

# Principles of AI Planning

## 7. Planning as search: relaxed planning tasks

Albert-Ludwigs-Universität Freiburg



**UNI  
FREIBURG**

Bernhard Nebel and Robert Mattmüller

November 11th, 2015



# How to obtain a heuristic

Obtaining  
heuristics

STRIPS heuristic

Relaxation and  
abstraction

Relaxed  
planning  
tasks

Summary



STRIPS (Fikes & Nilsson, 1971) used the number of state variables that differ in current state  $s$  and a STRIPS goal  $a_1 \wedge \dots \wedge a_n$ :

$$h(s) := |\{i \in \{1, \dots, n\} \mid s \neq a_i\}|.$$

**Intuition:** more true goal literals  $\rightsquigarrow$  closer to the goal

$\rightsquigarrow$  **STRIPS heuristic (a.k.a. goal-count heuristic)** (properties?)

**Note:** From now on, for convenience we usually write heuristics as functions of states (as above), not nodes. Node heuristic  $h'$  is defined from state heuristic  $h$  as  $h'(\sigma) := h(\text{state}(\sigma))$ .

Obtaining  
heuristics

STRIPS heuristic

Relaxation and  
abstraction

Relaxed  
planning  
tasks

Summary



## What is wrong with the STRIPS heuristic?

- quite **uninformative**:  
the range of heuristic values in a given task is small;  
typically, most successors have the same estimate
- very sensitive to **reformulation**:  
can easily transform any planning task into an equivalent one where  $h(s) = 1$  for all non-goal states (how?)
- ignores almost all **problem structure**:  
heuristic value does not depend on the set of operators!

⇒ need a better, principled way of coming up with heuristics

Obtaining  
heuristics

STRIPS heuristic

Relaxation and  
abstraction

Relaxed  
planning  
tasks

Summary



## General procedure for obtaining a heuristic

Solve an easier version of the problem.

Two common methods:

- **relaxation**: consider **less constrained** version of the problem
- **abstraction**: consider **smaller** version of real problem

Both have been very successfully applied in planning.

We consider both in this course, beginning with **relaxation**.

Obtaining  
heuristics

STRIPS heuristic

Relaxation and  
abstraction

Relaxed  
planning  
tasks

Summary

How do we relax a problem?

## Example (Route planning for a road network)

The road network is formalized as a weighted graph over points in the Euclidean plane. The weight of an edge is the **road distance** between two locations.

A relaxation **drops constraints** of the original problem.

## Example (Relaxation for route planning)

Use the **Euclidean distance**  $\sqrt{|x_1 - x_2|^2 + |y_1 - y_2|^2}$  as a heuristic for the road distance between  $\langle x_1, y_1 \rangle$  and  $\langle x_2, y_2 \rangle$ . This is a **lower bound** on the road distance ( $\rightsquigarrow$  admissible).

$\rightsquigarrow$  We drop the constraint of having to travel on roads.

