Lecture *Hierarchical Planning*

Chapter:
Complexity Results for Plan Verification

Dr. Pascal Bercher

Institute of Artificial Intelligence,
Ulm University, Germany

Winter Term 2018/2019

(Compiled on: February 20, 2019)

ulm university universität uulm

Overview:

What are Complexity Studies?

Complexity analysis studies the computational hardness of a decision problem. In this lecture we study:

What are Complexity Studies?

Complexity analysis studies the computational hardness of a decision problem. In this lecture we study:

- The *plan existence problem*:

What are Complexity Studies?

Complexity analysis studies the computational hardness of a decision problem. In this lecture we study:

- The *plan existence problem*:
  How hard is it to decide whether a problem $\mathcal{P}$ has a solution?

What are Complexity Studies?

Complexity analysis studies the computational hardness of a decision problem. In this lecture we study:

- The *plan existence problem*:
  How hard is it to decide whether a problem $\mathcal{P}$ has a solution?
- The *plan verification problem*:

What are Complexity Studies?

Complexity analysis studies the computational hardness of a decision problem. In this lecture we study:

- The *plan existence problem*:
  How hard is it to decide whether a problem $\mathcal{P}$ has a solution?
- The *plan verification problem*:
  How hard is it to decide whether a given plan is actually a solution?

Why are we Interested in Complexity Studies?

Benefits of complexity studies:

Why are we Interested in Complexity Studies?

Benefits of complexity studies:

- We know how to design algorithms:

Why are we Interested in Complexity Studies?

Benefits of complexity studies:

- We know how to design algorithms:
  - If a problem is undecidable, any terminating algorithm must be wrong. Similarly: if a problem is $\mathbb{NP}$-*complete*, it is not a good idea to design a decision procedure that runs in polynomial time.

Why are we Interested in Complexity Studies?

Benefits of complexity studies:

- We know how to design algorithms:
  - If a problem is undecidable, any terminating algorithm must be wrong. Similarly: if a problem is $\mathbb{NP}$-*complete*, it is not a good idea to design a decision procedure that runs in polynomial time.
  - If the complexity of a problem is not known, at which runtime should we aim? $\mathbb{P}$? $\mathbb{EXPTIME}$?

Why are we Interested in Complexity Studies?

Benefits of complexity studies:

- We know how to design algorithms:
  - If a problem is undecidable, any terminating algorithm must be wrong. Similarly: if a problem is $\mathbb{NP}$-*complete*, it is not a good idea to design a decision procedure that runs in polynomial time.
  - If the complexity of a problem is not known, at which runtime should we aim? $\mathbb{P}$? $\mathbb{EXPTIME}$?
- We can identify special cases to be exploited by algorithms.

Why are we Interested in Complexity Studies?

Benefits of complexity studies:

- We know how to design algorithms:
  - If a problem is undecidable, any terminating algorithm must be wrong. Similarly: if a problem is $\mathbb{NP}$-*complete*, it is not a good idea to design a decision procedure that runs in polynomial time.
  - If the complexity of a problem is not known, at which runtime should we aim? $\mathbb{P}$? $\mathbb{EXPTIME}$?
- We can identify special cases to be exploited by algorithms. Example: heuristics! (Most of them exploit special cases that can be decided in $\mathbb{P}$.)

Why are we Interested in Complexity Studies?

Benefits of complexity studies:

- We know how to design algorithms:
  - If a problem is undecidable, any terminating algorithm must be wrong. Similarly: if a problem is $\mathbb{NP}$-*complete*, it is not a good idea to design a decision procedure that runs in polynomial time.
  - If the complexity of a problem is not known, at which runtime should we aim? $\mathbb{P}$? $\mathbb{EXPTIME}$?
- We can identify special cases to be exploited by algorithms. Example: heuristics! (Most of them exploit special cases that can be decided in $\mathbb{P}$.)
- Insights may also allow for compilation techniques.

Why are we Interested in Complexity Studies?

Benefits of complexity studies:

- We know how to design algorithms:
  - If a problem is undecidable, any terminating algorithm must be wrong. Similarly: if a problem is $\mathbb{NP}$-*complete*, it is not a good idea to design a decision procedure that runs in polynomial time.
  - If the complexity of a problem is not known, at which runtime should we aim? $\mathbb{P}$? $\mathbb{EXPTIME}$?
- We can identify special cases to be exploited by algorithms. Example: heuristics! (Most of them exploit special cases that can be decided in $\mathbb{P}$.)
- Insights may also allow for compilation techniques.
- Last, but *not-at-all least*: they help understanding the problem! (Understanding the problem should always be the first step.)

