On the Complexity of HTN Plan Verification and its Implications for Plan Recognition

Gregor Behnke, Daniel Höller, Susanne Biundo

Ulm University, Institute of Artificial Intelligence

June 9, 2015

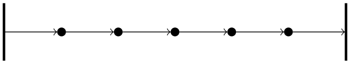
sfb transregio 62
Companion Technology

ulm university universität uulm

Deutsche Forschungsgemeinschaft
DFG

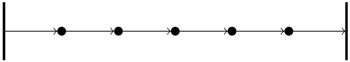
Plan Verification

Are we there yet?



Plan Verification

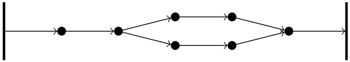
Are we there yet?



- $\mathcal{O}(n)$ for totally ordered classical plans

Plan Verification

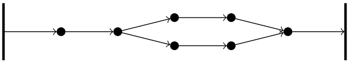
Are we there yet?



- $\mathcal{O}(n)$ for totally ordered classical plans
- $\mathcal{O}(n^2)$ for POCL plans

Plan Verification

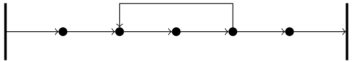
Are we there yet?



- $\mathcal{O}(n)$ for totally ordered classical plans
- $\mathcal{O}(n^2)$ for POCL plans
- $\mathbb{NP}$-complete for PO planning (Chapman 1987; Nebel and Bäckström 1994)

Plan Verification

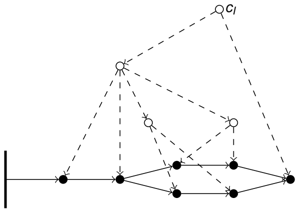
Are we there yet?



- $\mathcal{O}(n)$ for totally ordered classical plans
- $\mathcal{O}(n^2)$ for POCL plans
- $\mathbb{NP}$-complete for PO planning (Chapman 1987; Nebel and Bäckström 1994)
- Π_2^P-complete for a plans with control structures (Lang and Zanuttini 2012)

Plan Verification

Are we there yet?



- $\mathcal{O}(n)$ for totally ordered classical plans
- $\mathcal{O}(n^2)$ for POCL plans
- $\mathbb{NP}$-complete for PO planning (Chapman 1987; Nebel and Bäckström 1994)
- Π_2^P-complete for a plans with control structures (Lang and Zanuttini 2012)
- unknown for HTN planning

Why plan verification?

Plan Verification

Why plan verification?

Plan Verification

- post-optimization of solutions

Why plan verification?

Plan Verification

- post-optimization of solutions
- plan repair

Why plan verification?

Plan Verification

- post-optimization of solutions
- plan repair
- implications for Plan Recognition (HTN *Plan Libraries*)

Why plan verification?

Plan Verification

- post-optimization of solutions
- plan repair
- implications for Plan Recognition (HTN *Plan Libraries*)

What have we done?

Why plan verification?

Plan Verification

- post-optimization of solutions
- plan repair
- implications for Plan Recognition (HTN *Plan Libraries*)

What have we done?

1. HTN Plan Verification is $\mathbb{NP}$ complete

Why plan verification?

Plan Verification

- post-optimization of solutions
- plan repair
- implications for Plan Recognition (HTN *Plan Libraries*)

What have we done?

1. HTN Plan Verification is $\mathbb{NP}$ complete
2. Plan Compatibility is $\mathbb{NP}$ complete

Why plan verification?

Plan Verification

- post-optimization of solutions
- plan repair
- implications for Plan Recognition (HTN *Plan Libraries*)

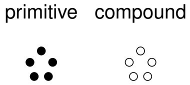
What have we done?

1. HTN Plan Verification is $\mathbb{NP}$ complete
2. Plan Compatibility is $\mathbb{NP}$ complete
3. implications for Plan Recognition

Hierarchical Task Network (HTN) Planning

$$\mathcal{P} = (P, C, c_I, M, L, s_I)$$

Hierarchical Task Network (HTN) Planning



$\mathcal{P} = (P, C, c_I, M, L, s_I)$

- P a set of primitive tasks
- C a set of compound tasks

Hierarchical Task Network (HTN) Planning

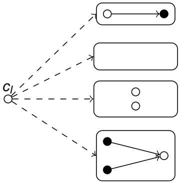
$\mathcal{P} = (P, C, c_I, M, L, s_I)$

- P a set of primitive tasks
- C a set of compound tasks
- $c_I \in C$ the initial task

