a_2\}$. $\Longrightarrow$
The optimal change is removing f from the actions a_1 and a_2.

Results: Reducing to the Vertex Covering Problem



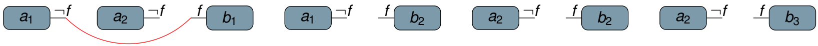


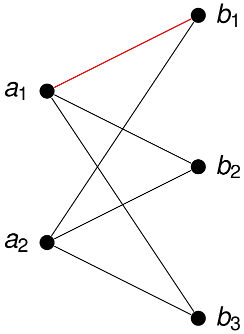
Definition (Vertex Cover)

Given a graph, finding the minimum subset of the vertices such that for every edge, at least one of its endpoints is in the set.

The minimal vertex cover is $\{a_1, a_2\}$. $\Longrightarrow$
The optimal change is removing f from the actions a_1 and a_2.

Results: Reducing to the Vertex Covering Problem





Definition (Vertex Cover)

Given a graph, finding the minimum subset of the vertices such that for every edge, at least one of its endpoints is in the set.

The minimal vertex cover is $\{a_1, a_2\}$. $\Longrightarrow$
The optimal change is removing f from the actions a_1 and a_2.

Results: Only FIX-ADD

Theorem

FIX-ACTION$^k_{\text{ADD}}$ *is NP-complete.*

Proof.

Suppose the given action sequence is $\overline{a} = a_1 \cdots a_n$.

- Membership: Guessing at most $\min\{k, \sum_{i=1}^{n} |prec(a_i)|\}$ changes.
- Hardness: Reducing from the set covering problem.

□

Results: Set Covering

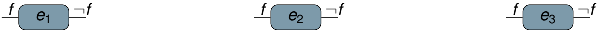
Definition

Given a set of sets $\mathcal{S} = \{S_1, \cdots, S_n\}$ and an integer k, is there a subset S' of S such that $|S| \leq k$ and $\bigcup_{S \in \mathcal{S}} S = \bigcup_{i=1}^{n} S_i$.

E.g., $\mathcal{S} = \{\underbrace{\{e_1\}}_{S_1}, \underbrace{\{e_1, e_2\}}_{S_2}, \underbrace{\{e_3\}}_{S_3}\}$ and $k = 2$.

- A solution to this instance is $S' = \{S_2,\ S_3\}$.

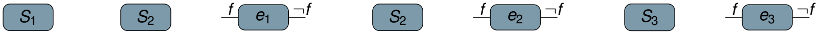
Results: Reduction from the Set Covering Problem



$S_1 = \{e_1\}, S_2 = \{e_1, e_2\}, S_3 = \{e_3\}$

- $e_i = (\{f\}, \emptyset, \{f\})$ $(1 \leq i \leq 3)$

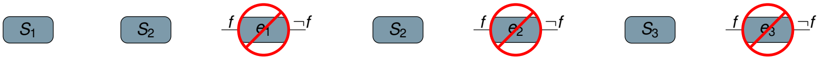
Results: Reduction from the Set Covering Problem



$S_1 = \{e_1\}, S_2 = \{e_1, e_2\}, S_3 = \{e_3\}$

- $e_i = (\{f\}, \emptyset, \{f\})$ $(1 \leq i \leq 3)$
- $S_i = (\emptyset, \emptyset, \emptyset)$ $(1 \leq i \leq 3)$

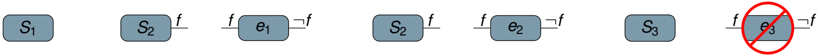
Results: Reduction from the Set Covering Problem



$S_1 = \{e_1\}, S_2 = \{e_1, e_2\}, S_3 = \{e_3\}$

- $e_i = (\{f\}, \emptyset, \{f\})$ $(1 \leq i \leq 3)$
- $S_i = (\emptyset, \emptyset, \emptyset)$ $(1 \leq i \leq 3)$

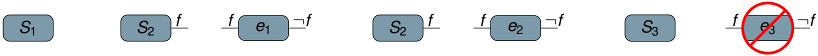
Results: Reduction from the Set Covering Problem



$S_1 = \{e_1\}, S_2 = \{e_1, e_2\}, S_3 = \{e_3\}$

- $e_i = (\{f\}, \emptyset, \{f\})$ $(1 \leq i \leq 3)$
- $S_i = (\emptyset, \emptyset, \emptyset)$ $(1 \leq i \leq 3)$

Results: Reduction from the Set Covering Problem



$S_1 = \{e_1\}, S_2 = \{e_1, e_2\}, S_3 = \{e_3\}$

- $e_i = (\{f\}, \emptyset, \{f\})$ $(1 \leq i \leq 3)$
- $S_i = (\emptyset, \emptyset, \emptyset)$ $(1 \leq i \leq 3)$

Corollary

FIX-ACTION$_X^k$ *is NP-complete if* ADD $\in X$.

