Lecture *Hierarchical Planning*

Chapter:
Heuristics for (Non-Hierarchical) Planning Problems

Dr. Pascal Bercher

Institute of Artificial Intelligence,
Ulm University, Germany

Winter Term 2018/2019

(Compiled on: November 28, 2023)

ulm university universität uulm

Overview:

Overview

What's a heuristic in planning?

- The same as in search! (See respective lecture.)

So, what's covered here?

- We discuss *planning-specific* problem relaxations.
- We investigate some of the easiest/most fundamental heuristics for classical and POCL planning.

What are Heuristics?

Recap: How to come up with heuristics in a *domain-independent* way?

What are Heuristics?

Recap: How to come up with heuristics in a *domain-independent* way?

- Perform a *problem relaxation*.

What are Heuristics?

Recap: How to come up with heuristics in a *domain-independent* way?

- Perform a *problem relaxation*.
- Solve the relaxed problem.

What are Heuristics?

Recap: How to come up with heuristics in a *domain-independent* way?

- Perform a *problem relaxation*.
- Solve the relaxed problem.
- Use the relaxation's solution cost as approximation (i.e., heuristic) of the actual (original, non-relaxed) problem.

What are Heuristics?

Recap: How to come up with heuristics in a *domain-independent* way?

- Perform a *problem relaxation*.
- Solve the relaxed problem.
- Use the relaxation's solution cost as approximation (i.e., heuristic) of the actual (original, non-relaxed) problem.

What is a problem relaxation?

- Sometimes *special cases of planning problems* (e.g., ignore all delete lists).

What are Heuristics?

Recap: How to come up with heuristics in a *domain-independent* way?

- Perform a *problem relaxation*.
- Solve the relaxed problem.
- Use the relaxation's solution cost as approximation (i.e., heuristic) of the actual (original, non-relaxed) problem.

What is a problem relaxation?

- Sometimes *special cases of planning problems* (e.g., ignore all delete lists).
- Sometimes *specialized calculations* (that might, however, still be interpreted as special cases of standard planning problems).

What are Heuristics?

Recap: How to come up with heuristics in a *domain-independent* way?

- Perform a *problem relaxation*.
- Solve the relaxed problem.
- Use the relaxation's solution cost as approximation (i.e., heuristic) of the actual (original, non-relaxed) problem.

What is a problem relaxation?

- Sometimes *special cases of planning problems* (e.g., ignore all delete lists).
- Sometimes *specialized calculations* (that might, however, still be interpreted as special cases of standard planning problems).
- They should be *easier* than the original problem: either in terms of computational complexity or in the problem size.

What are Heuristics?

Recap: How to come up with heuristics in a *domain-independent* way?

- Perform a *problem relaxation*.
- Solve the relaxed problem.
- Use the relaxation's solution cost as approximation (i.e., heuristic) of the actual (original, non-relaxed) problem.

What is a problem relaxation?

- Sometimes *special cases of planning problems* (e.g., ignore all delete lists).
- Sometimes *specialized calculations* (that might, however, still be interpreted as special cases of standard planning problems).
- They should be *easier* than the original problem: either in terms of computational complexity or in the problem size.
- Ordinarily *safe* (cf. search: unsolvable in relaxation implies unsolvable in original).

Problem Relaxations for Planning

How to relax a STRIPS planning problem?

- Ignore the delete effects. $\rightarrow$ *Delete-relaxation heuristics*.

Problem Relaxations for Planning

How to relax a STRIPS planning problem?

- Ignore the delete effects. $\rightarrow$ *Delete-relaxation heuristics*.
- Ignore an entire set of state variables. $\rightarrow$ *Abstraction heuristics*.

Problem Relaxations for Planning

How to relax a STRIPS planning problem?

- Ignore the delete effects. $\rightarrow$ *Delete-relaxation heuristics*.
- Ignore an entire set of state variables. $\rightarrow$ *Abstraction heuristics*.
- Compute and exploit state variables (or actions) that have to be part of (or are contained in) any solution at some point. $\rightarrow$ *Landmark-based heuristics*.

Problem Relaxations for Planning

How to relax a STRIPS planning problem?

- Ignore the delete effects. $\rightarrow$ *Delete-relaxation heuristics*.
- Ignore an entire set of state variables. $\rightarrow$ *Abstraction heuristics*.
- Compute and exploit state variables (or actions) that have to be part of (or are contained in) any solution at some point. $\rightarrow$ *Landmark-based heuristics*.
- Estimate plan length by making relaxed assumptions on when a set of variables is regarded reachable. $\rightarrow$ *Critical path heuristics*.

Problem Relaxations for Planning

How to relax a STRIPS planning problem?

- Ignore the delete effects. $\rightarrow$ *Delete-relaxation heuristics*.
- Ignore an entire set of state variables. $\rightarrow$ *Abstraction heuristics*.
- Compute and exploit state variables (or actions) that have to be part of (or are contained in) any solution at some point. $\rightarrow$ *Landmark-based heuristics*.
- Estimate plan length by making relaxed assumptions on when a set of variables is regarded reachable. $\rightarrow$ *Critical path heuristics*.
- And many more!

Problem Relaxations for Planning

How to relax a STRIPS planning problem?

- Ignore the delete effects. $\rightarrow$ *Delete-relaxation heuristics*.
- Ignore an entire set of state variables. $\rightarrow$ *Abstraction heuristics*.
- Compute and exploit state variables (or actions) that have to be part of (or are contained in) any solution at some point. $\rightarrow$ *Landmark-based heuristics*.
- Estimate plan length by making relaxed assumptions on when a set of variables is regarded reachable. $\rightarrow$ *Critical path heuristics*.
- And many more!

Further reading: Malte Helmert and Carmel Domshlak. "Landmarks, Critical Paths and Abstractions: What's the Difference Anyway?" In: *Proc. of the 19th Int. Conf. on Automated Planning and Scheduling (ICAPS 2009)*. AAAI Press, 2009, pp. 162–169

Motivation

- The planning graph is a relaxed representation of the state and action space.

Motivation

- The planning graph is a relaxed representation of the state and action space.
- It exists with varying degrees of constraints (mutexes, representing which state variables may be true at the same time) making it more or less informed.

Motivation

- The planning graph is a relaxed representation of the state and action space.
- It exists with varying degrees of constraints (mutexes, representing which state variables may be true at the same time) making it more or less informed.
- Here, we only cover the most relaxed form, which can be computed in polynomial time.

Motivation

- The planning graph is a relaxed representation of the state and action space.
- It exists with varying degrees of constraints (mutexes, representing which state variables may be true at the same time) making it more or less informed.
- Here, we only cover the most relaxed form, which can be computed in polynomial time.
- Its main purpose today:

Motivation

- The planning graph is a relaxed representation of the state and action space.
- It exists with varying degrees of constraints (mutexes, representing which state variables may be true at the same time) making it more or less informed.
- Here, we only cover the most relaxed form, which can be computed in polynomial time.
- Its main purpose today:
  - Use it to ground a domain (covered later).

Motivation

- The planning graph is a relaxed representation of the state and action space.
- It exists with varying degrees of constraints (mutexes, representing which state variables may be true at the same time) making it more or less informed.
- Here, we only cover the most relaxed form, which can be computed in polynomial time.
- Its main purpose today:
  - Use it to ground a domain (covered later).
  - Used for relaxed reachability analysis ("Given a state *s*, is there (maybe) a course of actions that enables the application of action *a* afterwards?")

Motivation

- The planning graph is a relaxed representation of the state and action space.
- It exists with varying degrees of constraints (mutexes, representing which state variables may be true at the same time) making it more or less informed.
- Here, we only cover the most relaxed form, which can be computed in polynomial time.
- Its main purpose today:
  - Use it to ground a domain (covered later).
  - Used for relaxed reachability analysis (“Given a state *s*, is there (maybe) a course of actions that enables the application of action *a* afterwards?”)
  - Basis for heuristics. $\rightarrow$ Both for classical and POCL planning!

Historical Remarks

The *planning graph* is a data structure that was invented for a planning system called *GraphPlan*.

Historical Remarks

The *planning graph* is a data structure that was invented for a planning system called *GraphPlan*.

That planning system is *not* relevant for this course.

Historical Remarks

The *planning graph* is a data structure that was invented for a planning system called *GraphPlan*.

That planning system is *not* relevant for this course.

Note:
Do not think of a pink elephant right now!

Historical Remarks

The *planning graph* is a data structure that was invented for a planning system called *GraphPlan*.

That planning system is *not* relevant for this course.

Note:
Do not think of a pink elephant right now!
Rephrased: Please *do not* confuse *GraphPlan* with the *planning graph*!
The first is a *planning system* – the latter a *data structure*.

Historical Remarks

The *planning graph* is a data structure that was invented for a planning system called *GraphPlan*.

That planning system is *not* relevant for this course.