A solution $\mathrm{tn} \in Sol(\mathcal{P})$ must

- be a refinement of the initial task

Hierarchical Task Network (HTN) Planning



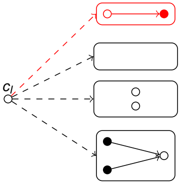
$\mathcal{P} = (P, C, c_I, M, L, s_I)$

- P a set of primitive tasks
- C a set of compound tasks
- $c_I \in C$ the initial task
- $M \subseteq C \times 2^{TN}$ the methods

A solution $\mathrm{tn} \in Sol(\mathcal{P})$ must

- be a refinement of the initial task

Hierarchical Task Network (HTN) Planning



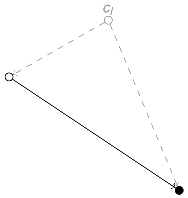
$\mathcal{P} = (P, C, c_I, M, L, s_I)$

- P a set of primitive tasks
- C a set of compound tasks
- $c_I \in C$ the initial task
- $M \subseteq C \times 2^{TN}$ the methods

A solution $\mathrm{tn} \in Sol(\mathcal{P})$ must

- be a refinement of the initial task

Hierarchical Task Network (HTN) Planning



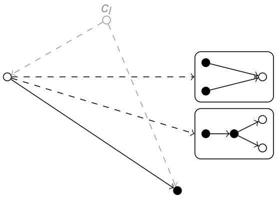
$\mathcal{P} = (P, C, c_I, M, L, s_I)$

- P a set of primitive tasks
- C a set of compound tasks
- $c_I \in C$ the initial task
- $M \subseteq C \times 2^{TN}$ the methods

A solution $\mathrm{tn} \in Sol(\mathcal{P})$ must

- be a refinement of the initial task

Hierarchical Task Network (HTN) Planning



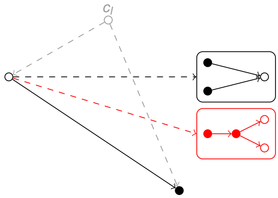
$\mathcal{P} = (P, C, c_I, M, L, s_I)$

- P a set of primitive tasks
- C a set of compound tasks
- $c_I \in C$ the initial task
- $M \subseteq C \times 2^{TN}$ the methods

A solution $\mathrm{tn} \in Sol(\mathcal{P})$ must

- be a refinement of the initial task

Hierarchical Task Network (HTN) Planning



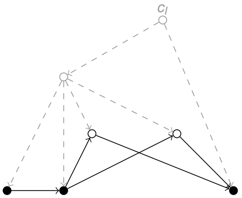
$\mathcal{P} = (P, C, c_I, M, L, s_I)$

- P a set of primitive tasks
- C a set of compound tasks
- $c_I \in C$ the initial task
- $M \subseteq C \times 2^{TN}$ the methods

A solution $\mathrm{tn} \in Sol(\mathcal{P})$ must

- be a refinement of the initial task

Hierarchical Task Network (HTN) Planning



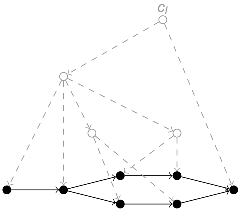
$\mathcal{P} = (P, C, c_I, M, L, s_I)$

- P a set of primitive tasks
- C a set of compound tasks
- $c_I \in C$ the initial task
- $M \subseteq C \times 2^{TN}$ the methods

A solution $\mathrm{tn} \in Sol(\mathcal{P})$ must

- be a refinement of the initial task

Hierarchical Task Network (HTN) Planning



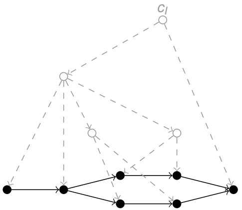
$\mathcal{P} = (P, C, c_I, M, L, s_I)$

- P a set of primitive tasks
- C a set of compound tasks
- $c_I \in C$ the initial task
- $M \subseteq C \times 2^{TN}$ the methods

A solution $\mathrm{tn} \in Sol(\mathcal{P})$ must

- be a refinement of the initial task
- only contain primitive tasks

Hierarchical Task Network (HTN) Planning



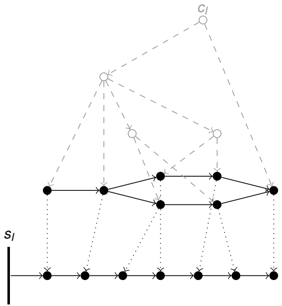
$\mathcal{P} = (P, C, c_I, M, L, s_I)$

- P a set of primitive tasks
- C a set of compound tasks
- $c_I \in C$ the initial task
- $M \subseteq C \times 2^{TN}$ the methods
- L a set of fluent