Changing Planning Models in HTN Planning

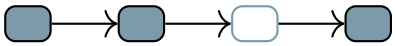
HTN Planning: Components

- State s: A set of proposition variables.

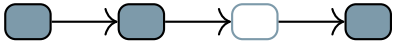
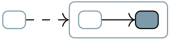
HTN Planning: Components

- State s: A set of proposition variables.
- Primitive tasks/actions $(prec, add, del)$
  - Preconditions $prec$: $prec \subseteq s$
  - Effects add and del: $(s \backslash del) \cup add$
- Compound tasks

HTN Planning: Components

- State s: A set of proposition variables.
- Primitive tasks/actions $(prec, add, del)$
  - Preconditions $prec$: $prec \subseteq s$
  - Effects add and del: $(s \setminus del) \cup add$
- Compound tasks
- Task networks 

HTN Planning: Components

- State s: A set of proposition variables.
- Primitive tasks/actions $(prec, add, del)$ 
  - Preconditions *prec*: $prec \subseteq s$
  - Effects *add* and *del*: $(s \setminus del) \cup add$
- Compound tasks 
- Task networks 
- Methods 

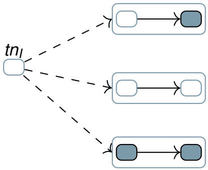
HTN Planning: Formalism

tn_I

$P = (D, tn_I, s_I)$ with
$D = (F, N_c, N_p, \delta, M)$

- F: A set of proposition variables called facts.
- N_c: A set of compound tasks.
- N_p: A set of primitive tasks (actions).
- δ: $N_p \rightarrow 2^F \times 2^F \times 2^F$
- M: A set of methods.
- tn_I: An initial task network.

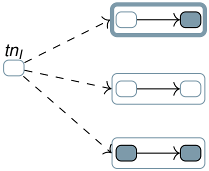
HTN Planning: Formalism



$P = (D, tn_I, s_I)$ with
$D = (F, N_c, N_p, \delta, M)$

- F: A set of proposition variables called facts.
- N_c: A set of compound tasks.
- N_p: A set of primitive tasks (actions).
- δ: $N_p \rightarrow 2^F \times 2^F \times 2^F$
- M: A set of methods.
- tn_I: An initial task network.

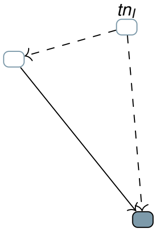
HTN Planning: Formalism



$P = (D, tn_I, s_I)$ with
$D = (F, N_c, N_p, \delta, M)$

- F: A set of proposition variables called facts.
- N_c: A set of compound tasks.
- N_p: A set of primitive tasks (actions).
- δ: $N_p \to 2^F \times 2^F \times 2^F$
- M: A set of methods.
- tn_I: An initial task network.

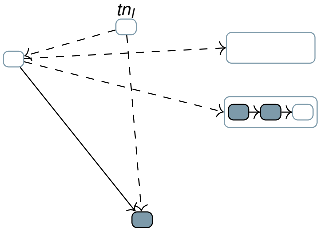
HTN Planning: Formalism



$P = (D, tn_I, s_I)$ with
$D = (F, N_c, N_p, \delta, M)$

- F: A set of proposition variables called facts.
- N_c: A set of compound tasks.
- N_p: A set of primitive tasks (actions).
- δ: $N_p \rightarrow 2^F \times 2^F \times 2^F$
- M: A set of methods.
- tn_I: An initial task network.

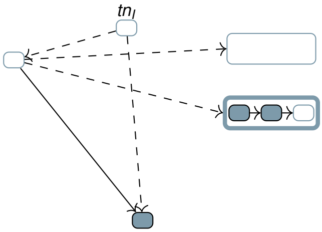
HTN Planning: Formalism



$P = (D, tn_I, s_I)$ with
$D = (F, N_c, N_p, \delta, M)$

- F: A set of proposition variables called facts.
- N_c: A set of compound tasks.
- N_p: A set of primitive tasks (actions).
- δ: $N_p \rightarrow 2^F \times 2^F \times 2^F$
- M: A set of methods.
- tn_I: An initial task network.