Note:
Do not think of a pink elephant right now!
Rephrased: Please *do not* confuse *GraphPlan* with the *planning graph*!
The first is a *planning system* – the latter a *data structure*.

Further reading: Avrim L. Blum and Merrick L. Furst. "Fast Planning Through Planning Graph Analysis". In: *Artificial Intelligence* 90 (1997), pp. 281–300. DOI: 10.1016/S0004-3702(96)00047-1

Definitions, Delete Relaxation

Definition (Delete-free and -relaxed Planning Problems)

Let $\mathcal{P} = \langle V, A, s_I, g \rangle$ be a STRIPS planning problem.

- It is called delete-free if for all $a \in A$, $del(a) = \emptyset$.

Definitions, Delete Relaxation

Definition (Delete-free and -relaxed Planning Problems)

Let $\mathcal{P} = \langle V, A, s_I, g \rangle$ be a STRIPS planning problem.

- It is called delete-free if for all $a \in A$, $del(a) = \emptyset$.
- Its delete-relaxation is the (delete-free) problem $\langle V, A', s_I, g \rangle$, where $A' = \{(pre, add, \emptyset, c) \mid (pre, add, del, c) \in A\}$.

Definitions, Delete Relaxation

Definition (Delete-free and -relaxed Planning Problems)

Let $\mathcal{P} = \langle V, A, s_I, g\rangle$ be a STRIPS planning problem.

- It is called delete-free if for all $a \in A$, $del(a) = \emptyset$.
- Its delete-relaxation is the (delete-free) problem $\langle V, A', s_I, g\rangle$, where $A' = \{(pre, add, \emptyset, c) \mid (pre, add, del, c) \in A\}$.

$\rightarrow$ $\mathcal{P}^+$ refers to the delete-relaxation of $\mathcal{P}$ and

Definitions, Delete Relaxation

Definition (Delete-free and -relaxed Planning Problems)

Let $\mathcal{P} = \langle V, A, s_I, g \rangle$ be a STRIPS planning problem.

- It is called delete-free if for all $a \in A$, $del(a) = \emptyset$.
- Its delete-relaxation is the (delete-free) problem $\langle V, A', s_I, g \rangle$, where $A' = \{(pre, add, \emptyset, c) \mid (pre, add, del, c) \in A\}$.

$\rightarrow$ $\mathcal{P}^+$ refers to the delete-relaxation of $\mathcal{P}$ and

$\rightarrow$ h^+ refers to the perfect heuristic (h^*) for $\mathcal{P}^+$.

Definitions, Delete Relaxation

What's the core idea behind delete relaxation?

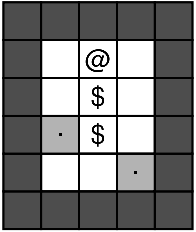
Definitions, Delete Relaxation

What's the core idea behind delete relaxation?
$\rightarrow$ What's true once stays true!

Definitions, Delete Relaxation

What's the core idea behind delete relaxation?
$\rightarrow$ What's true once stays true!

Consider Sokoban:

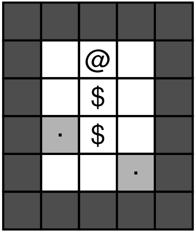


@ = the figure $ = a crate   · = a goal position

Definitions, Delete Relaxation

What's the core idea behind delete relaxation?
$\rightarrow$ What's true once stays true!

Consider Sokoban: ...after moving left, down, right...

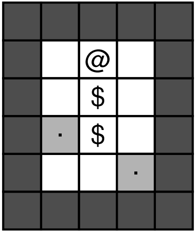


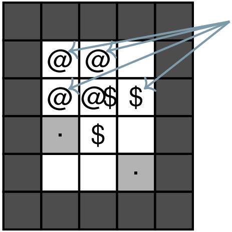
@ = the figure $ = a crate   · = a goal position

Definitions, Delete Relaxation

What's the core idea behind delete relaxation?
$\rightarrow$ What's true once stays true!

Consider Sokoban: ...after moving left, down, right...





These positions are also *free!* (Since they were free before or have become so.)

@ = the figure $\quad$ \$ = a crate $\quad$ ⋅ = a goal position

Definitions, Relaxed Planning Graph

Definition (Relaxed Planning Graph)

Let $\langle V, A, s_I, g\rangle$ be a (delete-free) planning problem.

Then, a *relaxed planning graph (rPG)* is a graph $\langle \bar{V}, \bar{A}\rangle$ consisting of:

- $\bar{V} = V^0 \dots V^n$, $V^i \subseteq V$, $0 \leq i \leq n$, a sequence of *variable layers*.

Definitions, Relaxed Planning Graph

Definition (Relaxed Planning Graph)

Let $\langle V, A, s_I, g \rangle$ be a (delete-free) planning problem.

Then, a *relaxed planning graph (rPG)* is a graph $\langle \bar{V}, \bar{A} \rangle$ consisting of:

- $\bar{V} = V^0 \dots V^n$, $V^i \subseteq V$, $0 \leq i \leq n$, a sequence of *variable layers*.
- $\bar{A} = A^1 \dots A^n$, $A^i \subseteq A$, $1 \leq i \leq n$, a sequence of *action layers*.

Definitions, Relaxed Planning Graph

Definition (Relaxed Planning Graph)

Let $\langle V, A, s_I, g\rangle$ be a (delete-free) planning problem.

Then, a *relaxed planning graph (rPG)* is a graph $\langle \bar{V}, \bar{A}\rangle$ consisting of:

- $\bar{V} = V^0 \ldots V^n$, $V^i \subseteq V$, $0 \leq i \leq n$, a sequence of *variable layers*.
- $\bar{A} = A^1 \ldots A^n$, $A^i \subseteq A$, $1 \leq i \leq n$, a sequence of *action layers*.
- $V^0 = s_I$.

Definitions, Relaxed Planning Graph

Definition (Relaxed Planning Graph)

Let $\langle V, A, s_I, g\rangle$ be a (delete-free) planning problem.

Then, a *relaxed planning graph (rPG)* is a graph $\langle \bar{V}, \bar{A}\rangle$ consisting of:

- $\bar{V} = V^0 \dots V^n$, $V^i \subseteq V$, $0 \leq i \leq n$, a sequence of *variable layers*.
- $\bar{A} = A^1 \dots A^n$, $A^i \subseteq A$, $1 \leq i \leq n$, a sequence of *action layers*.
- $V^0 = s_I$.
- $A^i = \{a \in A \mid pre(a) \subseteq V^{i-1}\}$, $1 \leq i \leq n$.

Definitions, Relaxed Planning Graph

Definition (Relaxed Planning Graph)

Let $\langle V, A, s_I, g\rangle$ be a (delete-free) planning problem.

Then, a *relaxed planning graph (rPG)* is a graph $\langle \bar{V}, \bar{A}\rangle$ consisting of:

- $\bar{V} = V^0 \dots V^n$, $V^i \subseteq V$, $0 \leq i \leq n$, a sequence of *variable layers*.
- $\bar{A} = A^1 \dots A^n$, $A^i \subseteq A$, $1 \leq i \leq n$, a sequence of *action layers*.
- $V^0 = s_I$.
- $A^i = \{a \in A \mid pre(a) \subseteq V^{i-1}\}$, $1 \leq i \leq n$.
- $V^i = V^{i-1} \cup \bigcup_{a \in A^i} add(a)$, $1 \leq i \leq n$.

Definitions, Relaxed Planning Graph

Definition (Relaxed Planning Graph)

Let $\langle V, A, s_I, g\rangle$ be a (delete-free) planning problem.

Then, a *relaxed planning graph (rPG)* is a graph $\langle \bar{V}, \bar{A}\rangle$ consisting of:

- $\bar{V} = V^0 \dots V^n$, $V^i \subseteq V$, $0 \leq i \leq n$, a sequence of *variable layers*.
- $\bar{A} = A^1 \dots A^n$, $A^i \subseteq A$, $1 \leq i \leq n$, a sequence of *action layers*.
- $V^0 = s_I$.
- $A^i = \{a \in A \mid pre(a) \subseteq V^{i-1}\}$, $1 \leq i \leq n$.
- $V^i = V^{i-1} \cup \bigcup_{a \in A^i} add(a)$, $1 \leq i \leq n$.
- Choose $n = i$, such that $V^{i-1} = V^i$ holds.

Definitions, Relaxed Planning Graph

Definition (Relaxed Planning Graph)

Let $\langle V, A, s_I, g\rangle$ be a (delete-free) planning problem.

Then, a *relaxed planning graph (rPG)* is a graph $\langle \bar{V}, \bar{A}\rangle$ consisting of:

- $\bar{V} = V^0 \dots V^n$, $V^i \subseteq V$, $0 \leq i \leq n$, a sequence of *variable layers*.
- $\bar{A} = A^1 \dots A^n$, $A^i \subseteq A$, $1 \leq i \leq n$, a sequence of *action layers*.
- $V^0 = s_I$.
- $A^i = \{a \in A \mid pre(a) \subseteq V^{i-1}\}$, $1 \leq i \leq n$.
- $V^i = V^{i-1} \cup \bigcup_{a \in A^i} add(a)$, $1 \leq i \leq n$.
- Choose $n = i$, such that $V^{i-1} = V^i$ holds.