A solution $\mathrm{tn} \in Sol(\mathcal{P})$ must

- be a refinement of the initial task
- only contain primitive tasks

Hierarchical Task Network (HTN) Planning



$\mathcal{P} = (P, C, c_I, M, L, s_I)$

- P a set of primitive tasks
- C a set of compound tasks
- $c_I \in C$ the initial task
- $M \subseteq C \times 2^{TN}$ the methods
- L a set of fluent
- $s_I \subseteq L$ the initial state

A solution $\mathrm{tn} \in Sol(\mathcal{P})$ must

- be a refinement of the initial task
- only contain primitive tasks
- have a linearization, executable from the initial state

Plan Verification

Definition (VERIFYTN)

Let $\mathcal{P}$ be a planning problem and tn be a task network.
Decide whether $\mathrm{tn} \in \mathit{Sol}(\mathcal{P})$.

Plan Verification

Definition (VERIFYTN)

Let $\mathcal{P}$ be a planning problem and tn be a task network.
Decide whether $\mathrm{tn} \in \mathit{Sol}(\mathcal{P})$.

What do we have to check?

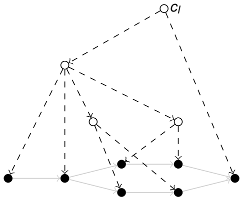
Plan Verification

Definition (VERIFYTN)

Let $\mathcal{P}$ be a planning problem and tn be a task network.
Decide whether $\mathrm{tn} \in \mathit{Sol}(\mathcal{P})$.

What do we have to check?

- refinement



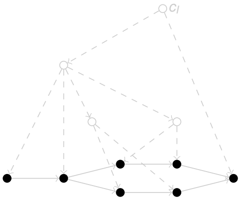
Plan Verification

Definition (VERIFYTN)

Let $\mathcal{P}$ be a planning problem and tn be a task network.
Decide whether $\mathrm{tn} \in \mathit{Sol}(\mathcal{P})$.

What do we have to check?

- refinement
- primitive



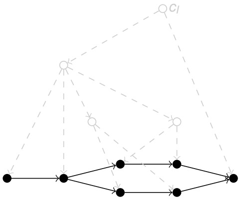
Plan Verification

Definition (VERIFYTN)

Let $\mathcal{P}$ be a planning problem and tn be a task network.
Decide whether $\mathrm{tn} \in \mathit{Sol}(\mathcal{P})$.

What do we have to check?

- refinement
- primitive
- executability



VERIFYTN: $\mathbb{NP}$-hardness

Theorem

VERIFYTN *is* $\mathbb{NP}$*-hard*

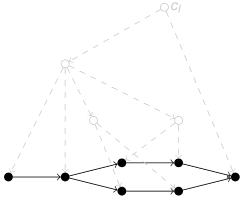
[1] (Erol, Hendler, and Nau 1994; Nebel and Bäckström 1994)

VERIFYTN: $\mathbb{NP}$-hardness

Theorem

VERIFYTN *is* $\mathbb{NP}$*-hard*

Proof.



Checking whether a partially ordered set of actions has an executable linearization is $\mathbb{NP}$-hard[1].

□

[1] (Erol, Hendler, and Nau 1994; Nebel and Bäckström 1994)

VERIFYTN: $\mathbb{NP}$-membership

Theorem

VERIFYTN *is in* $\mathbb{NP}$

VERIFYTN: $\mathbb{NP}$-membership

Theorem

VERIFYTN *is in* $\mathbb{NP}$

Proof.

Adapt the proof by Höller et al. (2014), showing that *Sol*($\mathcal{P}$) form a context sensitive language.

VERIFYTN: $\mathbb{NP}$-membership

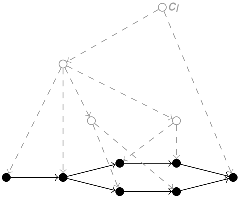
Theorem

VERIFYTN *is in* $\mathbb{NP}$

Proof.

Adapt the proof by Höller et al. (2014), showing that $Sol(\mathcal{P})$ form a context sensitive language.

