

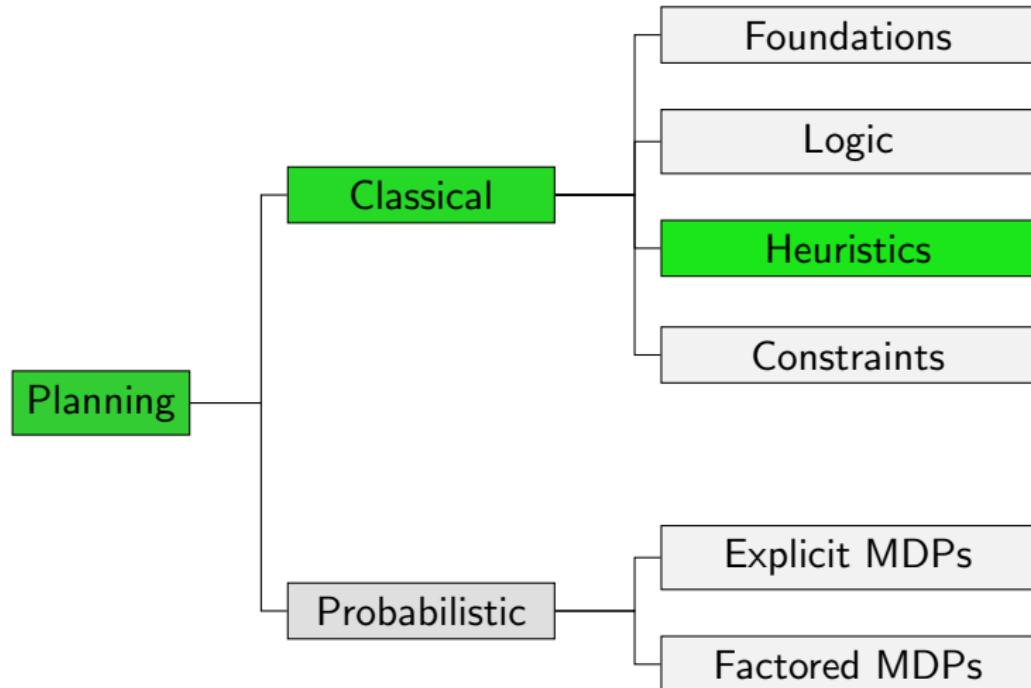
# Planning and Optimization

## C2. Delete Relaxation: Properties of Relaxed Planning Tasks

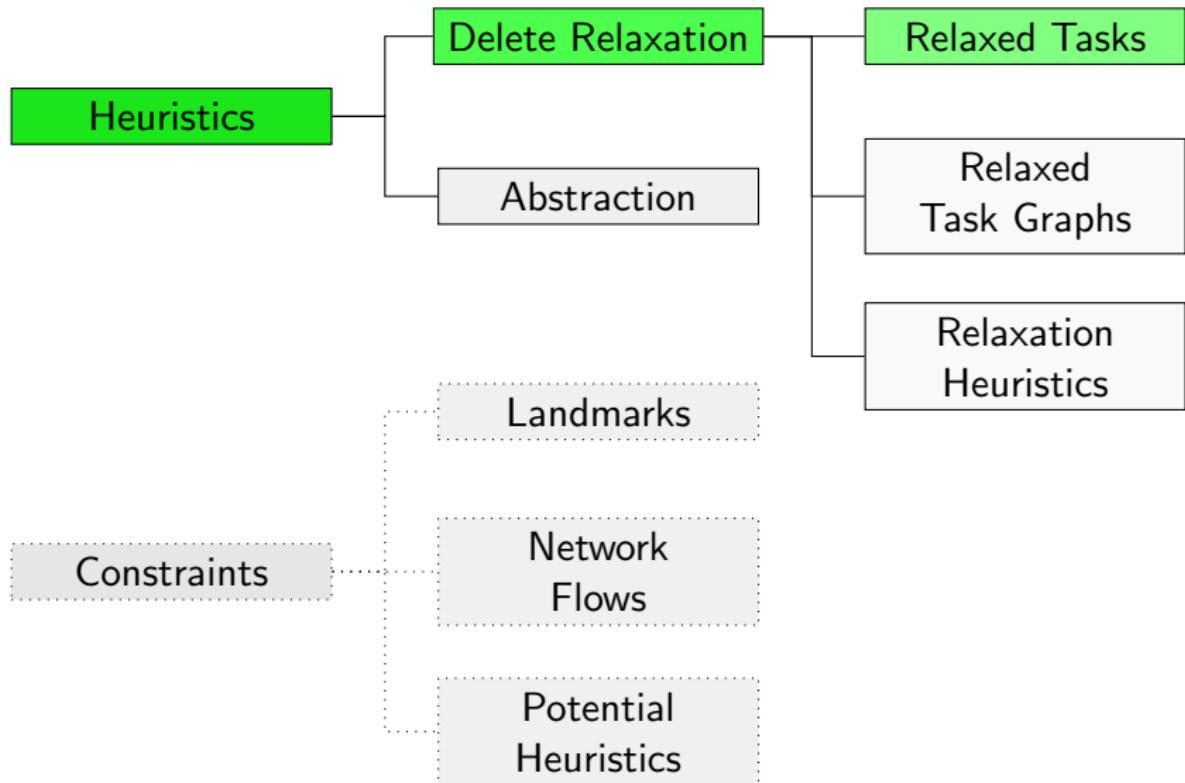
Malte Helmert and Gabriele Röger

Universität Basel

# Content of this Course



# Content of this Course: Heuristics



# The Domination Lemma

# On-Set and Dominating States

## Definition (On-Set)

The **on-set** of a valuation  $s$  is the set of propositional variables that are true in  $s$ , i.e.,  $on(s) = s^{-1}(\{\mathbf{T}\})$ .

~~ for **states** of propositional planning tasks:  
states can be viewed as **sets** of (true) state variables

## Definition (Dominate)

A valuation  $s'$  **dominates** a valuation  $s$  if  $on(s) \subseteq on(s')$ .

~~ all state variables true in  $s$  are also true in  $s'$

# Domination Lemma (1)

## Lemma (Domination)

*Let  $s$  and  $s'$  be valuations of a set of propositional variables  $V$ , and let  $\chi$  be a propositional formula over  $V$  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  $s \not\models \perp$ .

...

## Domination Lemma (2)

Proof (continued).

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

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

- **Inductive case  $\chi = \chi_1 \vee \chi_2$ :** analogous



# The Relaxation Lemma

# Add Sets and Delete Sets

## Definition (Add Set and Delete Set for an Effect)

Consider a propositional planning task with state variables  $V$ .

Let  $e$  be an effect over  $V$ , and let  $s$  be a state over  $V$ .

The **add set** of  $e$  in  $s$ , written  $\text{addset}(e, s)$ ,

and the **delete set** of  $e$  in  $s$ , written  $\text{delset}(e, s)$ ,

are defined as the following sets of state variables:

$$\text{addset}(e, s) = \{v \in V \mid s \models \text{effcond}(v, e)\}$$

$$\text{delset}(e, s) = \{v \in V \mid s \models \text{effcond}(\neg v, e)\}$$