Obtaining  
heuristics

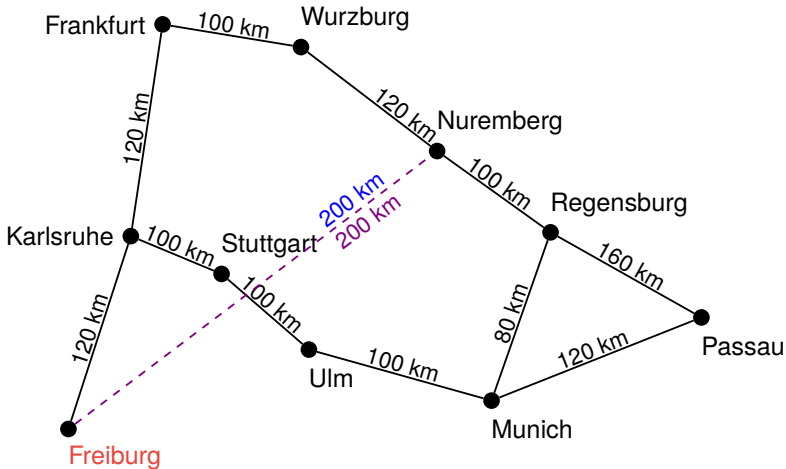
STRIPS heuristic

Relaxation and  
abstraction

Relaxed  
planning  
tasks

Summary

# A\* using the Euclidean distance heuristic



Obtaining heuristics

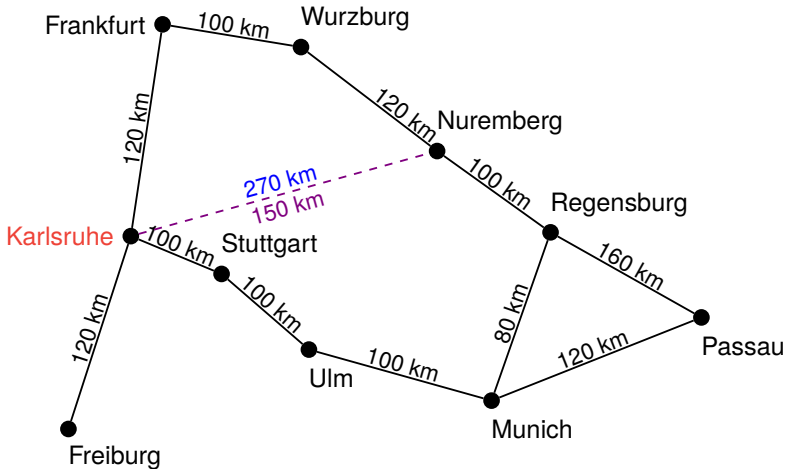
STRIPS heuristic

Relaxation and abstraction

Relaxed planning tasks

Summary

# A\* using the Euclidean distance heuristic



Obtaining  
heuristics

STRIPS heuristic

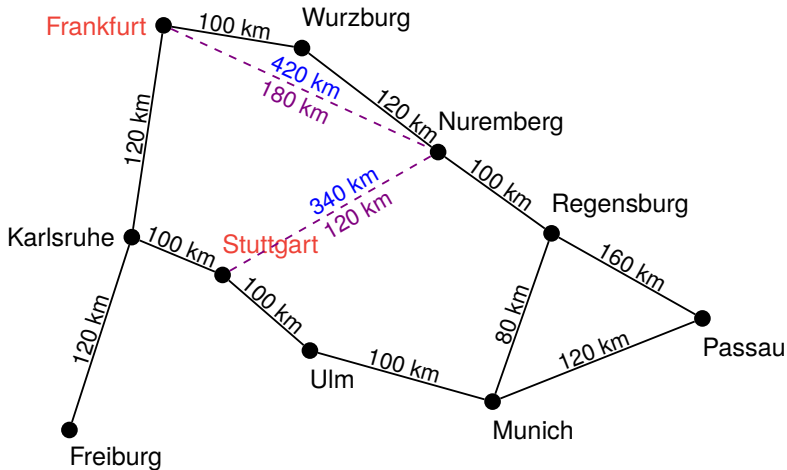
Relaxation and  
abstraction

Relaxed  
planning  
tasks

Summary



# A\* using the Euclidean distance heuristic



Obtaining  
heuristics

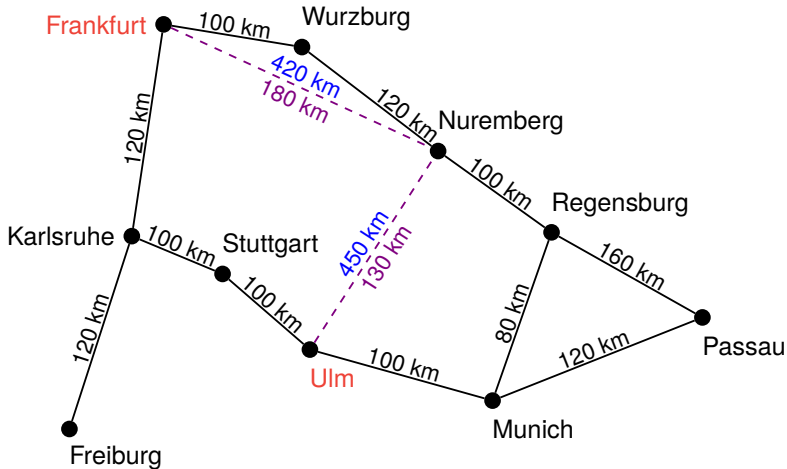
STRIPS heuristic

Relaxation and  
abstraction

Relaxed  
planning  
tasks

Summary

# A\* using the Euclidean distance heuristic



Obtaining  
heuristics

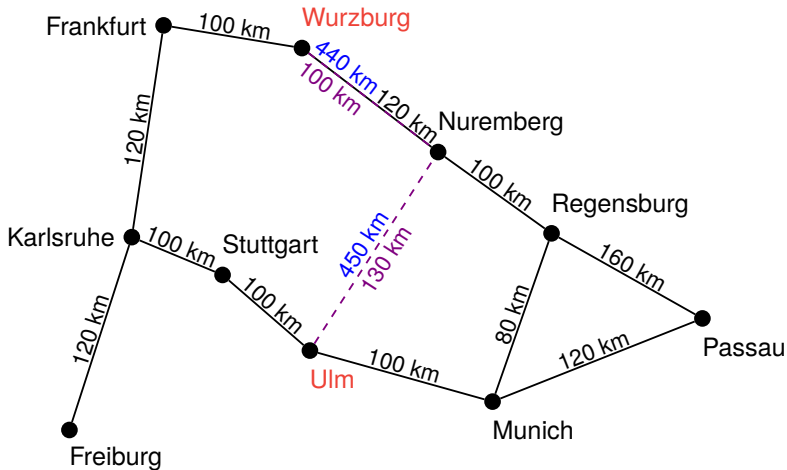
STRIPS heuristic

Relaxation and  
abstraction

Relaxed  
planning  
tasks

Summary

# A\* using the Euclidean distance heuristic



Obtaining  
heuristics

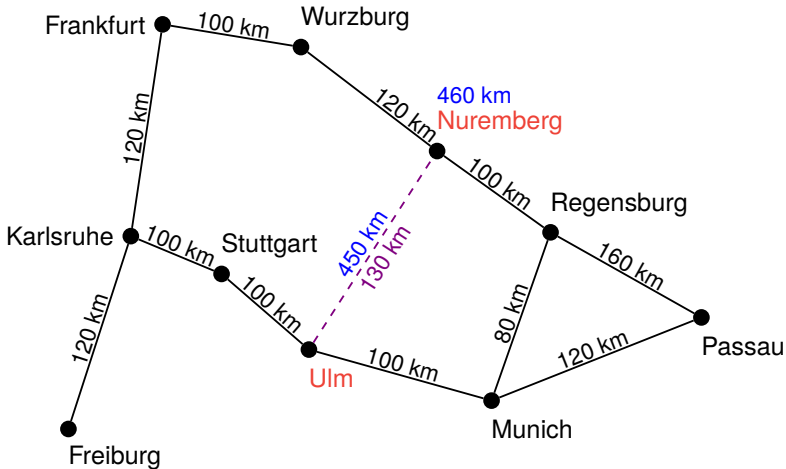
STRIPS heuristic

Relaxation and  
abstraction

Relaxed  
planning  
tasks

Summary

# A\* using the Euclidean distance heuristic



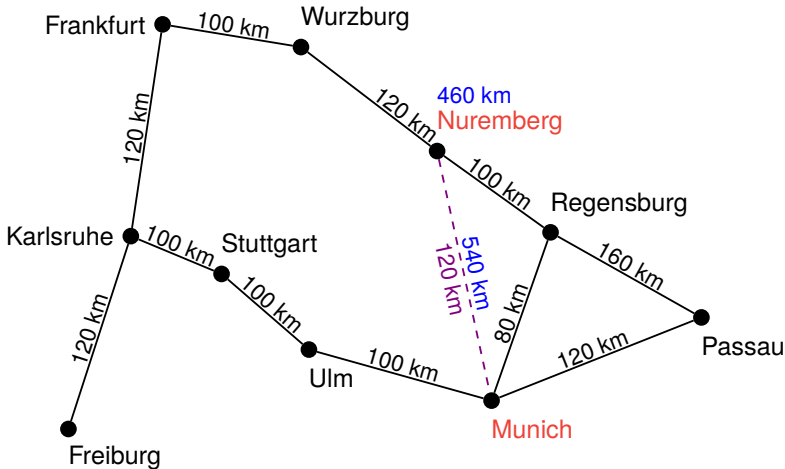
Obtaining  
heuristics

STRIPS heuristic  
Relaxation and  
abstraction

Relaxed  
planning  
tasks

Summary

# A\* using the Euclidean distance heuristic



Obtaining heuristics