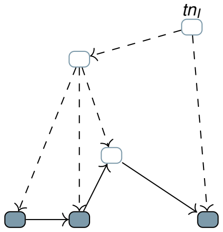
HTN Planning: Formalism



$P = (D, tn_I, s_I)$ with
$D = (F, N_c, N_p, \delta, M)$

- F: A set of proposition variables called facts.
- N_c: A set of compound tasks.
- N_p: A set of primitive tasks (actions).
- δ: $N_p \rightarrow 2^F \times 2^F \times 2^F$
- M: A set of methods.
- tn_I: An initial task network.

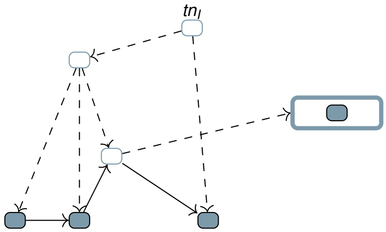
HTN Planning: Formalism



$P = (D, tn_I, s_I)$ with
$D = (F, N_c, N_p, \delta, M)$

- F: A set of proposition variables called facts.
- N_c: A set of compound tasks.
- N_p: A set of primitive tasks (actions).
- δ: $N_p \to 2^F \times 2^F \times 2^F$
- M: A set of methods.
- tn_I: An initial task network.

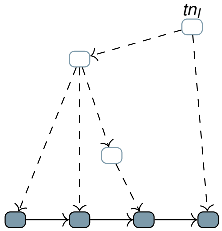
HTN Planning: Formalism



$P = (D, tn_I, s_I)$ with
$D = (F, N_c, N_p, \delta, M)$

- F: A set of proposition variables called facts.
- N_c: A set of compound tasks.
- N_p: A set of primitive tasks (actions).
- δ: $N_p \rightarrow 2^F \times 2^F \times 2^F$
- M: A set of methods.
- tn_I: An initial task network.

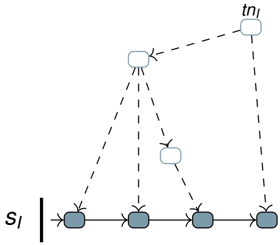
HTN Planning: Formalism



$P = (D, tn_I, s_I)$ with
$D = (F, N_c, N_p, \delta, M)$

- F: A set of proposition variables called facts.
- N_c: A set of compound tasks.
- N_p: A set of primitive tasks (actions).
- δ: $N_p \to 2^F \times 2^F \times 2^F$
- M: A set of methods.
- tn_I: An initial task network.

HTN Planning: Formalism



$P = (D, tn_I, s_I)$ with
$D = (F, N_c, N_p, \delta, M)$

- F: A set of proposition variables called facts.
- N_c: A set of compound tasks.
- N_p: A set of primitive tasks (actions).
- δ: $N_p \rightarrow 2^F \times 2^F \times 2^F$
- M: A set of methods.
- tn_I: An initial task network.
- s_I: An initial state.

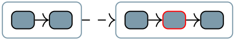
Changing Methods: Allowed Changes

Given an HTN planning problem and an action sequence, we want to change the planning model so that the given action sequence can be a solution.

Changing Methods: Allowed Changes

Given an HTN planning problem and an action sequence, we want to change the planning model so that the given action sequence can be a solution.

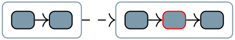
- ADD-ACTION: Adding an action to a method.



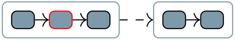
Changing Methods: Allowed Changes

Given an HTN planning problem and an action sequence, we want to change the planning model so that the given action sequence can be a solution.

- ADD-ACTION: Adding an action to a method.



- DEL-ACTION: Removing an action from a method.



Given Only an Action Sequence: Problem Definition

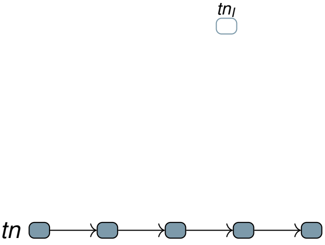
Definition (Fix-Meths$_X$, $X \subseteq \{\textsc{Add}, \textsc{Del}\}$ and $|X| \geq 1$)

Given an HTN planning problem P and an action sequence *tn*, is there a way to change the methods in P by applying the respective changes according to the value of X so that *tn* becomes a solution.