1. guess a linearization and check executability



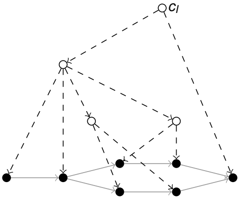
VERIFYTN: $\mathbb{NP}$-membership

Theorem

VERIFYTN *is in* $\mathbb{NP}$

Proof.
Adapt the proof by Höller et al. (2014), showing that $Sol(\mathcal{P})$ form a context sensitive language.

1. guess a linearization and check executability
2. guess decompositions and check them



VERIFYTN: $\mathbb{NP}$-membership

Proof. (continued)

VERIFYTN: $\mathbb{NP}$-membership

Proof. (continued)

starting with c_I, guess decompositions and apply them

repeat until tn has been found

VerifyTN: NP-membership

Proof. (continued)

starting with c_I, guess decompositions and apply them

repeat until tn has been found

Not sufficient (termination)

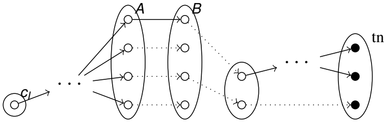
VERIFYTN: $\mathbb{NP}$-membership

Proof. (continued)

starting with c_I, guess decompositions and apply them

repeat until tn has been found

Not sufficient (termination)



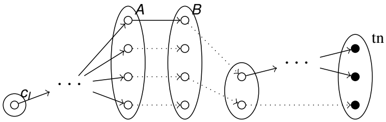
VERIFYTN: $\mathbb{NP}$-membership

Proof. (continued)

starting with c_I, guess decompositions and apply them

repeat until tn has been found

Not sufficient (termination)



- handle decompositions with $|\mathrm{tn}_m| = 0$

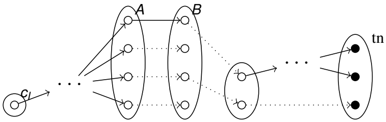
VERIFYTN: $\mathbb{NP}$-membership

Proof. (continued)

starting with c_I, guess decompositions and apply them

repeat until tn has been found

Not sufficient (termination)



- handle decompositions with $|\mathrm{tn}_m| = 0$
- estimate maximal number of decompositions with $|\mathrm{tn}_m| \geq 1$ needed to obtain any task network of size $|\mathrm{tn}|$

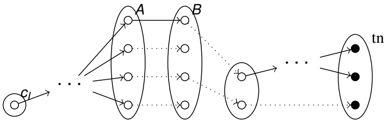
VERIFYTN: $\mathbb{NP}$-membership

Proof. (continued)

starting with c_I, guess decompositions and apply them

repeat until tn has been found

Not sufficient (termination)



- handle decompositions with $|\mathrm{tn}_m| = 0$
- estimate maximal number of decompositions with $|\mathrm{tn}_m| \geq 1$ needed to obtain any task network of size $|\mathrm{tn}|$
- result: $|\mathrm{tn}|(|C| + 1)$

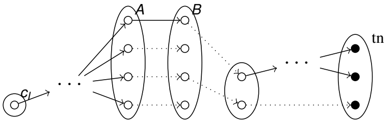
VERIFYTN: $\mathbb{NP}$-membership

Proof. (continued)

starting with c_I, guess decompositions and apply them

repeat until tn has been found

Not sufficient (termination)



- handle decompositions with $|\mathrm{tn}_m| = 0$
- estimate maximal number of decompositions with $|\mathrm{tn}_m| \geq 1$ needed to obtain any task network of size $|\mathrm{tn}|$
- result: $|\mathrm{tn}|(|C| + 1)$
- abort if more decompositions have been applied

□

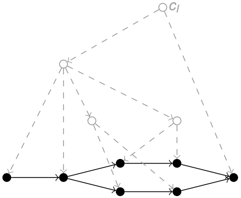
Plan Verification

Main reason for $\mathbb{NP}$-hardness:

Plan Verification

Main reason for $\mathbb{NP}$-hardness:

Find an executable linearization.



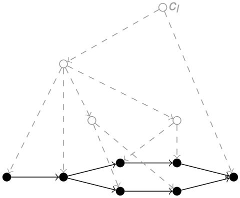
Plan Verification

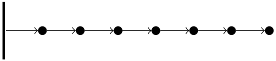
Main reason for $\mathbb{NP}$-hardness:

Find an executable linearization.

Suppose we already have one.