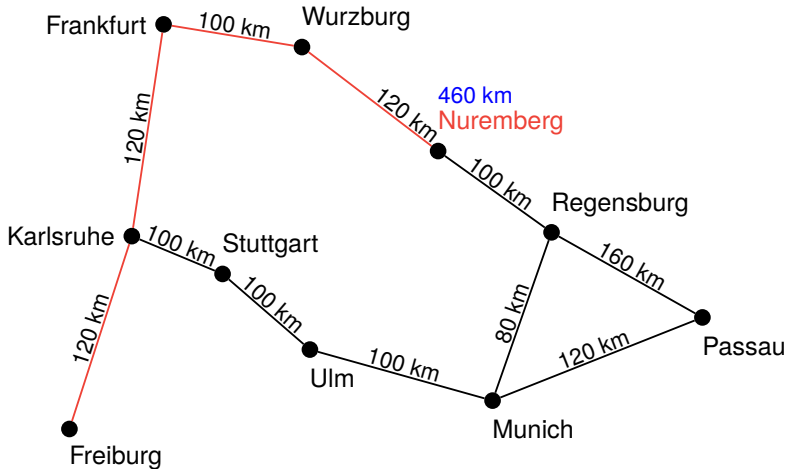
STRIPS heuristic

Relaxation and abstraction

Relaxed planning tasks

Summary

# A\* using the Euclidean distance heuristic



Obtaining  
heuristics

STRIPS heuristic

Relaxation and  
abstraction

Relaxed  
planning  
tasks

Summary



# Relaxed planning tasks

Obtaining  
heuristics

**Relaxed  
planning  
tasks**

Definition

The relaxation  
lemma

Greedy algorithm

Optimality

Discussion

Summary



In **positive normal form** (remember?), good and bad effects are easy to distinguish:

- Effects that make state variables true are good (**add effects**).
- Effects that make state variables false are bad (**delete effects**).

Idea for the heuristic: Ignore all delete effects.

Obtaining  
heuristics

Relaxed  
planning  
tasks

**Definition**

The relaxation  
lemma

Greedy algorithm

Optimality

Discussion

Summary





## Definition (relaxation of operators)

The **relaxation**  $o^+$  of an operator  $o = \langle \chi, e \rangle$  in positive normal form is the operator which is obtained by replacing all negative effects  $\neg a$  within  $e$  by the do-nothing effect  $\top$ .

## Definition (relaxation of planning tasks)

The **relaxation**  $\Pi^+$  of a planning task  $\Pi = \langle A, I, O, \gamma \rangle$  in positive normal form is the planning task  $\Pi^+ := \langle A, I, \{o^+ \mid o \in O\}, \gamma \rangle$ .