Questions:

- Why is “delete-free” in the problem description put in parentheses?
- Why is n chosen as is? Is there a bound on n?
- What happens if we choose $n = i$, such that $V^i = V^{i+1}$ holds?

Definitions, Relaxed Planning Graph, cont'd

We can extend this definition of rPGs to add information about reachability, i.e.,

Definitions, Relaxed Planning Graph, cont'd

We can extend this definition of rPGs to add information about reachability, i.e.,

- Which variable(s) enable which action precondition?

Definitions, Relaxed Planning Graph, cont'd

We can extend this definition of rPGs to add information about reachability, i.e.,

- Which variable(s) enable which action precondition?
- Which variable(s) get added by which action(s)?

Definitions, Relaxed Planning Graph, cont'd

We can extend this definition of rPGs to add information about reachability, i.e.,

- Which variable(s) enable which action precondition?
- Which variable(s) get added by which action(s)?
- How variables "remain valid" (due to the absence of deletions). (That is, all variables v in V^i and V^{i+1} share an edge)

Definitions, Relaxed Planning Graph, cont'd

We can extend this definition of rPGs to add information about reachability, i.e.,

- Which variable(s) enable which action precondition?
- Which variable(s) get added by which action(s)?
- How variables "remain valid" (due to the absence of deletions). (That is, all variables v in V^i and V^{i+1} share an edge)

Formal definition thereof:
Exercise!

Example – Exercise!

Draw the rPG with edges for the Cranes in the Harbor domain.

s_I: $\{CrateAtLoc1, TruckAtLoc2\}$ g: $\{CrateInTruck, TruckAtLoc2\}$

take
- pre: {CrateAtLoc1}
- add: {HoldCrate}
- del: {CrateAtLoc1}

put
- pre: {HoldCrate}
- add: {CrateAtLoc1}
- del: {HoldCrate}

moveLeft
- pre: {TruckAtLoc2}
- add: {TruckAtLoc1}
- del: {TruckAtLoc2}

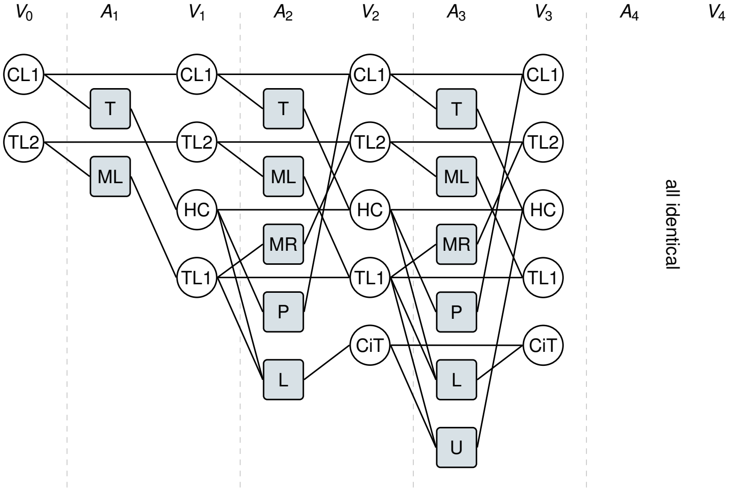
moveRight
- pre: {TruckAtLoc1}
- add: {TruckAtLoc2}
- del: {TruckAtLoc1}

load
- pre: {HoldCrate, TruckAtLoc1}
- add: {CrateInTruck}
- del: {HoldCrate}

unload
- pre: {CrateInTruck, TruckAtLoc1}
- add: {HoldCrate}
- del: {CrateInTruck}

Example – Exercise! cont'd

Solution:



h^{max} for Classical Planning

Let $\mathcal{P} = \langle V, A, s_I, g \rangle$ be a STRIPS planning problem and $\mathcal{G} = \langle \bar{V}, \bar{A} \rangle$ its rPG.

- $h^{max}(s)$ returns the first layer number in which all goal variables hold. Meaning: Number of action layers required in $\mathcal{P}^+$ to make the hardest variable in g true (starting in some $s \in S$, e.g., s_I).

h^{max} for Classical Planning

Let $\mathcal{P} = \langle V, A, s_I, g \rangle$ be a STRIPS planning problem and $\mathcal{G} = \langle \bar{V}, \bar{A} \rangle$ its rPG.

- $h^{max}(s)$ returns the first layer number in which all goal variables hold. Meaning: Number of action layers required in $\mathcal{P}^+$ to make the hardest variable in g true (starting in some $s \in S$, e.g., s_I).
- Formally, h^{max} can be calculated as follows:

h^{max} for Classical Planning

Let $\mathcal{P} = \langle V, A, s_I, g \rangle$ be a STRIPS planning problem and $\mathcal{G} = \langle \bar{V}, \bar{A} \rangle$ its rPG.

- $h^{max}(s)$ returns the first layer number in which all goal variables hold. Meaning: Number of action layers required in $\mathcal{P}^+$ to make the hardest variable in g true (starting in some $s \in S$, e.g., s_I).
- Formally, h^{max} can be calculated as follows:

action vertex The cost of an action vertex $a \in A^i$ is 1 plus the maximum of the predecessor vertex costs.

h^{max} for Classical Planning

Let $\mathcal{P} = \langle V, A, s_I, g \rangle$ be a STRIPS planning problem and $\mathcal{G} = \langle \bar{V}, \bar{A} \rangle$ its rPG.

- $h^{max}(s)$ returns the first layer number in which all goal variables hold. Meaning: Number of action layers required in $\mathcal{P}^+$ to make the hardest variable in g true (starting in some $s \in S$, e.g., s_I).
- Formally, h^{max} can be calculated as follows:

action vertex The cost of an action vertex $a \in A^i$ is 1 plus the maximum of the predecessor vertex costs.

variable vertex
- The cost of a variable vertex v is 0 if $v \in V^0$.

h^{max} for Classical Planning

Let $\mathcal{P} = \langle V, A, s_I, g \rangle$ be a STRIPS planning problem and $\mathcal{G} = \langle \bar{V}, \bar{A} \rangle$ its rPG.

- $h^{max}(s)$ returns the first layer number in which all goal variables hold. Meaning: Number of action layers required in $\mathcal{P}^+$ to make the hardest variable in g true (starting in some $s \in S$, e.g., s_I).
- Formally, h^{max} can be calculated as follows:

action vertex The cost of an action vertex $a \in A^i$ is 1 plus the maximum of the predecessor vertex costs.

variable vertex
- The cost of a variable vertex v is 0 if $v \in V^0$.
- For all $v \in V^i$, $i > 0$, the cost of v equals the minimum cost of all predecessor vertices (these might be either action or variable vertices).

h^{max} for Classical Planning

Let $\mathcal{P} = \langle V, A, s_I, g \rangle$ be a STRIPS planning problem and $\mathcal{G} = \langle \bar{V}, \bar{A} \rangle$ its rPG.

- $h^{max}(s)$ returns the first layer number in which all goal variables hold. Meaning: Number of action layers required in $\mathcal{P}^+$ to make the hardest variable in g true (starting in some $s \in S$, e.g., s_I).
- Formally, h^{max} can be calculated as follows:

action vertex The cost of an action vertex $a \in A^i$ is 1 plus the maximum of the predecessor vertex costs.

variable vertex
- The cost of a variable vertex v is 0 if $v \in V^0$.
- For all $v \in V^i$, $i > 0$, the cost of v equals the minimum cost of all predecessor vertices (these might be either action or variable vertices).

vertex set For a set of state variables $\bar{v} \subseteq V$, the cost equals the most expensive variable in $\bar{v}$.

h^{max} for Classical Planning

Let $\mathcal{P} = \langle V, A, s_I, g \rangle$ be a STRIPS planning problem and $\mathcal{G} = \langle \bar{V}, \bar{A} \rangle$ its rPG.

- $h^{max}(s)$ returns the first layer number in which all goal variables hold. Meaning: Number of action layers required in $\mathcal{P}^+$ to make the hardest variable in g true (starting in some $s \in S$, e.g., s_I).
- Formally, h^{max} can be calculated as follows:

action vertex The cost of an action vertex $a \in A^i$ is 1 plus the maximum of the predecessor vertex costs.

variable vertex
- The cost of a variable vertex v is 0 if $v \in V^0$.
- For all $v \in V^i$, $i > 0$, the cost of v equals the minimum cost of all predecessor vertices (these might be either action or variable vertices).

vertex set For a set of state variables $\bar{v} \subseteq V$, the cost equals the most expensive variable in $\bar{v}$.

heuristic For a state $s \in S$, $h^{max}(s)$ equals the cost of g.