- by an observation
- by using *hybrid planning*, fusing HTN and POCL





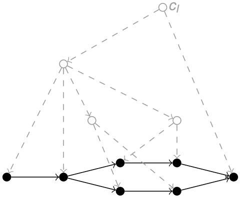
Plan Verification

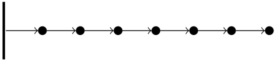
Main reason for $\mathbb{NP}$-hardness:

Find an executable linearization.

Suppose we already have one.

- by an observation
- by using *hybrid planning*, fusing HTN and POCL





Definition (VERIFYSEQ)

Let $\mathcal{P}$ be a planning problem and ω a sequence of actions.
Decide whether $\exists \mathrm{tn} \in \mathit{Sol}(\mathcal{P})$ and ω is an executable linearization of tn.

VERIFYSEQ: $\mathbb{NP}$-membership

Theorem

VERIFYSEQ *is in* $\mathbb{NP}$.

VERIFYSEQ: $\mathbb{NP}$-membership

Theorem

VERIFYSEQ *is in* $\mathbb{NP}$.

Proof.

Straightforward adaptation of previous proof. □

VERIFYSEQ: $\mathbb{NP}$-hardness

Theorem

VERIFYSEQ *is* $\mathbb{NP}$*-hard.*

VERIFYSEQ: $\mathbb{NP}$-hardness

Theorem

VERIFYSEQ *is* $\mathbb{NP}$*-hard.*

Proof (idea).

Reduction from VERTEXCOVER.

VERIFYSEQ: $\mathbb{NP}$-hardness

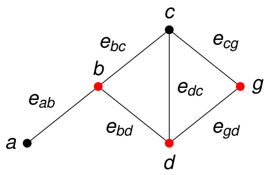
Theorem

VERIFYSEQ *is* $\mathbb{NP}$*-hard.*

Proof (idea).

Reduction from VERTEXCOVER.

Reminder: Decide, given a graph $G = (V, E)$ and a number k, whether $\exists V_C \subseteq V$ s.t. $|V_C| \leq k$ and each edge e is adjacent to a node in the cover V_C.



VerifySeq: $\mathbb{NP}$-hardness

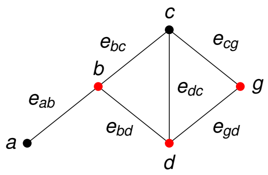
Theorem

VerifySeq *is* $\mathbb{NP}$*-hard.*

Proof (idea).

Reduction from VertexCover.

Reminder: Decide, given a graph $G = (V, E)$ and a number k, whether $\exists V_C \subseteq V$ s.t. $|V_C| \leq k$ and each edge e is adjacent to a node in the cover V_C.



- V_C is chosen by decomposition of vertex-tasks

VERIFYSEQ: $\mathbb{NP}$-hardness

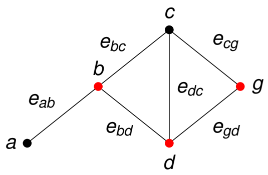
Theorem

VERIFYSEQ *is* $\mathbb{NP}$*-hard.*

Proof (idea).

Reduction from VERTEXCOVER.

Reminder: Decide, given a graph $G = (V, E)$ and a number k, whether $\exists V_C \subseteq V$ s.t. $|V_C| \leq k$ and each edge e is adjacent to a node in the cover V_C.



- V_C is chosen by decomposition of vertex-tasks
- ω ensures that $\leq k$ nodes are selected

VERIFYSEQ: $\mathbb{NP}$-hardness

$\mathbb{NP}$-completeness holds even for severely restricted HTN Planning Problems

The constructed domain needs

- neither preconditions nor effects
- no ordering constraints
- no cycles in the decomposition hierarchy
- only a depth of 2 (1 with an initial task network)

Plan Recognition

Plan Recognition

- Given observed actions, decide which goal the user pursues

Plan Recognition

- Given observed actions, decide which goal the user pursues
- Given observed actions, decide whether they can lead to a solution

Plan Recognition

- Given observed actions, decide which goal the user pursues
- Given observed actions, decide whether they can lead to a solution
- HTNs are commonly used as *Plan Libraries*

Plan Recognition

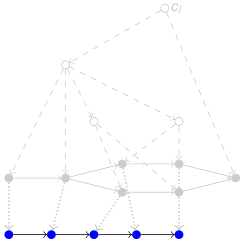
Definition (PLANREG)

Let $\mathcal{P}$ be a planning problem and ω a sequence of actions.
Decide whether $\exists \mathrm{tn} \in Sol(\mathcal{P})$ and ω is **a prefix of** a linearization of tn.