## Definition (relaxation of operator sequences)

The **relaxation** of an operator sequence  $\pi = o_1 \dots o_n$  is the operator sequence  $\pi^+ := o_1^+ \dots o_n^+$ .

Obtaining  
heuristics

Relaxed  
planning  
tasks

Definition

The relaxation  
lemma

Greedy algorithm

Optimality

Discussion

Summary



- Planning tasks in positive normal form without delete effects are called **relaxed planning tasks**.
- Plans for relaxed planning tasks are called **relaxed plans**.
- If  $\Pi$  is a planning task in positive normal form and  $\pi^+$  is a plan for  $\Pi^+$ , then  $\pi^+$  is called a **relaxed plan for  $\Pi$** .

Obtaining  
heuristics

Relaxed  
planning  
tasks

**Definition**

The relaxation  
lemma

Greedy algorithm

Optimality

Discussion

Summary

The **on-set**  $on(s)$  of a state  $s$  is the set of true state variables in  $s$ , i.e.  $on(s) = s^{-1}(\{1\})$ .

A state  $s'$  **dominates** another state  $s$  iff  $on(s) \subseteq on(s')$ .

## Lemma (domination)

*Let  $s$  and  $s'$  be valuations of a set of propositional variables  $A$  and let  $\chi$  be a propositional formula over  $A$  which does not contain negation symbols.*

*If  $s \models \chi$  and  $s'$  dominates  $s$ , then  $s' \models \chi$ .*

## Proof.

Proof by induction over the structure of  $\chi$ .

- Base case  $\chi = \top$ : then  $s' \models \top$ .
- Base case  $\chi = \perp$ : then  $s \not\models \perp$ .