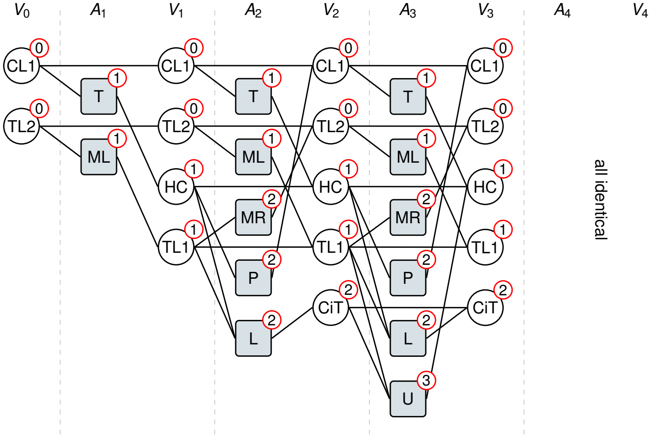
Example – Exercise!

Calculate h^{max} for the Cranes in the Harbor domain.

$$s_I = \{CrateAtLoc1, TruckAtLoc2\} \quad g = \{CrateInTruck, TruckAtLoc2\}$$

Example – Exercise!

Calculate h^{max} for the Cranes in the Harbor domain.



$s_I = \{CrateAtLoc1, TruckAtLoc2\}$ $g = \{CrateInTruck, TruckAtLoc2\}$

$h^{max}(s_I) = 2$ $h^*(s_I) = 4$ $h^*_{makespan}(s_I) = 3$

Admissibility

Is h^{max} admissible?

Admissibility

Is h^{max} admissible?

Yes. (trivial)

h^{add} for Classical Planning

Let $\mathcal{P} = \langle V, A, s_I, g \rangle$ be a STRIPS planning problem and $\mathcal{G} = \langle \bar{V}, \bar{A} \rangle$ its rPG.

- $h^{add}(s)$ calculates the cost reaching g from $s \in S$ via adding the costs of the actions' preconditions. Implicit assumption: "subgoal independence".

h^{add} for Classical Planning

Let $\mathcal{P} = \langle V, A, s_I, g \rangle$ be a STRIPS planning problem and $\mathcal{G} = \langle \bar{V}, \bar{A} \rangle$ its rPG.

- $h^{add}(s)$ calculates the cost reaching g from $s \in S$ via adding the costs of the actions' preconditions. Implicit assumption: "subgoal independence".
- Formally, h^{add} can be calculated as follows:

h^{add} for Classical Planning

Let $\mathcal{P} = \langle V, A, s_I, g \rangle$ be a STRIPS planning problem and $\mathcal{G} = \langle \bar{V}, \bar{A} \rangle$ its rPG.

- $h^{add}(s)$ calculates the cost reaching g from $s \in S$ via adding the costs of the actions' preconditions. Implicit assumption: "subgoal independence".
- Formally, h^{add} can be calculated as follows:

action vertex: The cost of an action vertex $a \in A^i$ is $c(a)$ plus the sum of the predecessor vertex costs.

h^{add} for Classical Planning

Let $\mathcal{P} = \langle V, A, s_I, g \rangle$ be a STRIPS planning problem and $\mathcal{G} = \langle \bar{V}, \bar{A} \rangle$ its rPG.

- $h^{add}(s)$ calculates the cost reaching g from $s \in S$ via adding the costs of the actions' preconditions. Implicit assumption: "subgoal independence".
- Formally, h^{add} can be calculated as follows:

action vertex: The cost of an action vertex $a \in A^i$ is $c(a)$ plus the sum of the predecessor vertex costs.

variable vertex:
- The cost of a variable vertex v is 0 if $v \in V^0$.

h^{add} for Classical Planning

Let $\mathcal{P} = \langle V, A, s_I, g \rangle$ be a STRIPS planning problem and $\mathcal{G} = \langle \bar{V}, \bar{A} \rangle$ its rPG.

- $h^{add}(s)$ calculates the cost reaching g from $s \in S$ via adding the costs of the actions' preconditions. Implicit assumption: "subgoal independence".
- Formally, h^{add} can be calculated as follows:

action vertex The cost of an action vertex $a \in A^i$ is $c(a)$ plus the sum of the predecessor vertex costs.

variable vertex
- The cost of a variable vertex v is 0 if $v \in V^0$.
- For all $v \in V^i$, $i > 0$, the cost of v equals the minimum cost of all predecessor vertices (these might be either action or variable vertices).

h^{add} for Classical Planning

Let $\mathcal{P} = \langle V, A, s_I, g \rangle$ be a STRIPS planning problem and $\mathcal{G} = \langle \bar{V}, \bar{A} \rangle$ its rPG.

- $h^{add}(s)$ calculates the cost reaching g from $s \in S$ via adding the costs of the actions' preconditions. Implicit assumption: "subgoal independence".
- Formally, h^{add} can be calculated as follows:

action vertex The cost of an action vertex $a \in A^i$ is $c(a)$ plus the sum of the predecessor vertex costs.

variable vertex
- The cost of a variable vertex v is 0 if $v \in V^0$.
- For all $v \in V^i$, $i > 0$, the cost of v equals the minimum cost of all predecessor vertices (these might be either action or variable vertices).

vertex set For a set of state variables $\bar{v} \subseteq V$, the cost equals the sum of costs of the variables in $\bar{v}$.

h^{add} for Classical Planning

Let $\mathcal{P} = \langle V, A, s_I, g\rangle$ be a STRIPS planning problem and $\mathcal{G} = \langle \bar{V}, \bar{A}\rangle$ its rPG.

- $h^{add}(s)$ calculates the cost reaching g from $s \in S$ via adding the costs of the actions' preconditions. Implicit assumption: "subgoal independence".
- Formally, h^{add} can be calculated as follows:

action vertex The cost of an action vertex $a \in A^i$ is $c(a)$ plus the sum of the predecessor vertex costs.

variable vertex
- The cost of a variable vertex v is 0 if $v \in V^0$.
- For all $v \in V^i$, $i > 0$, the cost of v equals the minimum cost of all predecessor vertices (these might be either action or variable vertices).

vertex set For a set of state variables $\bar{v} \subseteq V$, the cost equals the sum of costs of the variables in $\bar{v}$.

heuristic For a state $s \in S$, $h^{add}(s)$ equals the cost of g.

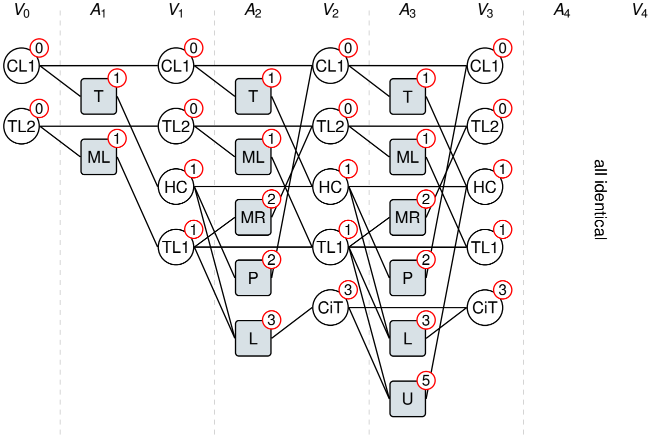
Example – Exercise!

Calculate h^{add} for the Cranes in the Harbor domain.

$s_I = \{CrateAtLoc1, TruckAtLoc2\} \quad g = \{CrateInTruck, TruckAtLoc2\}$

Example – Exercise!

Calculate h^{add} for the Cranes in the Harbor domain.



$s_I = \{CrateAtLoc1, TruckAtLoc2\}$ $\quad g = \{CrateInTruck, TruckAtLoc2\}$

$h^{add}(s_I) = 3 \quad h^*(s_I) = 4$

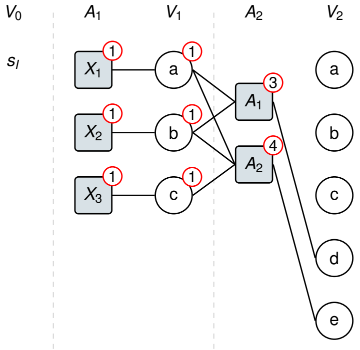
Admissibility

Is h^{add} admissible?

Admissibility

Is h^{add} admissible? No.

Heuristic assumes subgoal independence, which is normally not given:



$$s_I = \emptyset \quad g = \{d, e\} \quad h^{add}(s_I) = 7 \quad h^*(s_I) = 5$$

h^{FF} for Classical Planning

Let $\mathcal{P} = \langle V, A, s_I, g \rangle$ be a STRIPS planning problem and $\mathcal{G} = \langle \bar{V}, \bar{A} \rangle$ its rPG.