Plan Recognition

Definition (PLANREG)

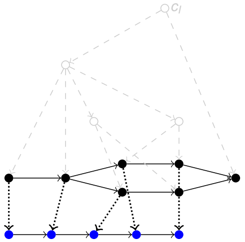
Let $\mathcal{P}$ be a planning problem and ω a sequence of actions.
Decide whether $\exists \mathrm{tn} \in Sol(\mathcal{P})$ and ω is **a prefix of** a linearization of tn.



Plan Recognition

Definition (PLANREG)

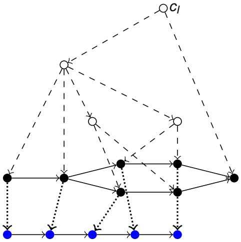
Let $\mathcal{P}$ be a planning problem and ω a sequence of actions.
Decide whether $\exists \mathrm{tn} \in \mathit{Sol}(\mathcal{P})$ and ω is **a prefix of** a linearization of tn.



Plan Recognition

Definition (PLANREG)

Let $\mathcal{P}$ be a planning problem and ω a sequence of actions.
Decide whether $\exists \mathrm{tn} \in \mathit{Sol}(\mathcal{P})$ and ω is **a prefix of** a linearization of tn.



PLANREC

Theorem

PLANREC *is strictly semi-decidable.*

PLANREC

Theorem

PLANREC *is strictly semi-decidable.*

Proof:

Undecidability: Choose $\omega = \varepsilon$.
This is equivalent to the question whether *Sol*$(\mathcal{P}) \neq \emptyset$.

PLANREC

Theorem

PLANREC *is strictly semi-decidable.*

Proof:

Undecidability: Choose $\omega = \varepsilon$.
This is equivalent to the question whether *Sol*$(\mathcal{P}) \neq \emptyset$.
Semi-decidability: *Sol*$(\mathcal{P})$ is enumerable.
Generate next solution and check. □

PLANREC

Theorem

PLANREC *is strictly semi-decidable.*

Proof:

Undecidability: Choose $\omega = \varepsilon$.
This is equivalent to the question whether *Sol*$(\mathcal{P}) \neq \emptyset$.
Semi-decidability: *Sol*$(\mathcal{P})$ is enumerable.
Generate next solution and check. □

Suppose we know that the sequence is complete.

PLANREC

Theorem

PLANREC *is strictly semi-decidable.*

Proof:

Undecidability: Choose $\omega = \varepsilon$.
This is equivalent to the question whether *Sol*$(\mathcal{P}) \neq \emptyset$.
Semi-decidability: *Sol*$(\mathcal{P})$ is enumerable.
Generate next solution and check. □

Suppose we know that the sequence is complete.

This is VERIFYSEQ, i.e., still $\mathbb{NP}$-complete.

Conclusion

- HTN Plan Verification is $\mathbb{NP}$-*complete*
  - for task networks
  - for task sequences
- still $\mathbb{NP}$-*complete* for severely restricted HTN Planning Problems
- HTN Plan Recognition is strictly semi-decidable
- even if the complete plan has been observed, it is still $\mathbb{NP}$-*complete*
- Plan Compatibility is $\mathbb{NP}$-*complete*

Conclusion

- HTN Plan Verification is $\mathbb{NP}$-*complete*
    - for task networks
    - for task sequences
- still $\mathbb{NP}$-*complete* for severely restricted HTN Planning Problems
- HTN Plan Recognition is strictly semi-decidable
- even if the complete plan has been observed, it is still $\mathbb{NP}$-*complete*
- Plan Compatibility is $\mathbb{NP}$-*complete*

Hierarchical Task Network Planning

A *task network* $\mathrm{tn} = (T, \prec, \alpha)$ is a partially ordered set of tasks

- T is a finite set of tasks
- $\prec \; \subseteq T \times T$ is a strict partial order on T
- $\alpha : T \mapsto C \cup O$ the action for each task

Hierarchical Task Network Planning

A *task network* $\mathrm{tn} = (T, \prec, \alpha)$ is a partially ordered set of tasks

- T is a finite set of tasks
- $\prec \ \subseteq T \times T$ is a strict partial order on T
- $\alpha : T \mapsto C \cup O$ the action for each task