XTIME, XSPACE, NXTIME, NXSPACE, with $X \in \{\mathbb{P}, \mathbb{EXP}\}$

Informally, we can state the following definitions for decision problems:

- A problem can be decided in $\mathbb{P}$ (also: $\mathbb{PTIME}$) if there is an algorithm that requires only polynomial time. (Similar for higher classes, such as $\mathbb{EXPTIME}$.

$X\mathbb{TIME}$, $X\mathbb{SPACE}$, $\mathbb{N}X\mathbb{TIME}$, $\mathbb{N}X\mathbb{SPACE}$, with $X \in \{\mathbb{P}, \mathbb{EXP}\}$

Informally, we can state the following definitions for decision problems:

- A problem can be decided in $\mathbb{P}$ (also: $\mathbb{PTIME}$) if there is an algorithm that requires only polynomial time. (Similar for higher classes, such as $\mathbb{EXPTIME}$.
- A problem can be decided in $\mathbb{NP}$ (also: $\mathbb{NPTIME}$) if there is a non-deterministic algorithm that requires only polynomial time. (Similar for higher classes, such as $\mathbb{NEXPTIME}$.

XTIME, XSPACE, NXTIME, NXSPACE, with $X \in \{\mathbb{P}, \mathbb{EXP}\}$

Informally, we can state the following definitions for decision problems:

- A problem can be decided in $\mathbb{P}$ (also: $\mathbb{PTIME}$) if there is an algorithm that requires only polynomial time. (Similar for higher classes, such as $\mathbb{EXPTIME}$.
- A problem can be decided in $\mathbb{NP}$ (also: $\mathbb{NPTIME}$) if there is a non-deterministic algorithm that requires only polynomial time. (Similar for higher classes, such as $\mathbb{NEXPTIME}$.
- A problem can be decided in $\mathbb{PSPACE}$ if there is an algorithm that requires only polynomial space. (Similar for higher classes, such as $\mathbb{EXPSPACE}$.

XTIME, XSPACE, NXTIME, NXSPACE, with $X \in \{\mathbb{P}, \mathbb{EXP}\}$

Informally, we can state the following definitions for decision problems:

- A problem can be decided in $\mathbb{P}$ (also: $\mathbb{PTIME}$) if there is an algorithm that requires only polynomial time. (Similar for higher classes, such as $\mathbb{EXPTIME}$.
- A problem can be decided in $\mathbb{NP}$ (also: $\mathbb{NPTIME}$) if there is a non-deterministic algorithm that requires only polynomial time. (Similar for higher classes, such as $\mathbb{NEXPTIME}$.
- A problem can be decided in $\mathbb{PSPACE}$ if there is an algorithm that requires only polynomial space. (Similar for higher classes, such as $\mathbb{EXPSPACE}$.
- Note: $\mathbb{PSPACE} = \mathbb{NPSPACE}$ (holds also for higher classes).

Problem Definition

- Decision problem: given a task network *tn* and an HTN planning problem $\mathcal{P}$, is P a solution for $\mathcal{P}$?

Problem Definition

- Decision problem: given a task network *tn* and an HTN planning problem $\mathcal{P}$, is P a solution for $\mathcal{P}$?
- What do we need to verify?

Problem Definition

- Decision problem: given a task network *tn* and an HTN planning problem $\mathcal{P}$, is P a solution for $\mathcal{P}$?
- What do we need to verify?
  - *tn* is a refinement of the initial task (network).

Problem Definition

- Decision problem: given a task network *tn* and an HTN planning problem $\mathcal{P}$, is P a solution for $\mathcal{P}$?
- What do we need to verify?
  - *tn* is a refinement of the initial task (network).
  - *tn* is executable.

Problem Definition

- Decision problem: given a task network *tn* and an HTN planning problem $\mathcal{P}$, is P a solution for $\mathcal{P}$?
- What do we need to verify?
  - *tn* is a refinement of the initial task (network).
  - *tn* is executable. According to which definition?

Problem Definition

- Decision problem: given a task network *tn* and an HTN planning problem $\mathcal{P}$, is P a solution for $\mathcal{P}$?
- What do we need to verify?
  - *tn* is a refinement of the initial task (network).
  - *tn* is executable. According to which definition?
    - All linearizations of P are executable.

Problem Definition

- Decision problem: given a task network *tn* and an HTN planning problem $\mathcal{P}$, is P a solution for $\mathcal{P}$?
- What do we need to verify?
  - *tn* is a refinement of the initial task (network).
  - *tn* is executable. According to which definition?
    - All linearizations of P are executable.
    - There exist a linearization of P.