- h^{FF} calculates some (delete-relaxed) plan for $\mathcal{P}^+$.

h^{FF} for Classical Planning

Let $\mathcal{P} = \langle V, A, s_I, g \rangle$ be a STRIPS planning problem and $\mathcal{G} = \langle \bar{V}, \bar{A} \rangle$ its rPG.

- h^{FF} calculates some (delete-relaxed) plan for $\mathcal{P}^+$.
- Formally, h^{FF} is defined as follows:

h^{FF} for Classical Planning

Let $\mathcal{P} = \langle V, A, s_I, g \rangle$ be a STRIPS planning problem and $\mathcal{G} = \langle \bar{V}, \bar{A} \rangle$ its rPG.

- h^{FF} calculates some (delete-relaxed) plan for $\mathcal{P}^+$.
- Formally, h^{FF} is defined as follows:
  - Compute rPG until the goals are reached.

h^{FF} for Classical Planning

Let $\mathcal{P} = \langle V, A, s_I, g \rangle$ be a STRIPS planning problem and $\mathcal{G} = \langle \bar{V}, \bar{A} \rangle$ its rPG.

- h^{FF} calculates some (delete-relaxed) plan for $\mathcal{P}^+$.
- Formally, h^{FF} is defined as follows:
  - Compute rPG until the goals are reached.
  - For making a set of state variables true (starting with the goals), select a set of actions that achieve them.

h^{FF} for Classical Planning

Let $\mathcal{P} = \langle V, A, s_I, g \rangle$ be a STRIPS planning problem and $\mathcal{G} = \langle \bar{V}, \bar{A} \rangle$ its rPG.

- h^{FF} calculates some (delete-relaxed) plan for $\mathcal{P}^+$.
- Formally, h^{FF} is defined as follows:
  - Compute rPG until the goals are reached.
  - For making a set of state variables true (starting with the goals), select a set of actions that achieve them.
  - For each selected action, repeat the process for their preconditions.

h^{FF} for Classical Planning

Let $\mathcal{P} = \langle V, A, s_I, g \rangle$ be a STRIPS planning problem and $\mathcal{G} = \langle \bar{V}, \bar{A} \rangle$ its rPG.

- h^{FF} calculates some (delete-relaxed) plan for $\mathcal{P}^+$.
- Formally, h^{FF} is defined as follows:
  - Compute rPG until the goals are reached.
  - For making a set of state variables true (starting with the goals), select a set of actions that achieve them.
  - For each selected action, repeat the process for their preconditions.

Tie-Breaking

h^{FF} for Classical Planning

Let $\mathcal{P} = \langle V, A, s_I, g \rangle$ be a STRIPS planning problem and $\mathcal{G} = \langle \bar{V}, \bar{A} \rangle$ its rPG.

- h^{FF} calculates some (delete-relaxed) plan for $\mathcal{P}^+$.
- Formally, h^{FF} is defined as follows:
  - Compute rPG until the goals are reached.
  - For making a set of state variables true (starting with the goals), select a set of actions that achieve them.
  - For each selected action, repeat the process for their preconditions.

Tie-Breaking: Always select an "easy" action first – easy meaning small $\sum_{v \in pre(a)} \min\{i \mid v \text{ is in fact layer } i\}$ value.

h^{FF} for Classical Planning

Let $\mathcal{P} = \langle V, A, s_I, g \rangle$ be a STRIPS planning problem and $\mathcal{G} = \langle \bar{V}, \bar{A} \rangle$ its rPG.

- h^{FF} calculates some (delete-relaxed) plan for $\mathcal{P}^+$.
- Formally, h^{FF} is defined as follows:
  - Compute rPG until the goals are reached.
  - For making a set of state variables true (starting with the goals), select a set of actions that achieve them.
  - For each selected action, repeat the process for their preconditions.

Tie-Breaking: Always select an "easy" action first – easy meaning small $\sum_{v \in pre(a)} \min\{i \mid v \text{ is in fact layer } i\}$ value.

heuristic: For a state $s \in S$, $h^{FF}(s)$ equals the cost of the extracted plan (selected actions).

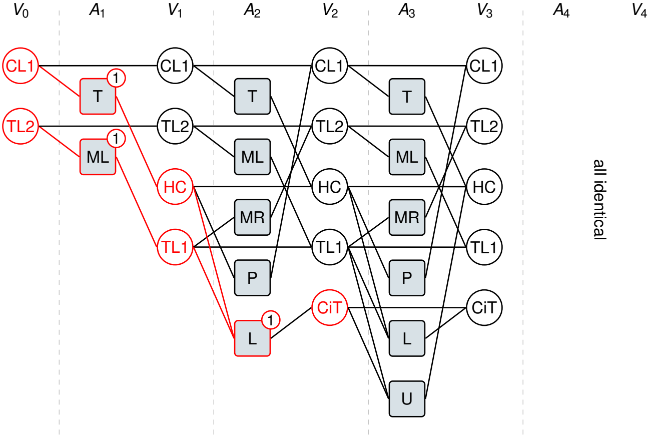
Example – Exercise!

Calculate h^{FF} for the Cranes in the Harbor domain.

$$s_I = \{CrateAtLoc1, TruckAtLoc2\} \quad g = \{CrateInTruck, TruckAtLoc2\}$$

Example – Exercise!

Calculate h^{FF} for the Cranes in the Harbor domain.



$s_I = \{CrateAtLoc1, TruckAtLoc2\}$ $\quad g = \{CrateInTruck, TruckAtLoc2\}$

$h^{FF}(s_I) = 3 \quad h^*(s_I) = 4$

Admissibility

Is h^{FF} admissible?

Admissibility

Is h^{FF} admissible? No (except for the practice).

Admissibility

Is h^{FF} admissible? No (except for the practice).

If the *original* planning problem $\mathcal{P}$ happens to be delete-free already, it might (easily) happen that a suboptimal set of actions is selected.

Admissibility

Is h^{FF} admissible? No (except for the practice).

If the *original* planning problem $\mathcal{P}$ happens to be delete-free already, it might (easily) happen that a suboptimal set of actions is selected.

Theorem

Computing h^{+} is $\mathbb{NP}$-complete.

Hardness of Solving $\mathcal{P}^+$ Optimally

Theorem

Computing h^+ is $\mathbb{NP}$-complete.

Hardness of Solving $\mathcal{P}^+$ Optimally

Theorem

Computing h^+ is $\mathbb{NP}$-complete.

Membership Proof:

Hardness of Solving $\mathcal{P}^+$ Optimally

Theorem

Computing h^+ is $\mathbb{NP}$-complete.

Membership Proof:

- Each action needs to be applied at most once.

Hardness of Solving $\mathcal{P}^+$ Optimally

Theorem

Computing h^+ is $\mathbb{NP}$-complete.

Membership Proof:

- Each action needs to be applied at most once.
- Thus, the maximum *required* plan length (to achieve any goal description) is bounded by $b \leq |A|$.

Hardness of Solving $\mathcal{P}^+$ Optimally

Theorem

Computing h^+ is $\mathbb{NP}$-complete.

Membership Proof:

- Each action needs to be applied at most once.
- Thus, the maximum *required* plan length (to achieve any goal description) is bounded by $b \leq |A|$.
- Now, guess a sequence of b actions and verify in linear time whether it's applicable.

Hardness of Solving $\mathcal{P}^+$ Optimally

Theorem

Computing h^+ is $\mathbb{NP}$-complete.

Membership Proof:

- Each action needs to be applied at most once.
- Thus, the maximum *required* plan length (to achieve any goal description) is bounded by $b \leq |A|$.
- Now, guess a sequence of b actions and verify in linear time whether it's applicable.
- Return true or false (depending on whether all goals hold in the final state).

Hardness of Solving $\mathcal{P}^+$ Optimally

Theorem

Computing h^+ is $\mathbb{NP}$-complete.

Hardness Proof:

Hardness of Solving $\mathcal{P}^+$ Optimally

Theorem

Computing h^+ is $\mathbb{NP}$-complete.

Hardness Proof: Reduction from CNF-SAT:

Hardness of Solving $\mathcal{P}^+$ Optimally

Theorem

Computing h^+ is $\mathbb{NP}$-complete.

Hardness Proof: Reduction from CNF-SAT:

- Let $\varphi = \underbrace{\{C_1, \ldots, C_n\}}_{\text{clauses}}$, $C_j = \underbrace{\{\varphi_{j_1}, \ldots, \varphi_{j_k}\}}_{\text{literals}}$, and $V = \underbrace{\{x_1, \ldots, x_m\}}_{\text{variables}}$.

Hardness of Solving $\mathcal{P}^+$ Optimally

Theorem

Computing h^+ is $\mathbb{NP}$-complete.