**Note:** For all states  $s$  and operators  $o$  applicable in  $s$ , we have  $\text{on}(s[o]) = (\text{on}(s) \setminus \text{delset}(\text{eff}(o), s)) \cup \text{addset}(\text{eff}(o), s)$ .

# Relaxation Lemma

For this and the following chapters on delete relaxation,  
we assume implicitly that we are working with  
propositional planning tasks in positive normal form.

## Lemma (Relaxation)

Let  $s$  be a state, and let  $s'$  be a state that dominates  $s$ .

- ① If  $o$  is an operator applicable in  $s$ ,  
then  $o^+$  is applicable in  $s'$  and  $s'[\![o^+]\!]$  dominates  $s[\![o]\!]$ .
- ② If  $\pi$  is an operator sequence applicable in  $s$ ,  
then  $\pi^+$  is applicable in  $s'$  and  $s'[\![\pi^+]\!]$  dominates  $s[\![\pi]\!]$ .
- ③ If additionally  $\pi$  leads to a goal state from state  $s$ ,  
then  $\pi^+$  leads to a goal state from state  $s'$ .

# Proof of Relaxation Lemma (1)

## Proof.

Let  $V$  be the set of state variables.

**Part 1:** Because  $o$  is applicable in  $s$ , we have  $s \models \text{pre}(o)$ .

Because  $\text{pre}(o)$  is negation-free and  $s'$  dominates  $s$ ,  
we get  $s' \models \text{pre}(o)$  from the domination lemma.

Because  $\text{pre}(o^+) = \text{pre}(o)$ , this shows that  $o^+$  is applicable in  $s'$ .

...

## Proof of Relaxation Lemma (2)

Proof (continued).

To prove that  $s'[\![o^+]\!]$  dominates  $s[\![o]\!]$ ,  
we first compare the relevant add sets:

$$\begin{aligned} \text{addset}(\text{eff}(o), s) &= \{v \in V \mid s \models \text{effcond}(v, \text{eff}(o))\} \\ &= \{v \in V \mid s \models \text{effcond}(v, \text{eff}(o^+))\} \quad (1) \end{aligned}$$

$$\begin{aligned} &\subseteq \{v \in V \mid s' \models \text{effcond}(v, \text{eff}(o^+))\} \quad (2) \\ &= \text{addset}(\text{eff}(o^+), s'), \end{aligned}$$

where (1) uses  $\text{effcond}(v, \text{eff}(o)) \equiv \text{effcond}(v, \text{eff}(o^+))$   
and (2) uses the dominance lemma (note that effect conditions  
are negation-free for operators in positive normal form). ...

# Proof of Relaxation Lemma (3)

Proof (continued).