Given Only an Action Sequence: Problem Definition

Definition (FIX-METHS$_X$, $X \subseteq \{\text{ADD}, \text{DEL}\}$ and $|X| \geq 1$)

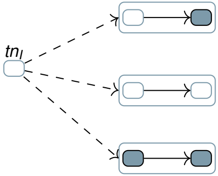
Given an HTN planning problem P and an action sequence tn, is there a way to change the methods in P by applying the respective changes according to the value of X so that tn becomes a solution.

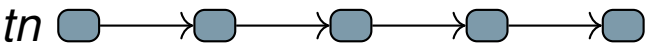


Given Only an Action Sequence: Problem Definition

Definition (FIX-METHS$_X$, $X \subseteq \{\text{ADD}, \text{DEL}\}$ and $|X| \geq 1$)

Given an HTN planning problem P and an action sequence tn, is there a way to change the methods in P by applying the respective changes according to the value of X so that tn becomes a solution.

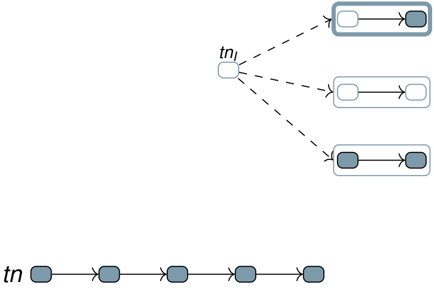




Given Only an Action Sequence: Problem Definition

Definition (FIX-METHS$_X$, $X \subseteq \{\text{ADD}, \text{DEL}\}$ and $|X| \geq 1$)

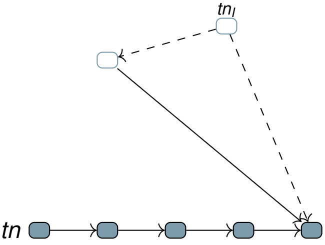
Given an HTN planning problem P and an action sequence tn, is there a way to change the methods in P by applying the respective changes according to the value of X so that tn becomes a solution.



Given Only an Action Sequence: Problem Definition

Definition (FIX-METHS$_X$, $X \subseteq \{\text{ADD}, \text{DEL}\}$ and $|X| \geq 1$)

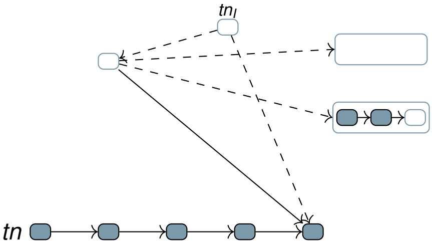
Given an HTN planning problem *P* and an action sequence *tn*, is there a way to change the methods in *P* by applying the respective changes according to the value of *X* so that *tn* becomes a solution.



Given Only an Action Sequence: Problem Definition

Definition (FIX-METHS$_X$, $X \subseteq \{\text{ADD}, \text{DEL}\}$ and $|X| \geq 1$)

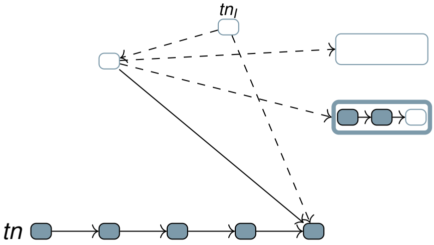
Given an HTN planning problem P and an action sequence tn, is there a way to change the methods in P by applying the respective changes according to the value of X so that tn becomes a solution.



Given Only an Action Sequence: Problem Definition

Definition (FIX-METHS$_X$, $X \subseteq \{\text{ADD}, \text{DEL}\}$ and $|X| \geq 1$)

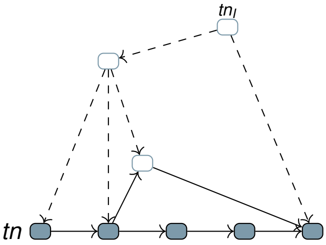
Given an HTN planning problem P and an action sequence tn, is there a way to change the methods in P by applying the respective changes according to the value of X so that tn becomes a solution.



Given Only an Action Sequence: Problem Definition

Definition (FIX-METHS$_X$, $X \subseteq \{\text{ADD}, \text{DEL}\}$ and $|X| \geq 1$)

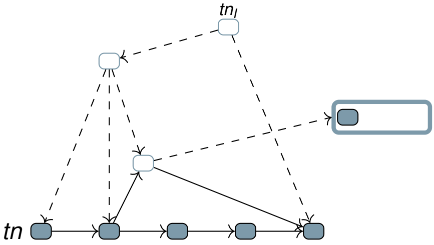
Given an HTN planning problem P and an action sequence tn, is there a way to change the methods in P by applying the respective changes according to the value of X so that tn becomes a solution.