Hardness Proof: Reduction from CNF-SAT:

- Let $\varphi = \underbrace{\{C_1, \ldots, C_n\}}_{\text{clauses}}$, $C_j = \underbrace{\{\varphi_{j_1}, \ldots, \varphi_{j_k}\}}_{\text{literals}}$, and $V = \underbrace{\{x_1, \ldots, x_m\}}_{\text{variables}}$.
- For each boolean variable $x_i \in V$ add two actions to A:

$$\boxed{x_i \mapsto \top}\,\frac{x_i - \top}{x_i - set} \qquad \boxed{x_i \mapsto \bot}\,\frac{x_i - \bot}{x_i - set}$$

Hardness of Solving $\mathcal{P}^+$ Optimally

Theorem

Computing h^+ is $\mathbb{NP}$-complete.

Hardness Proof: Reduction from CNF-SAT:

- Let $\varphi = \underbrace{\{C_1, \ldots, C_n\}}_{\text{clauses}}$, $C_j = \underbrace{\{\varphi_{j_1}, \ldots, \varphi_{j_k}\}}_{\text{literals}}$, and $V = \underbrace{\{x_1, \ldots, x_m\}}_{\text{variables}}$.
- For each boolean variable $x_i \in V$ add two actions to A:

$$\boxed{x_i \mapsto \top}\ \begin{array}{l} x_i - \top \\ x_i - set \end{array} \qquad \boxed{x_i \mapsto \bot}\ \begin{array}{l} x_i - \bot \\ x_i - set \end{array}$$

- For each *positive* $\varphi_{j_i} = x_{j_i}$ or *negative* $\varphi_{j_i} = \neg x_{j_i}$ add

$$x_{j_i} - \top\ \boxed{\begin{array}{c} \text{“}x_{j_i} = \top\text{”} \\ C_j \mapsto \top \end{array}}\ C_j - \top \quad \text{or} \quad x_{j_i} - \bot\ \boxed{\begin{array}{c} \text{“}x_{j_i} = \bot\text{”} \\ C_j \mapsto \top \end{array}}\ C_j - \top$$

Hardness of Solving $\mathcal{P}^+$ Optimally

Theorem

Computing h^+ is $\mathbb{NP}$-complete.

Hardness Proof: Reduction from CNF-SAT:

- Let $\varphi = \underbrace{\{C_1, \dots, C_n\}}_{\text{clauses}}$, $C_j = \underbrace{\{\varphi_{j_1}, \dots, \varphi_{j_k}\}}_{\text{literals}}$, and $V = \underbrace{\{x_1, \dots, x_m\}}_{\text{variables}}$.
- For each boolean variable $x_i \in V$ add two actions to A:

  $\boxed{x_i \mapsto \top}$ $x_i - \top$, $x_i - set$ $\quad$ $\boxed{x_i \mapsto \bot}$ $x_i - \bot$, $x_i - set$

- For each *positive* $\varphi_{j_i} = x_{j_i}$ or *negative* $\varphi_{j_i} = \neg x_{j_i}$ add

  $x_{j_i} - \top$ $\boxed{\begin{array}{c}\text{“}x_{j_i} = \top\text{”} \\ C_j \mapsto \top\end{array}}$ $C_j - \top$ or $x_{j_i} - \bot$ $\boxed{\begin{array}{c}\text{“}x_{j_i} = \bot\text{”} \\ C_j \mapsto \top\end{array}}$ $C_j - \top$

- $g = \{x_i - set \mid 1 \leq i \leq m\} \cup \{C_j - \top \mid 1 \leq j \leq n\}$

Hardness of Solving $\mathcal{P}^+$ Optimally

Theorem

Computing h^+ is $\mathbb{NP}$-complete.

Hardness Proof: Reduction from CNF-SAT:

- Let $\varphi = \underbrace{\{C_1, \ldots, C_n\}}_{\text{clauses}}$, $C_j = \underbrace{\{\varphi_{j_1}, \ldots, \varphi_{j_k}\}}_{\text{literals}}$, and $V = \underbrace{\{x_1, \ldots, x_m\}}_{\text{variables}}$.
- For each boolean variable $x_i \in V$ add two actions to A:

  $\boxed{x_i \mapsto \top}$ effects: $x_i - \top$, $x_i - set$

  $\boxed{x_i \mapsto \bot}$ effects: $x_i - \bot$, $x_i - set$

- For each *positive* $\varphi_{j_i} = x_{j_i}$ or *negative* $\varphi_{j_i} = \neg x_{j_i}$ add

  precondition $x_{j_i} - \top$ → $\boxed{\text{“}x_{j_i} = \top\text{” } C_j \mapsto \top}$ → effect $C_j - \top$ or precondition $x_{j_i} - \bot$ → $\boxed{\text{“}x_{j_i} = \bot\text{” } C_j \mapsto \top}$ → effect $C_j - \top$

- $g = \{x_i - set \mid 1 \leq i \leq m\} \cup \{C_j - \top \mid 1 \leq j \leq n\}$
- φ is satisfiable if and only if a plan of size $n + m$ exists.

Classical Heuristics, Literature

Heuristics h^{add} and planner HSP:

- Blai Bonet and Héctor Geffner. "Planning as Heuristic Search: New Results". In: *Proc. of the 5th Europ. Conf. on Planning: Recent Advances in AI Planning (ECP 1999)*. Springer, 1999, pp. 360–372
- Patrik Haslum and Héctor Geffner. "Admissible Heuristics for Optimal Planning". In: *Proc. of the 5th Int. Conf. on Artificial Intelligence Planning Systems (AIPS 2000)*. AAAI Press, 2000, pp. 140–149

Heuristics h^{max}, h^{m} (not shown here), h^{add} (recap)

- Patrik Haslum and Héctor Geffner. "Admissible Heuristics for Optimal Planning". In: *Proc. of the 5th Int. Conf. on Artificial Intelligence Planning Systems (AIPS 2000)*. AAAI Press, 2000, pp. 140–149

Heuristic h^{FF}, planner FF, and *relaxed* planning graph:

- Jörg Hoffmann and Berhard Nebel. "The FF Planning System: Fast Plan Generation Through Heuristic Search". In: *Journal of Artificial Intelligence Research (JAIR)* 14 (2001), pp. 253–302

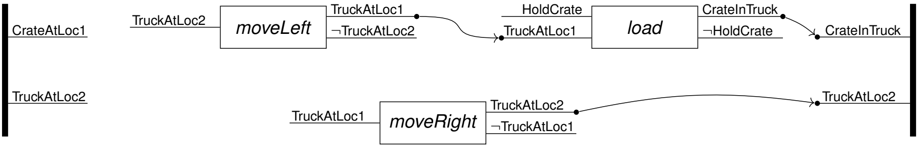
Classical vs. POCL Heuristics

Reminder Classical Planning:

Classical planning heuristics take the *current state* as input and estimate the goal distance to some *goal state*.

POCL Planning:

Here, there is neither a *current state* nor a goal description (it might be satisfied already). Instead, what do we have?



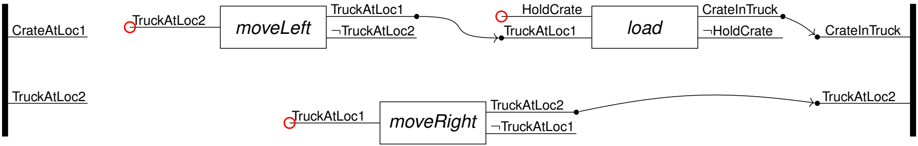
Classical vs. POCL Heuristics

Reminder Classical Planning:

Classical planning heuristics take the *current state* as input and estimate the goal distance to some *goal state*.

POCL Planning:

Here, there is neither a *current state* nor a goal description (it might be satisfied already). Instead, what do we have? → Flaws!



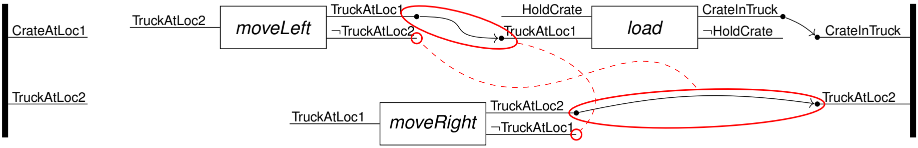
Classical vs. POCL Heuristics

Reminder Classical Planning:

Classical planning heuristics take the *current state* as input and estimate the goal distance to some *goal state*.

POCL Planning:

Here, there is neither a *current state* nor a goal description (it might be satisfied already). Instead, what do we have? → Flaws!



POCL Heuristics

So, how to compute heuristics for partial plans?

- Count *all* flaws.

POCL Heuristics

So, how to compute heuristics for partial plans?

- Count *all* flaws.
- Count a *subset* of all flaws

POCL Heuristics

So, how to compute heuristics for partial plans?

- Count *all* flaws.
- Count a *subset* of all flaws, e.g. the open preconditions – called the *OC* heuristic (see Nguyen and Kambhampati).

POCL Heuristics

So, how to compute heuristics for partial plans?