Problem Definition

- Decision problem: given a task network *tn* and an HTN planning problem $\mathcal{P}$, is P a solution for $\mathcal{P}$?
- What do we need to verify?
  - *tn* is a refinement of the initial task (network).
  - *tn* is executable. According to which definition?
    - All linearizations of P are executable.
    - There exist a linearization of P.
- We might consider special cases:

Problem Definition

- Decision problem: given a task network *tn* and an HTN planning problem $\mathcal{P}$, is P a solution for $\mathcal{P}$?
- What do we need to verify?
  - *tn* is a refinement of the initial task (network).
  - *tn* is executable. According to which definition?
    - All linearizations of P are executable.
    - There exist a linearization of P.
- We might consider special cases:
  - Task insertion.

Problem Definition

- Decision problem: given a task network *tn* and an HTN planning problem $\mathcal{P}$, is P a solution for $\mathcal{P}$?
- What do we need to verify?
  - *tn* is a refinement of the initial task (network).
  - *tn* is executable. According to which definition?
    - All linearizations of P are executable.
    - There exist a linearization of P.
- We might consider special cases:
  - Task insertion.
  - Empty or primitive initial task network.

Problem Definition

- Decision problem: given a task network *tn* and an HTN planning problem $\mathcal{P}$, is P a solution for $\mathcal{P}$?
- What do we need to verify?
  - *tn* is a refinement of the initial task (network).
  - *tn* is executable. According to which definition?
    - All linearizations of P are executable.
    - There exist a linearization of P.
- We might consider special cases:
  - Task insertion.
  - Empty or primitive initial task network.
  - Totally ordered methods/initial task network.

Verification of Classical Solutions

Theorem

Let $\mathcal{P}$ be a STRIPS planning problem and $\bar{a}$ an action sequence. Then, deciding whether $\bar{a}$ is a solution for $\mathcal{P}$ is

Verification of Classical Solutions

Theorem

Let $\mathcal{P}$ be a STRIPS planning problem and $\bar{a}$ an action sequence. Then, deciding whether $\bar{a}$ is a solution for $\mathcal{P}$ is in $\mathbb{P}$.

Verification of Classical Solutions

Theorem

Let $\mathcal{P}$ be a STRIPS planning problem and $\bar{a}$ an action sequence. Then, deciding whether $\bar{a}$ is a solution for $\mathcal{P}$ is in $\mathbb{P}$.

Proof:
Execute the plan and check whether every action can be applied in the respective state and whether a goal is produced. ($\rightarrow$ Linear effort.)

Verification of Partial Orders

Theorem

Let *tn* be a primitive task network, i.e., a partially ordered set of (labeled) actions.
Then, deciding whether *tn* has an executable linearization is

Verification of Partial Orders

Theorem

Let *tn* be a primitive task network, i.e., a partially ordered set of (labeled) actions.
Then, deciding whether *tn* has an executable linearization is $\mathbb{NP}$-complete.

Verification of Partial Orders

Theorem

Let *tn* be a primitive task network, i.e., a partially ordered set of (labeled) actions.
Then, deciding whether *tn* has an executable linearization is $\mathbb{NP}$-complete.

Proof:

Verification of Partial Orders

Theorem

Let *tn* be a primitive task network, i.e., a partially ordered set of (labeled) actions.
Then, deciding whether *tn* has an executable linearization is $\mathbb{NP}$-complete.

Proof:

Membership: guess and verify.

Verification of Partial Orders

Theorem

Let *tn* be a primitive task network, i.e., a partially ordered set of (labeled) actions.
Then, deciding whether *tn* has an executable linearization is $\mathbb{NP}$-complete.

Proof:
Membership: guess and verify.
Hardness: Reduction from CNF Sat (proof idea via black board).

Verification of HTN Plans

Corollary

Let *tn* be a primitive task network and $\mathcal{P}$ an HTN or TIHTN planning problem. Then, deciding whether *tn* is a solution is

Verification of HTN Plans

Corollary

Let *tn* be a primitive task network and $\mathcal{P}$ an HTN or TIHTN planning problem. Then, deciding whether *tn* is a solution is $\mathbb{NP}$-hard.

Verification of HTN Plans

Corollary

Let *tn* be a primitive task network and $\mathcal{P}$ an HTN or TIHTN planning problem. Then, deciding whether *tn* is a solution is $\mathbb{NP}$-hard.

- What about membership?

Verification of HTN Plans

Corollary

Let *tn* be a primitive task network and $\mathcal{P}$ an HTN or TIHTN planning problem. Then, deciding whether *tn* is a solution is $\mathbb{NP}$-hard.

- What about membership? Easy if methods are non-empty, tricky (but possible) otherwise.

Verification of HTN Plans

Corollary

Let *tn* be a primitive task network and $\mathcal{P}$ an HTN or TIHTN planning problem. Then, deciding whether *tn* is a solution is $\mathbb{NP}$-hard.

- What about membership? Easy if methods are non-empty, tricky (but possible) otherwise.
- What if we have the witness for executability given?
  $\rightarrow$ Then, checking executability is in $\mathbb{P}$, right?