Given Only an Action Sequence: Problem Definition

Definition (FIX-METHS$_X$, $X \subseteq \{$ADD, DEL$\}$ and $|X| \geq 1$)

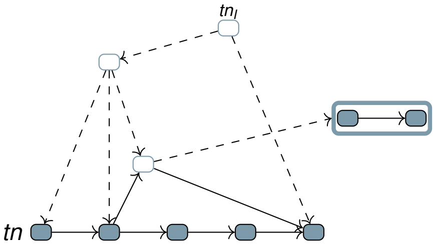
Given an HTN planning problem P and an action sequence tn, is there a way to change the methods in P by applying the respective changes according to the value of X so that tn becomes a solution.



Given Only an Action Sequence: Problem Definition

Definition (FIX-METHS$_X$, $X \subseteq \{\text{ADD}, \text{DEL}\}$ and $|X| \geq 1$)

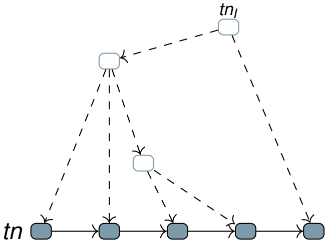
Given an HTN planning problem P and an action sequence tn, is there a way to change the methods in P by applying the respective changes according to the value of X so that tn becomes a solution.



Given Only an Action Sequence: Problem Definition

Definition (FIX-METHS$_X$, $X \subseteq \{\textsc{Add}, \textsc{Del}\}$ and $|X| \geq 1$)

Given an HTN planning problem P and an action sequence tn, is there a way to change the methods in P by applying the respective changes according to the value of X so that tn becomes a solution.



Given Only an Action Sequence: Membership

Theorem

FIX-METHS$_X$ *is in NP.*

Proof.

Key Observation: If there exists a sequence of changes that turns *tn* into a solution, then there must be one of length bounded by a polynomial.

- A totally-ordered HTN planning can be regarded as a CFG.
- Parsing based plan verification algorithms.

□

Given Only an Action Sequence

Theorem

FIX-METHS$_X$ *is NP-hard.*

Proof.

- Reducing from the independent set problem.

□

Given an Action Sequence & a Method Sequence: Problem Definition

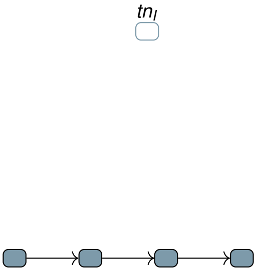
Definition (FIX-SEQ$_X$)

Given a planning problem P, a task network tn, and a method sequence $\overline{m}$. Is there a way to change the methods in P by using the allowed changes specified by X (e.g., ADD and DEL) such that $\overline{m}$ decomposes the initial task network of P into tn.

Given an Action Sequence & a Method Sequence: Problem Definition

Definition (FIX-SEQ$_X$)

Given a planning problem P, a task network tn, and a method sequence $\overline{m}$. Is there a way to change the methods in P by using the allowed changes specified by X (e.g., ADD and DEL) such that $\overline{m}$ decomposes the initial task network of P into tn.

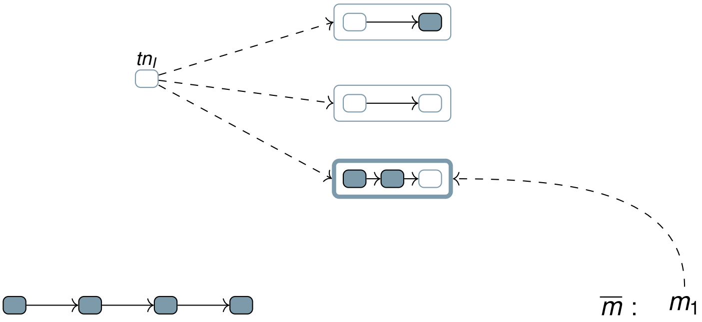


$\overline{m}$:

Given an Action Sequence & a Method Sequence: Problem Definition

Definition (FIX-SEQ$_X$)

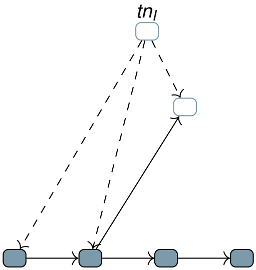
Given a planning problem P, a task network tn, and a method sequence $\overline{m}$. Is there a way to change the methods in P by using the allowed changes specified by X (e.g., ADD and DEL) such that $\overline{m}$ decomposes the initial task network of P into tn.