- Count *all* flaws.
- Count a *subset* of all flaws, e.g. the open preconditions – called the *OC* heuristic (see Nguyen and Kambhampati).
- Via compilation:

POCL Heuristics

So, how to compute heuristics for partial plans?

- Count *all* flaws.
- Count a *subset* of all flaws, e.g. the open preconditions – called the *OC* heuristic (see Nguyen and Kambhampati).
- Via compilation:
  - Translate each search node into a linear program (see Bylander).

POCL Heuristics

So, how to compute heuristics for partial plans?

- Count *all* flaws.
- Count a *subset* of all flaws, e.g. the open preconditions – called the *OC* heuristic (see Nguyen and Kambhampati).
- Via compilation:
  - Translate each search node into a linear program (see Bylander).
  - Translate each search node into a (new/altered) *classical* problem and use standard classical heuristics (see Bercher et al.).

POCL Heuristics

So, how to compute heuristics for partial plans?

- Count *all* flaws.
- Count a *subset* of all flaws, e.g. the open preconditions – called the *OC* heuristic (see Nguyen and Kambhampati).
- Via compilation:
  - Translate each search node into a linear program (see Bylander).
  - Translate each search node into a (new/altered) *classical* problem and use standard classical heuristics (see Bercher et al.).
- Directly *adapt* heuristics for classical planning:

POCL Heuristics

So, how to compute heuristics for partial plans?

- Count *all* flaws.
- Count a *subset* of all flaws, e.g. the open preconditions – called the *OC* heuristic (see Nguyen and Kambhampati).
- Via compilation:
  - Translate each search node into a linear program (see Bylander).
  - Translate each search node into a (new/altered) *classical* problem and use standard classical heuristics (see Bercher et al.).
- Directly *adapt* heuristics for classical planning:
  - FF heuristic $\rightarrow$ *Relax* heuristic (see Nguyen and Kambhampati).

POCL Heuristics

So, how to compute heuristics for partial plans?

- Count *all* flaws.
- Count a *subset* of all flaws, e.g. the open preconditions – called the *OC* heuristic (see Nguyen and Kambhampati).
- Via compilation:
  - Translate each search node into a linear program (see Bylander).
  - Translate each search node into a (new/altered) *classical* problem and use standard classical heuristics (see Bercher et al.).
- Directly *adapt* heuristics for classical planning:
  - FF heuristic $\rightarrow$ *Relax* heuristic (see Nguyen and Kambhampati).
  - Add heuristic $\rightarrow$ *Add heuristic for POCL planning* (see Younes and Simmons).

POCL Heuristics, Literature

Nguyen and Kambhampati — XuanLong Nguyen and Subbarao Kambhampati. "Reviving Partial Order Planning". In: *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001)*. Morgan Kaufmann, 2001, pp. 459–466

Bercher et al. — Pascal Bercher et al. "Using State-Based Planning Heuristics for Partial-Order Causal-Link Planning". In: *Advances in Artificial Intelligence, Proc. of the 36th German Conf. on Artificial Intelligence (KI 2013)*. Springer, 2013, pp. 1–12

Bylander — Tom Bylander. "A Linear Programming Heuristic for Optimal Planning". In: *Proc. of the 14th National Conf. on Artificial Intelligence (AAAI 1997)*. AAAI Press, 1997, pp. 694–699

Younes and Simmons — Håkan L. S. Younes and Reid G. Simmons. "VHPOP: Versatile heuristic partial order planner". In: *Journal of Artificial Intelligence Research (JAIR)* 20 (2003), pp. 405–430

Adapting Classical Heuristics for POCL Planning

Again:

Classical planning heuristics take the *current state* as input and estimate the goal distance to some *goal state*.

POCL Planning:

Here, there is neither a *current state* nor a goal description – but a partial plan with flaws.

Now what?

- What do we do? How to bring both worlds together?

Adapting Classical Heuristics for POCL Planning

Again:

Classical planning heuristics take the *current state* as input and estimate the goal distance to some *goal state*.

POCL Planning:

Here, there is neither a *current state* nor a goal description – but a partial plan with flaws.

Now what?

- What do we do? How to bring both worlds together?

$\rightarrow$ Use the partial plan's initial state as initial state of heuristic.

Adapting Classical Heuristics for POCL Planning

Again:

Classical planning heuristics take the *current state* as input and estimate the goal distance to some *goal state*.

POCL Planning:

Here, there is neither a *current state* nor a goal description – but a partial plan with flaws.

Now what?

- What do we do? How to bring both worlds together?

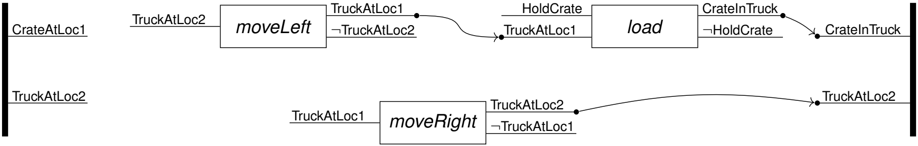
$\rightarrow$ Use the partial plan's initial state as initial state of heuristic.

$\rightarrow$ Use the the open (i.e., unprotected) preconditions as goal state.

Adapting Classical Heuristics for POCL Planning, cont'd

Using classical heuristics in POCL planning:

- Use the partial plan's initial state as initial state of heuristic.
- Use the the open (i.e., unprotected) preconditions as goal state.

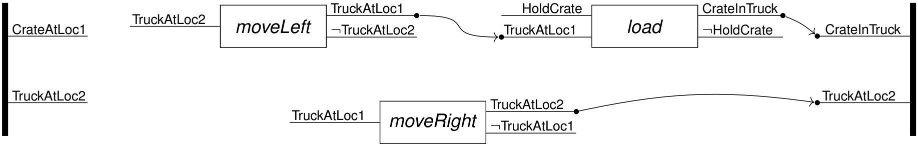


What problems could arise from doing this?

Adapting Classical Heuristics for POCL Planning, cont'd

Using classical heuristics in POCL planning:

- Use the partial plan's initial state as initial state of heuristic.
- Use the the open (i.e., unprotected) preconditions as goal state.



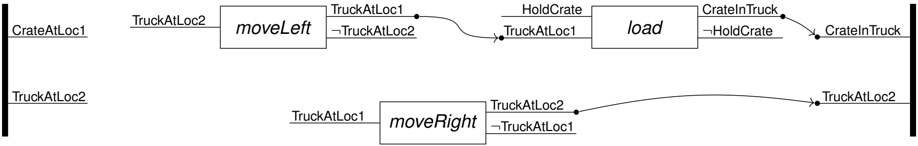
What problems could arise from doing this?

- This ignores negative effects and the causal links' pruning power.

Adapting Classical Heuristics for POCL Planning, cont'd

Using classical heuristics in POCL planning:

- Use the partial plan's initial state as initial state of heuristic.
- Use the the open (i.e., unprotected) preconditions as goal state.



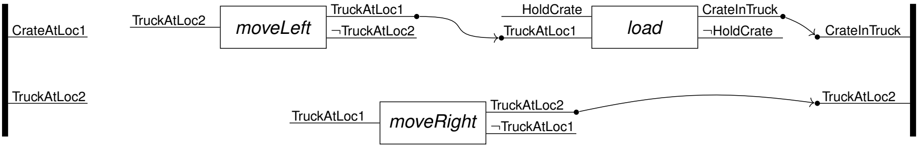
What problems could arise from doing this?

- This ignores negative effects and the causal links' pruning power.
- We get unreachable goals: $\{TruckAtLoc1, TruckAtLoc2\}$

Adapting Classical Heuristics for POCL Planning, cont'd

Using classical heuristics in POCL planning:

- Use the partial plan's initial state as initial state of heuristic.
- Use the the open (i.e., unprotected) preconditions as goal state.



What problems could arise from doing this?

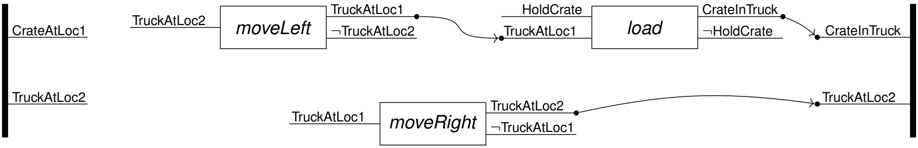
- This ignores negative effects and the causal links' pruning power.
- We get unreachable goals: $\{TruckAtLoc1, TruckAtLoc2\}$

Why – or more precisely: *when* – does this work?

Adapting Classical Heuristics for POCL Planning, cont'd

Using classical heuristics in POCL planning:

- Use the partial plan's initial state as initial state of heuristic.
- Use the the open (i.e., unprotected) preconditions as goal state.



What problems could arise from doing this?

- This ignores negative effects and the causal links' pruning power.
- We get unreachable goals: $\{TruckAtLoc1, TruckAtLoc2\}$

Why – or more precisely: *when* – does this work?