A *planning problem* is a 6-tuple $\mathcal{P} = (V, O, C, M, c_I, s_I)$

- V is a finite set of *state variables*
- O is a finite set of *primitive tasks*, for $o \in O$, $(\mathrm{prec}(o), \mathrm{add}(o), \mathrm{del}(o)) \in 2^V \times 2^V \times 2^V$ is an *operator*
- C is a finite set of *compound tasks*
- $M \subseteq C \times TN$ is a finite set of *decomposition methods*

Hierarchical Task Network Planning

A *task network* $\mathrm{tn} = (T, \prec, \alpha)$ is a partially ordered set of tasks

- T is a finite set of tasks
- $\prec \; \subseteq T \times T$ is a strict partial order on T
- $\alpha : T \mapsto C \cup O$ the action for each task

A *planning problem* is a 6-tuple $\mathcal{P} = (V, O, C, M, c_I, s_I)$

- V is a finite set of *state variables*
- O is a finite set of *primitive tasks*, for $o \in O$, $(\mathrm{prec}(o), \mathrm{add}(o), \mathrm{del}(o)) \in 2^V \times 2^V \times 2^V$ is an *operator*
- C is a finite set of *compound tasks*
- $M \subseteq C \times TN$ is a finite set of *decomposition methods*
- $c_I \in C$ is the *initial task*
- $s_I \in 2^V$ is the *initial state*

HTN Modifications

Decomposition:

- Given a task network $\mathrm{tn} = (T, \prec, \alpha)$,use method $(t, \mathrm{tn}') \in M$ to replace $t \in T$ by tn'

HTN Modifications

Decomposition:

- Given a task network $\mathrm{tn} = (T, \prec, \alpha)$,use method $(t, \mathrm{tn}') \in M$ to replace $t \in T$ by tn'

Task Insertion:

- Insert primitive tasks from O

HTN Solutions

- A task network tn is an HTN solution iff:
  - tn is obtained via decomposition
  - contains only primitive tasks
  - there is an executable linearization of tn's tasks
- $Sol_{HTN}(\mathcal{P})$ denotes the set of all solutions to a problem $\mathcal{P}$

HTN Solutions

- A task network tn is an HTN solution iff:
  - tn is obtained via decomposition
  - contains only primitive tasks
  - there is an executable linearization of tn's tasks
- $Sol_{HTN}(\mathcal{P})$ denotes the set of all solutions to a problem $\mathcal{P}$

- A task network tn is a TIHTN solution iff:
  - tn is obtained by decomposition followed by insertion
  - contains only primitive tasks
  - there is an executable linearization of tn's tasks
- $Sol_{TiHTN}(\mathcal{P})$ denotes the set of all solutions to a problem $\mathcal{P}$

VERIFYSEQ: $\mathbb{NP}$-membership and hardness

Theorem

VERIFYSEQ *is in* $\mathbb{NP}$.

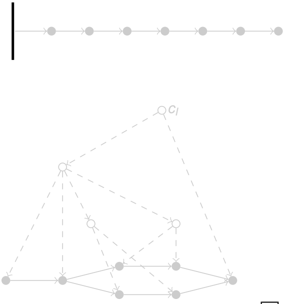
VERIFYSEQ: $\mathbb{NP}$-membership and hardness

Theorem

VERIFYSEQ *is in* $\mathbb{NP}$.

Proof.

Straightforward adaptation of previous proof.



□

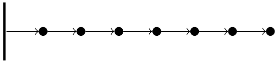
VERIFYSEQ: $\mathbb{NP}$-membership and hardness

Theorem

VERIFYSEQ *is in* $\mathbb{NP}$.

Proof.

Straightforward adaptation of previous proof.



- check executability of ω

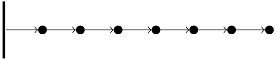
□

VERIFYSEQ: $\mathbb{NP}$-membership and hardness

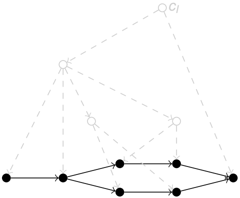
Theorem

VERIFYSEQ *is in* $\mathbb{NP}$.

Proof.
Straightforward adaptation of previous proof.



- check executability of ω
- guess a tn with linearization ω



□

VERIFYSEQ: $\mathbb{NP}$-membership and hardness

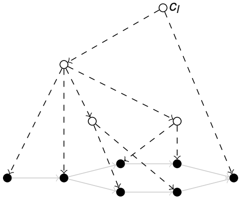
Theorem

VERIFYSEQ *is in* $\mathbb{NP}$.

Proof.