Obtaining  
heuristics

Relaxed  
planning  
tasks

Definition

The relaxation  
lemma

Greedy algorithm

Optimality

Discussion

Summary

## Proof (ctd.)

- Base case  $\chi = a \in A$ : assume  $s \models a$  and  $on(s) \subseteq on(s')$ .  
With  $a \in on(s)$  we get  $a \in on(s')$ , hence  $s' \models a$ .
- Inductive case  $\chi = \chi_1 \wedge \chi_2$ : by induction hypothesis, our claim holds for the proper subformulas  $\chi_1$  and  $\chi_2$  of  $\chi$ .

$$\begin{aligned} s \models \chi &\iff s \models \chi_1 \wedge \chi_2 \\ &\iff s \models \chi_1 \text{ and } s \models \chi_2 \\ \text{I.H. (twice)} &\implies s' \models \chi_1 \text{ and } s' \models \chi_2 \\ &\iff s' \models \chi_1 \wedge \chi_2 \\ &\iff s' \models \chi. \end{aligned}$$

- Inductive case  $\chi = \chi_1 \vee \chi_2$ : Analogous.



Obtaining  
heuristics

Relaxed  
planning  
tasks

Definition

The relaxation  
lemma

Greedy algorithm

Optimality

Discussion

Summary

# The relaxation lemma



For the rest of this chapter, we assume that all planning tasks are in positive normal form.

## Lemma (relaxation)

*Let  $s$  be a state, let  $s'$  be a state that dominates  $s$ , and let  $\pi$  be an operator sequence which is applicable in  $s$ . Then  $\pi^+$  is applicable in  $s'$  and  $app_{\pi^+}(s')$  dominates  $app_{\pi}(s)$ . Moreover, if  $\pi$  leads to a goal state from  $s$ , then  $\pi^+$  leads to a goal state from  $s'$ .*

## Proof.

The “moreover” part follows from the rest by the domination lemma. Prove the rest by induction over the length of  $\pi$ .

Base case:  $\pi = \varepsilon$

$app_{\pi^+}(s') = s'$  dominates  $app_{\pi}(s) = s$  by assumption.

Obtaining heuristics

Relaxed planning tasks

Definition

The relaxation lemma

Greedy algorithm

Optimality

Discussion

Summary

# The relaxation lemma



For the rest of this chapter, we assume that all planning tasks are in positive normal form.

## Lemma (relaxation)

*Let  $s$  be a state, let  $s'$  be a state that dominates  $s$ , and let  $\pi$  be an operator sequence which is applicable in  $s$ . Then  $\pi^+$  is applicable in  $s'$  and  $app_{\pi^+}(s')$  dominates  $app_{\pi}(s)$ . Moreover, if  $\pi$  leads to a goal state from  $s$ , then  $\pi^+$  leads to a goal state from  $s'$ .*

## Proof.

The “moreover” part follows from the rest by the domination lemma. Prove the rest by induction over the length of  $\pi$ .

**Base case:**  $\pi = \varepsilon$

$app_{\pi^+}(s') = s'$  dominates  $app_{\pi}(s) = s$  by assumption.

Obtaining heuristics

Relaxed planning tasks

Definition

The relaxation lemma

Greedy algorithm

Optimality

Discussion

Summary

# The relaxation lemma (ctd.)



## Proof (ctd.)

**Inductive case:**  $\pi = o_1 \dots o_{n+1}$

By the induction hypothesis,  $o_1^+ \dots o_n^+$  is applicable in  $s'$ , and  $t' = \text{app}_{o_1^+ \dots o_n^+}(s')$  dominates  $t = \text{app}_{o_1 \dots o_n}(s)$ .

Let  $o := o_{n+1} = \langle \chi, e \rangle$  and  $o^+ = \langle \chi, e^+ \rangle$ . By assumption,  $o$  is applicable in  $t$ , and thus  $t \models \chi$ . By the domination lemma, we get  $t' \models \chi$  and hence  $o^+$  is applicable in  $t'$ . Therefore,  $\pi^+$  is applicable in  $s'$ .

Because  $o$  is in positive normal form, all effect conditions satisfied by  $t$  are also satisfied by  $t'$  (by the domination lemma). Therefore,  $([e]_t \cap A) \subseteq [e^+]_{t'}$  (where  $A$  is the set of state variables, or positive literals).

We get

$$\text{on}(\text{app}_{\pi}(s)) \subseteq \text{on}(t) \cup ([e]_t \cap A) \subseteq \text{on}(t') \cup [e^+]_{t'} = \text{on}(\text{app}_{\pi^+}(s')),$$

Obtaining  
heuristics

Relaxed  
planning  
tasks

Definition

The relaxation  
lemma

Greedy algorithm

Optimality

Discussion

Summary

# The relaxation lemma (ctd.)



## Proof (ctd.)

**Inductive case:**  $\pi = o_1 \dots o_{n+1}$

By the induction hypothesis,  $o_1^+ \dots o_n^+$  is applicable in  $s'$ , and  $t' = \text{app}_{o_1^+ \dots o_n^+}(s')$  dominates  $t = \text{app}_{o_1 \dots o_n}(s)$ .