Verification of HTN Plans

Corollary

Let *tn* be a primitive task network and $\mathcal{P}$ an HTN or TIHTN planning problem. Then, deciding whether *tn* is a solution is $\mathbb{NP}$-hard.

- What about membership? Easy if methods are non-empty, tricky (but possible) otherwise.
- What if we have the witness for executability given?
  $\rightarrow$ Then, checking executability is in $\mathbb{P}$, right?
  - No! This is only the case if we know the *labels/task ids* rather than just the actions.

Verification of HTN Plans

Corollary

Let *tn* be a primitive task network and $\mathcal{P}$ an HTN or TIHTN planning problem. Then, deciding whether *tn* is a solution is $\mathbb{NP}$-hard.

- What about membership? Easy if methods are non-empty, tricky (but possible) otherwise.
- What if we have the witness for executability given?
  $\rightarrow$ Then, checking executability is in $\mathbb{P}$, right?
  - No! This is only the case if we know the *labels/task ids* rather than just the actions.
  - Otherwise, we have to check whether there is a refinement of the task network's ordering constraint leading to the witness, which is again $\mathbb{NP}$-hard.

Verification of HTN Plans – for Altered Executability Semantics

- Let's consider a practically more Useful definition of executability.
- Let's require a primitive task network to be executable if and only if *every linearization* is executable.

Verification of HTN Plans – for Altered Executability Semantics

- Let's consider a practically more Useful definition of executability.
- Let's require a primitive task network to be executable if and only if *every linearization* is executable.

Theorem

Deciding whether a (primitive) task network is executable (in the sense given above) is

Verification of HTN Plans – for Altered Executability Semantics

- Let's consider a practically more Useful definition of executability.
- Let's require a primitive task network to be executable if and only if *every linearization* is executable.

Theorem

Deciding whether a (primitive) task network is executable (in the sense given above) is in $\mathbb{P}$.

Verification of HTN Plans – for Altered Executability Semantics

- Let's consider a practically more Useful definition of executability.
- Let's require a primitive task network to be executable if and only if *every linearization* is executable.

Theorem

Deciding whether a (primitive) task network is executable (in the sense given above) is in $\mathbb{P}$.

Proof:
Black board.

Verification of HTN Plans – for Altered Executability Semantics

- So, given a task network with *all executability semantics*, plan verification can be decided in $\mathbb{P}$, right?
- No! We still need to check the refinement criterion. This is $\mathbb{NP}$-hard, however. (Reduction from Vertex Cover, maybe later.)

Verification of HTN Plans – for Altered Executability Semantics

- So, given a task network with *all executability semantics*, plan verification can be decided in $\mathbb{P}$, right?
- No! We still need to check the refinement criterion. This is $\mathbb{NP}$-hard, however. (Reduction from Vertex Cover, maybe later.)

Theorem

Deciding whether a (primitive) task network is a solution is, even for *all executability semantics*, $\mathbb{NP}$-complete.

Summary

- So far, we studied the computational complexity of the *plan verification problem*.

Summary

- So far, we studied the computational complexity of the *plan verification problem*.
- It ranges from $\mathbb{P}$ to $\mathbb{NP}$-*complete*.

Summary

- So far, we studied the computational complexity of the *plan verification problem*.
- It ranges from $\mathbb{P}$ to $\mathbb{NP}$-*complete*.
- Verifying total-order plans is much easier than verifying partially ordered plans.

Summary

- So far, we studied the computational complexity of the *plan verification problem*.
- It ranges from $\mathbb{P}$ to $\mathbb{NP}$-*complete*.
- Verifying total-order plans is much easier than verifying partially ordered plans.
- The hardness of verifying partially ordered plans depends on whether an executable linearization needs to exist or whether all of them need to be executable.

Summary

- So far, we studied the computational complexity of the *plan verification problem*.
- It ranges from $\mathbb{P}$ to $\mathbb{NP}$-*complete*.
- Verifying total-order plans is much easier than verifying partially ordered plans.
- The hardness of verifying partially ordered plans depends on whether an executable linearization needs to exist or whether all of them need to be executable.
- Verifying hierarchical plans is often harder, because we also need to check the refinement criterion.

Summary

- So far, we studied the computational complexity of the *plan verification problem*.
- It ranges from $\mathbb{P}$ to $\mathbb{NP}$-*complete*.
- Verifying total-order plans is much easier than verifying partially ordered plans.
- The hardness of verifying partially ordered plans depends on whether an executable linearization needs to exist or whether all of them need to be executable.
- Verifying hierarchical plans is often harder, because we also need to check the refinement criterion.
- Complexity results give raise to specialized algorithms, to heuristics, and to translations to other problem classes.