- We only use heuristics that rely on (full) delete relaxation!

Add Heuristic for POCL Planning

Let $\langle V, A, s_I, g\rangle$ be a STRIPS planning problem.

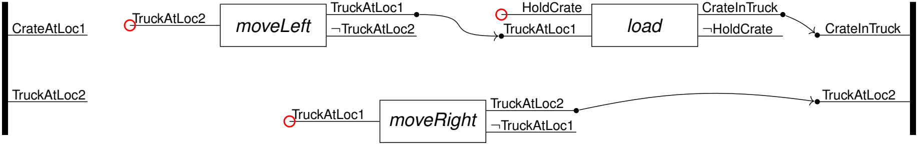
- Then, let $h^{add}_g(s)$ be the *classical* Add heuristic estimating the goal distance from some state $s \in S$ to the goals g.
  (In contrast to the last section, we now made the goals g explicit in the sub script.)

Add Heuristic for POCL Planning

Let $\langle V, A, s_I, g\rangle$ be a STRIPS planning problem.

- Then, let $h^{add}_g(s)$ be the *classical* Add heuristic estimating the goal distance from some state $s \in S$ to the goals g.
  (In contrast to the last section, we now made the goals g explicit in the sub script.)
- Then, with $h^{add}_{POCL}(P)$ we refer to the *Add Heuristic for POCL Planning* that estimates the goal distance from some current partial plan P to some solution plan. It is defined as $h^{add}_G(s_I)$, where G is the set of open preconditions of P.

Add Heuristic for POCL Planning, Example

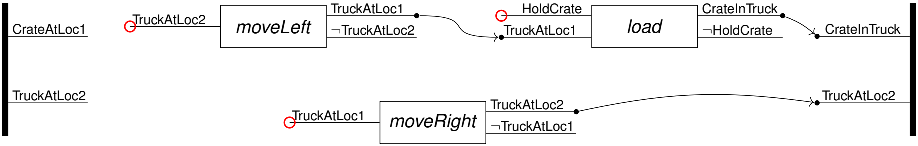


For $h^{add}_{POCL}(P)$, we use:

- $s_I = \{CrateAtLoc1, TruckAtLoc2\}$
- $G = \{TruckAtLoc1, TruckAtLoc2, HoldCrate\}$

Accounting for Positive Interactions

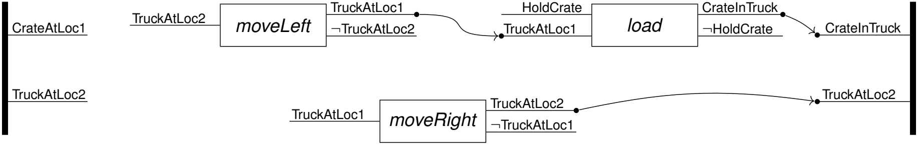
Estimating the goal distance to *all* open preconditions might be too pessimistic:



Here, the precondition *TrackAtLoc*2 might be accomplished by using the effect of *moveRight*.

Accounting for Positive Interactions

Estimating the goal distance to *all* open preconditions might be too pessimistic:



Here, the precondition *TrackAtLoc*2 might be accomplished by using the effect of *moveRight*.

Another example: Consider a (large) ”solution” plan in which (almost) all causal links are missing. $h^{add}_{POCL}(P)$ would be *highly* inaccurate.

Add Heuristic for POCL Planning Reusing Actions

Let $\langle V, A, s_I, g\rangle$ be a STRIPS planning problem.

- Then, let $h^{add}_g(s)$ be the *classical* Add heuristic estimating the goal distance from some state $s \in S$ to the goals g.
  (In contrast to the last section, we now made the goals g explicit in the sub script.)

Add Heuristic for POCL Planning Reusing Actions

Let $\langle V, A, s_I, g\rangle$ be a STRIPS planning problem.

- Then, let $h^{add}_g(s)$ be the *classical* Add heuristic estimating the goal distance from some state $s \in S$ to the goals g.
  (In contrast to the last section, we now made the goals g explicit in the sub script.)
- Then, with $h^{add-r}_{POCL}(P)$, $P = (PS, \prec, CL)$, we refer to the *Add Heuristic for POCL Planning Reusing Actions* that estimates the goal distance from some current partial plan P to some solution plan. It is defined as $h^{add}_G(s_I)$, where G is a subset of open preconditions of P, i.e.,

.

Add Heuristic for POCL Planning Reusing Actions

Let $\langle V, A, s_I, g\rangle$ be a STRIPS planning problem.

- Then, let $h^{add}_{g}(s)$ be the *classical* Add heuristic estimating the goal distance from some state $s \in S$ to the goals g.
  (In contrast to the last section, we now made the goals g explicit in the sub script.)
- Then, with $h^{add-r}_{POCL}(P)$, $P = (PS, \prec, CL)$, we refer to the *Add Heuristic for POCL Planning Reusing Actions* that estimates the goal distance from some current partial plan P to some solution plan. It is defined as $h^{add}_{G}(s_I)$, where G is a subset of open preconditions of P, i.e., $G = \{v \mid (v, ps)$ is an open precondition of P and there is no plan step $ps' \in PS$ with $v \in add(ps')$ such that $\prec \cup \{(ps', ps)\}$ is a strict partial order$\}$.

Relax Heuristic

The FF heuristic for POCL planning – called *Relax heuristic* or h^{FF}_{POCL} – transfers the ideas of h^{add}_{POCL} to the FF heuristic:

Relax Heuristic

The FF heuristic for POCL planning – called *Relax heuristic* or h^{FF}_{POCL} – transfers the ideas of h^{add}_{POCL} to the FF heuristic:

- It relies on a rPG. (This is not *perfectly* true, but for the sake of simplicity we assume this here.)

Relax Heuristic

The FF heuristic for POCL planning – called *Relax heuristic* or h^{FF}_{POCL} – transfers the ideas of h^{add}_{POCL} to the FF heuristic:

- It relies on a rPG. (This is not *perfectly* true, but for the sake of simplicity we assume this here.)
- As goal state we consider the set of open preconditions – just as h^{add}_{POCL} does.

Relax Heuristic

The FF heuristic for POCL planning – called *Relax heuristic* or h^{FF}_{POCL} – transfers the ideas of h^{add}_{POCL} to the FF heuristic:

- It relies on a rPG. (This is not *perfectly* true, but for the sake of simplicity we assume this here.)
- As goal state we consider the set of open preconditions – just as h^{add}_{POCL} does.
- We than extract a plan from the rPG in the same way the FF heuristic does. However, the cost of an action a, $c(a)$ in that relaxed solution plan is only accounted for if a does not occur in the input plan P.

Summary

- Many heuristics base upon the relaxed planning graph (or an relaxed action model).

Summary

- Many heuristics base upon the relaxed planning graph (or an relaxed action model).
- h^{max} is admissible and can be computed in $\mathbb{P}$.

Summary

- Many heuristics base upon the relaxed planning graph (or an relaxed action model).
- h^{max} is admissible and can be computed in $\mathbb{P}$.
- h^{add} is inadmissible and can be computed in $\mathbb{P}$.

Summary

- Many heuristics base upon the relaxed planning graph (or an relaxed action model).
- h^{max} is admissible and can be computed in $\mathbb{P}$.
- h^{add} is inadmissible and can be computed in $\mathbb{P}$.
- h^{FF} is inadmissible (in theory, but often admissible in practice) and can be computed in $\mathbb{P}$.

Summary

- Many heuristics base upon the relaxed planning graph (or an relaxed action model).
- h^{max} is admissible and can be computed in $\mathbb{P}$.
- h^{add} is inadmissible and can be computed in $\mathbb{P}$.
- h^{FF} is inadmissible (in theory, but often admissible in practice) and can be computed in $\mathbb{P}$.
- h^{+} is admissible and $\mathbb{NP}$-complete to compute.

Summary

- Many heuristics base upon the relaxed planning graph (or an relaxed action model).
- h^{max} is admissible and can be computed in $\mathbb{P}$.
- h^{add} is inadmissible and can be computed in $\mathbb{P}$.
- h^{FF} is inadmissible (in theory, but often admissible in practice) and can be computed in $\mathbb{P}$.
- h^{+} is admissible and $\mathbb{NP}$-complete to compute.
- All these heuristics take the *current state* as input and estimate the goal distance to some *goal state*.

Summary

- Many heuristics base upon the relaxed planning graph (or an relaxed action model).
- h^{max} is admissible and can be computed in $\mathbb{P}$.
- h^{add} is inadmissible and can be computed in $\mathbb{P}$.
- h^{FF} is inadmissible (in theory, but often admissible in practice) and can be computed in $\mathbb{P}$.
- h^{+} is admissible and $\mathbb{NP}$-complete to compute.
- All these heuristics take the *current state* as input and estimate the goal distance to some *goal state*.
- But since they are delete-relaxed, they can be used for POCL planning as well.