Let  $o := o_{n+1} = \langle \chi, e \rangle$  and  $o^+ = \langle \chi, e^+ \rangle$ . By assumption,  $o$  is applicable in  $t$ , and thus  $t \models \chi$ . By the domination lemma, we get  $t' \models \chi$  and hence  $o^+$  is applicable in  $t'$ . Therefore,  $\pi^+$  is applicable in  $s'$ .

Because  $o$  is in positive normal form, all effect conditions satisfied by  $t$  are also satisfied by  $t'$  (by the domination lemma). Therefore,  $([e]_t \cap A) \subseteq [e^+]_{t'}$  (where  $A$  is the set of state variables, or positive literals).

We get

$$\text{on}(\text{app}_{\pi}(s)) \subseteq \text{on}(t) \cup ([e]_t \cap A) \subseteq \text{on}(t') \cup [e^+]_{t'} = \text{on}(\text{app}_{\pi^+}(s')),$$

Obtaining  
heuristics

Relaxed  
planning  
tasks

Definition

The relaxation  
lemma

Greedy algorithm

Optimality

Discussion

Summary



# The relaxation lemma (ctd.)



## Proof (ctd.)

**Inductive case:**  $\pi = o_1 \dots o_{n+1}$

By the induction hypothesis,  $o_1^+ \dots o_n^+$  is applicable in  $s'$ , and  $t' = \text{app}_{o_1^+ \dots o_n^+}(s')$  dominates  $t = \text{app}_{o_1 \dots o_n}(s)$ .

Let  $o := o_{n+1} = \langle \chi, e \rangle$  and  $o^+ = \langle \chi, e^+ \rangle$ . By assumption,  $o$  is applicable in  $t$ , and thus  $t \models \chi$ . By the domination lemma, we get  $t' \models \chi$  and hence  $o^+$  is applicable in  $t'$ . Therefore,  $\pi^+$  is applicable in  $s'$ .

Because  $o$  is in positive normal form, all effect conditions satisfied by  $t$  are also satisfied by  $t'$  (by the domination lemma). Therefore,  $([e]_t \cap A) \subseteq [e^+]_{t'}$  (where  $A$  is the set of state variables, or positive literals).

We get

$$\text{on}(\text{app}_{\pi}(s)) \subseteq \text{on}(t) \cup ([e]_t \cap A) \subseteq \text{on}(t') \cup [e^+]_{t'} = \text{on}(\text{app}_{\pi^+}(s')),$$

Obtaining  
heuristics

Relaxed  
planning  
tasks

Definition

The relaxation  
lemma

Greedy algorithm

Optimality

Discussion

Summary



## Proof (ctd.)

**Inductive case:**  $\pi = o_1 \dots o_{n+1}$

By the induction hypothesis,  $o_1^+ \dots o_n^+$  is applicable in  $s'$ , and  $t' = \text{app}_{o_1^+ \dots o_n^+}(s')$  dominates  $t = \text{app}_{o_1 \dots o_n}(s)$ .

Let  $o := o_{n+1} = \langle \chi, e \rangle$  and  $o^+ = \langle \chi, e^+ \rangle$ . By assumption,  $o$  is applicable in  $t$ , and thus  $t \models \chi$ . By the domination lemma, we get  $t' \models \chi$  and hence  $o^+$  is applicable in  $t'$ . Therefore,  $\pi^+$  is applicable in  $s'$ .

Because  $o$  is in positive normal form, all effect conditions satisfied by  $t$  are also satisfied by  $t'$  (by the domination lemma). Therefore,  $([e]_t \cap A) \subseteq [e^+]_{t'}$  (where  $A$  is the set of state variables, or positive literals).