We then get:

$$\begin{aligned}on(s\llbracket o \rrbracket) &= (on(s) \setminus delset(eff(o), s)) \cup addset(eff(o), s) \\&\subseteq on(s) \cup addset(eff(o), s) \\&\subseteq on(s') \cup addset(eff(o^+), s') \\&= on(s'\llbracket o^+ \rrbracket),\end{aligned}$$

and thus  $s'\llbracket o^+ \rrbracket$  dominates  $s\llbracket o \rrbracket$ .

This concludes the proof of Part 1.

...

## Proof of Relaxation Lemma (4)

Proof (continued).

Part 2: by induction over  $n = |\pi|$

Base case:  $\pi = \langle \rangle$

The empty plan is trivially applicable in  $s'$ , and  $s'[\langle \rangle^+] = s'$  dominates  $s[\langle \rangle] = s$  by prerequisite.

Inductive case:  $\pi = \langle o_1, \dots, o_{n+1} \rangle$

By the induction hypothesis,  $\langle o_1^+, \dots, o_n^+ \rangle$  is applicable in  $s'$ , and  $t' = s'[\langle o_1^+, \dots, o_n^+ \rangle]$  dominates  $t = s[\langle o_1, \dots, o_n \rangle]$ .

Also,  $o_{n+1}$  is applicable in  $t$ .

Using Part 1,  $o_{n+1}^+$  is applicable in  $t'$  and  $s'[\pi^+] = t'[\langle o_{n+1}^+ \rangle]$  dominates  $s[\pi] = t[\langle o_{n+1} \rangle]$ .

This concludes the proof of Part 2.

...

## Proof of Relaxation Lemma (5)

Proof (continued).

Part 3: Let  $\gamma$  be the goal formula.

From Part 2, we obtain that  $t' = s'[\pi^+]$  dominates  $t = s[\pi]$ .  
By prerequisite,  $t$  is a goal state and hence  $t \models \gamma$ .

Because the task is in positive normal form,  $\gamma$  is negation-free, and hence  $t' \models \gamma$  because of the domination lemma.

Therefore,  $t'$  is a goal state. □

# Further Properties

# Further Properties of Delete Relaxation

- The relaxation lemma is the main technical result that we will use to study delete relaxation.
- Next, we derive some further properties of delete relaxation that will be useful for us.
- Two of these are direct consequences of the relaxation lemma.

# Consequences of the Relaxation Lemma (1)

## Corollary (Relaxation Preserves Plans and Leads to Dominance)

Let  $\pi$  be an operator sequence that is applicable in state  $s$ .

Then  $\pi^+$  is applicable in  $s$  and  $s[\pi^+]$  dominates  $s[\pi]$ .

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.
- ~~> Delete relaxation is no harder to solve than original task.
- ~~> Optimal relaxed plans are never more expensive than optimal plans for original tasks.

## Consequences of the Relaxation Lemma (2)

### 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  $s'[\pi^+]$  dominates  $s[\pi^+]$ .

### Proof.

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



- ~ If there is a relaxed plan starting from state  $s$ , the same plan can be used starting from a dominating state  $s'$ .
- ~ Dominating states are always “better” in relaxed tasks.

# Monotonicity of Relaxed Planning Tasks

## Lemma (Monotonicity)

*Let  $s$  be a state in which relaxed operator  $o^+$  is applicable.  
Then  $s\llbracket o^+ \rrbracket$  dominates  $s$ .*

## Proof.

Since relaxed operators only have positive effects,  
we have  $on(s) \subseteq on(s) \cup addset(eff(o^+), s) = on(s\llbracket o^+ \rrbracket)$ . □

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

# Finding Relaxed Plans

Using the theory we developed, we are now ready to study the problem of **finding plans** for **relaxed planning tasks**.

# Greedy Algorithm

# Greedy Algorithm for Relaxed Planning Tasks

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

## Greedy Planning Algorithm for $\langle V, I, O^+, \gamma \rangle$

**$s := I$**

**$\pi^+ := \langle \rangle$**

**loop forever:**

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

**return**  $\pi^+$

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

with  $s[o^+] \neq s$ :

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

$s := s[o^+]$

**else:**

**return** unsolvable

# Correctness of the Greedy Algorithm

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  $I$ .
  - By the relaxation lemma, there is no solution from  $I$ .

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  $|V|$  iterations.
- Thus, the algorithm can clearly be implemented to run in polynomial time.
  - A good implementation runs in  $O(\|\Pi\|)$ .

# Using the Greedy Algorithm as a Heuristic

We can apply the greedy algorithm within heuristic search:

- When evaluating a state  $s$  in progression search, solve relaxation of planning task with initial state  $s$ .
- When evaluating a subgoal  $\varphi$  in regression search, solve relaxation of planning task with goal  $\varphi$ .
- Set  $h(s)$  to the cost 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 relaxed planning algorithm is very poor.

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

# Summary

# Summary

- Delete relaxation is a **simplification** in the sense that it is never harder to solve a relaxed task than the original one.
- Delete-relaxed tasks have a **domination** property: it is always beneficial to make more state variables true.
- Because of their **monotonicity** property, delete-relaxed tasks can be solved in polynomial time by a greedy algorithm.
- However, the solution quality of this algorithm is poor.