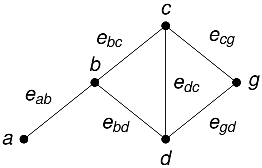
Straightforward adaptation of previous proof.

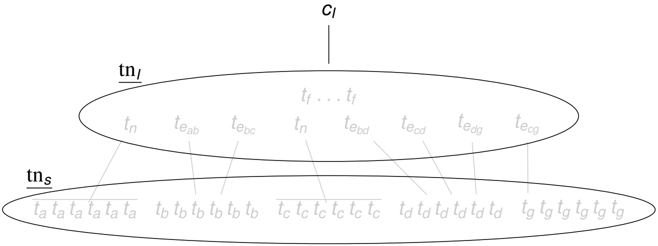
- check executability of ω
- guess a tn with linearization ω
- check whether tn can be decomposed form c_I



□

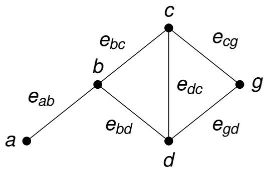
VERIFYSEQ: $\mathbb{NP}$-hardness

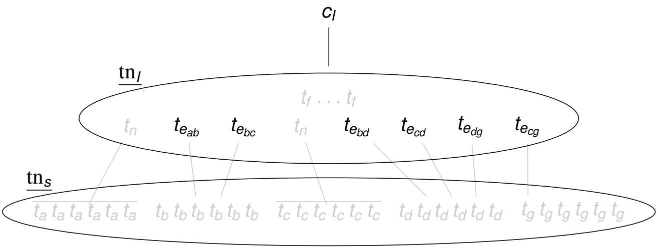




VERIFYSEQ: $\mathbb{NP}$-hardness

represent each edge by an abstract task

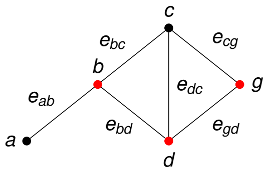


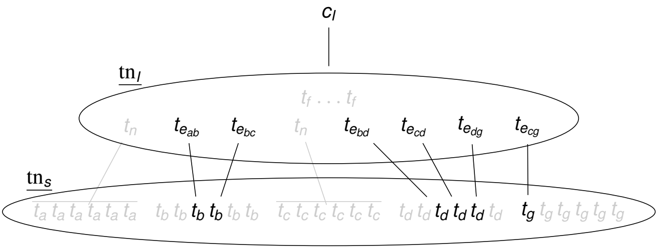


VERIFYSEQ: $\mathbb{NP}$-hardness

represent each edge by an abstract task

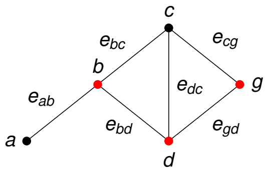
decomposition determines which node is in the VC

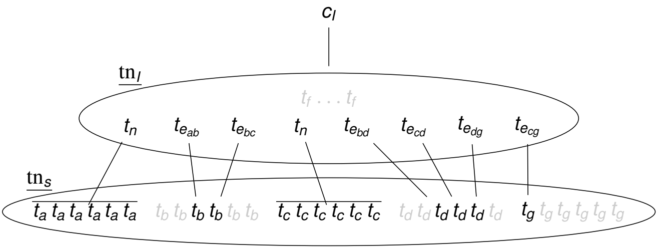




VERIFYSEQ: $\mathbb{NP}$-hardness

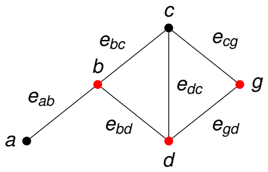
- represent each edge by an abstract task
- decomposition determines which node is in the VC
- enforce with ω and other abstract tasks, that at most k different tasks can be chosen

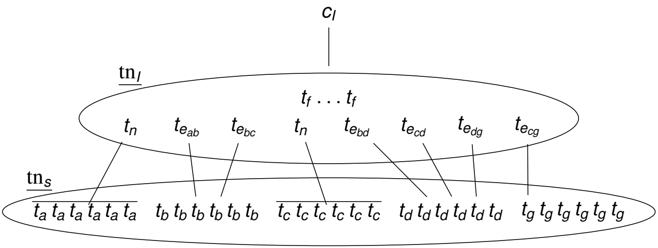




VERIFYSEQ: $\mathbb{NP}$-hardness

- represent each edge by an abstract task
- decomposition determines which node is in the VC
- enforce with ω and other abstract tasks, that at most k different tasks can be chosen





□