We get

$$\text{on}(\text{app}_{\pi}(s)) \subseteq \text{on}(t) \cup ([e]_t \cap A) \subseteq \text{on}(t') \cup [e^+]_{t'} = \text{on}(\text{app}_{\pi^+}(s')),$$

Obtaining  
heuristics

Relaxed  
planning  
tasks

Definition

The relaxation  
lemma

Greedy algorithm

Optimality

Discussion

Summary



## Corollary (relaxation leads to dominance and preserves plans)

*Let  $\pi$  be an operator sequence that is applicable in state  $s$ . Then  $\pi^+$  is applicable in  $s$  and  $app_{\pi^+}(s)$  dominates  $app_{\pi}(s)$ . If  $\pi$  is a plan for  $\Pi$ , then  $\pi^+$  is a plan for  $\Pi^+$ .*

## Proof.

Apply relaxation lemma with  $s' = s$ .

- ⇒ Relaxations of plans are relaxed plans.
- ⇒ Relaxations are no harder to solve than original task.
- ⇒ Optimal relaxed plans are never longer than optimal plans for original tasks.

Obtaining heuristics

Relaxed planning tasks

Definition

The relaxation lemma

Greedy algorithm

Optimality

Discussion

Summary



## Corollary (relaxation preserves dominance)

*Let  $s$  be a state, let  $s'$  be a state that dominates  $s$ , and let  $\pi^+$  be a relaxed operator sequence applicable in  $s$ . Then  $\pi^+$  is applicable in  $s'$  and  $app_{\pi^+}(s')$  dominates  $app_{\pi^+}(s)$ .*

## Proof.

Apply relaxation lemma with  $\pi^+$  for  $\pi$ , noting that  $(\pi^+)^+ = \pi^+$ .

- $\rightsquigarrow$  If there is a relaxed plan starting from state  $s$ , the same plan can be used starting from a dominating state  $s'$ .
- $\rightsquigarrow$  Making a transition to a dominating state never hurts in relaxed planning tasks.

Obtaining  
heuristics

Relaxed  
planning  
tasks

Definition

The relaxation  
lemma

Greedy algorithm

Optimality

Discussion

Summary

# Monotonicity of relaxed planning tasks



We need one final property before we can provide an algorithm for solving relaxed planning tasks.

## Lemma (monotonicity)

*Let  $o^+ = \langle \chi, e^+ \rangle$  be a relaxed operator and let  $s$  be a state in which  $o^+$  is applicable.*

*Then  $app_{o^+}(s)$  dominates  $s$ .*

## Proof.

Since relaxed operators only have positive effects, we have  $on(s) \subseteq on(s) \cup [e^+]_s = on(app_{o^+}(s))$ .

$\rightsquigarrow$  Together with our previous results, this means that making a transition in a relaxed planning task **never** hurts.

Obtaining  
heuristics

Relaxed  
planning  
tasks

Definition

The relaxation  
lemma

Greedy algorithm

Optimality

Discussion

Summary

# Greedy algorithm for relaxed planning tasks



The relaxation and monotonicity lemmas suggest the following algorithm for solving relaxed planning tasks:

## Greedy planning algorithm for $\langle A, I, O^+, \gamma \rangle$

$s := I$

$\pi^+ := \varepsilon$

**forever:**

**if**  $s \models \gamma$ :

**return**  $\pi^+$

**else if** there is an operator  $o^+ \in O^+$  applicable in  $s$

    with  $app_{o^+}(s) \neq s$ :

    Append such an operator  $o^+$  to  $\pi^+$ .

$s := app_{o^+}(s)$

**else:**

**return** unsolvable

Obtaining  
heuristics

Relaxed  
planning  
tasks

Definition

The relaxation  
lemma

**Greedy algorithm**

Optimality

Discussion

Summary



The algorithm is **sound**:

- If it returns a plan, this is indeed a correct solution.
- If it returns “unsolvable”, the task is indeed unsolvable
  - Upon termination, there clearly is no relaxed plan from  $s$ .
  - By iterated application of the monotonicity lemma,  $s$  dominates  $l$ .
  - By the relaxation lemma, there is no solution from  $l$ .

What about **completeness** (termination) and **runtime**?

- Each iteration of the loop adds at least one atom to  $on(s)$ .
- This guarantees termination after at most  $|A|$  iterations.
- Thus, the algorithm can clearly be implemented to run in polynomial time.
  - A good implementation runs in  $O(\|\Pi\|)$ .

Obtaining  
heuristics

Relaxed  
planning  
tasks

Definition  
The relaxation  
lemma

Greedy algorithm

Optimality  
Discussion

Summary



We can apply the greedy algorithm within heuristic search:

- In a search node  $\sigma$ , solve the relaxation of the planning task with  $state(\sigma)$  as the initial state.
- Set  $h(\sigma)$  to the length of the generated relaxed plan.