Given an Action Sequence & a Method Sequence: Problem Definition

Definition (FIX-SEQ$_X$)

Given a planning problem P, a task network tn, and a method sequence $\overline{m}$. Is there a way to change the methods in P by using the allowed changes specified by X (e.g., ADD and DEL) such that $\overline{m}$ decomposes the initial task network of P into tn.

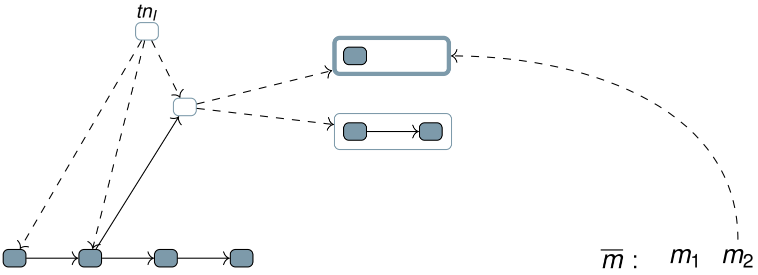


$\overline{m}$: m_1

Given an Action Sequence & a Method Sequence: Problem Definition

Definition (FIX-SEQ$_X$)

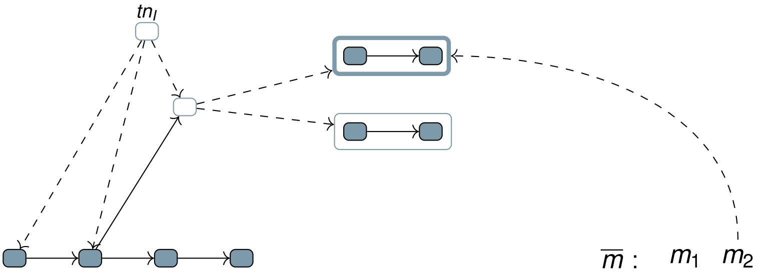
Given a planning problem P, a task network tn, and a method sequence $\overline{m}$. Is there a way to change the methods in P by using the allowed changes specified by X (e.g., ADD and DEL) such that $\overline{m}$ decomposes the initial task network of P into tn.



Given an Action Sequence & a Method Sequence: Problem Definition

Definition (FIX-SEQ$_X$)

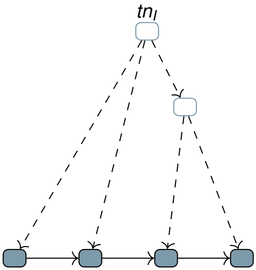
Given a planning problem *P*, a task network *tn*, and a method sequence $\overline{m}$. Is there a way to change the methods in *P* by using the allowed changes specified by *X* (e.g., ADD and DEL) such that $\overline{m}$ decomposes the initial task network of *P* into *tn*.



Given an Action Sequence & a Method Sequence: Problem Definition

Definition (FIX-SEQ$_X$)

Given a planning problem P, a task network tn, and a method sequence $\overline{m}$. Is there a way to change the methods in P by using the allowed changes specified by X (e.g., ADD and DEL) such that $\overline{m}$ decomposes the initial task network of P into tn.



$\overline{m}$: m_1 m_2

Given an Action Sequence & a Method Sequence

Theorem

FIX-SEQ$_X$ *is NP-complete.*

Proof.

- Reduction from the independent set problem again.

□

Optimization: Finding the Minimal Number of Changes

Definition ($\textsc{Fix-Meths}_X^k$ & $\textsc{Fix-Seq}_X^k$)

Given an integer k, the problems $\textsc{Fix-Meths}_X^k$ and $\textsc{Fix-Seq}_X^k$ are identical to $\textsc{Fix-Meths}_X$ and $\textsc{Fix-Seq}_X$ except that we bounded the number of changes by k.

Corollary

$\textsc{Fix-Meths}_X^k$ and $\textsc{Fix-Seq}_X^k$ are NP-complete.

Conclusion

Summary

Changes	Complexity
prec *del* *prec*, *del*	P
add *prec*, *add* *del*, *add* *prec*, *add*, *del*	NP-complete

Computational Complexity of Changing Actions.

Methods Given?	Changes	Complexity	
		Any Changes	*k Changes*
No	Del Add Add, Del	NP-complete	NP-complete
Yes	All	NP-complete	NP-complete
Yes: Unique	All	P	P

Computational Complexity of Changing Methods.