Is this an **admissible** heuristic?

- Yes if the relaxed plans are **optimal** (due to the plan preservation corollary).
- However, usually they are not, because our greedy planning algorithm is very poor.

(What about safety? Goal-awareness? Consistency?)

Obtaining  
heuristics

Relaxed  
planning  
tasks

Definition

The relaxation  
lemma

Greedy algorithm

**Optimality**

Discussion

Summary



# The set cover problem



To obtain an admissible heuristic, we need to generate optimal relaxed plans. Can we do this efficiently?

This question is related to the following problem:

## Problem (set cover)

*Given: a finite set  $U$ , a collection of subsets  $C = \{C_1, \dots, C_n\}$  with  $C_i \subseteq U$  for all  $i \in \{1, \dots, n\}$ , and a natural number  $K$ .*

*Question: Does there exist a set cover of size at most  $K$ , i. e., a subcollection  $S = \{S_1, \dots, S_m\} \subseteq C$  with  $S_1 \cup \dots \cup S_m = U$  and  $m \leq K$ ?*

The following is a classical result from complexity theory:

## Theorem (Karp 1972)

*The set cover problem is NP-complete.*

Obtaining  
heuristics

Relaxed  
planning  
tasks

Definition

The relaxation  
lemma

Greedy algorithm

Optimality

Discussion

Summary



## Theorem (optimal relaxed planning is hard)

*The problem of deciding whether a given relaxed planning task has a plan of length at most  $K$  is NP-complete.*

## Proof.

For **membership in NP**, guess a plan and verify. It is sufficient to check plans of length at most  $|A|$ , so this can be done in nondeterministic polynomial time.

For **hardness**, we reduce from the set cover problem.

Obtaining  
heuristics

Relaxed  
planning  
tasks

Definition  
The relaxation  
lemma  
Greedy algorithm

**Optimality**  
Discussion

Summary



## Proof (ctd.)

Given a set cover instance  $\langle U, C, K \rangle$ , we generate the following relaxed planning task  $\Pi^+ = \langle A, I, O^+, \gamma \rangle$ :

- $A = U$
- $I = \{a \mapsto 0 \mid a \in A\}$
- $O^+ = \{ \langle T, \bigwedge_{a \in C_i} a \rangle \mid C_i \in C \}$
- $\gamma = \bigwedge_{a \in U} a$

If  $S$  is a set cover, the corresponding operators form a plan. Conversely, each plan induces a set cover by taking the subsets corresponding to the operators. There exists a plan of length at most  $K$  iff there exists a set cover of size  $K$ .

Moreover,  $\Pi^+$  can be generated from the set cover instance in polynomial time, so this is a polynomial reduction. □

Obtaining  
heuristics

Relaxed  
planning  
tasks

Definition

The relaxation  
lemma

Greedy algorithm

**Optimality**

Discussion

Summary



How can we use relaxations for heuristic planning in practice?

Different possibilities:

- Implement an **optimal planner** for relaxed planning tasks and use its solution lengths as an estimate, even though it is NP-hard.  
     $\rightsquigarrow$   **$h^+$  heuristic**
- Do not actually solve the relaxed planning task, but compute an estimate of its difficulty in a different way.  
     $\rightsquigarrow$   **$h_{\max}$  heuristic,  $h_{\text{add}}$  heuristic,  $h_{\text{LM-cut}}$  heuristic**
- Compute a solution for relaxed planning tasks which is not necessarily optimal, but “reasonable”.  
     $\rightsquigarrow$   **$h_{\text{FF}}$  heuristic**

Obtaining  
heuristics

Relaxed  
planning  
tasks

Definition  
The relaxation  
lemma  
Greedy algorithm  
Optimality  
Discussion

Summary



- Two general methods for coming up with heuristics:
  - **relaxation**: solve a **less constrained** problem
  - **abstraction**: solve a **small** problem
- Here, we consider the **delete relaxation**, which requires tasks in positive normal form and ignores delete effects.
- Delete-relaxed tasks have a **domination** property: it is always beneficial to make more fluents true.
- They also have a **monotonicity** property: applying operators leads to dominating states.
- Because of these two properties, **finding some relaxed plan** greedily is **easy** (polynomial).
- For an informative heuristic, we would ideally want to find **optimal relaxed plans**. This is **NP-complete**.

Obtaining  
heuristics

Relaxed  
planning  
tasks

Summary