

# Planning and Optimization

## C3. Delete Relaxation: Hardness of Optimal Planning & AND/OR Graphs

Gabriele Röger and Thomas Keller

Universität Basel

October 22, 2018

# Planning and Optimization

October 22, 2018 — C3. Delete Relaxation: Hardness of Optimal Planning & AND/OR Graphs

C3.1 Optimal Relaxed Plans

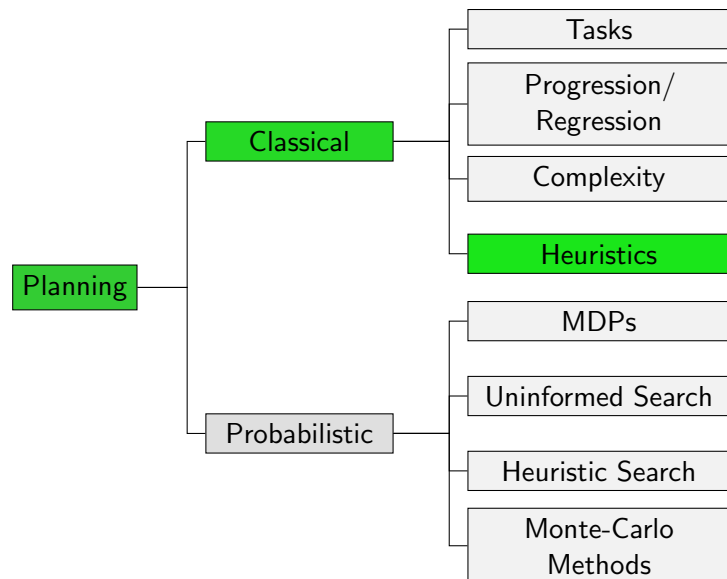
C3.2 AND/OR Graphs

C3.3 Forced Nodes

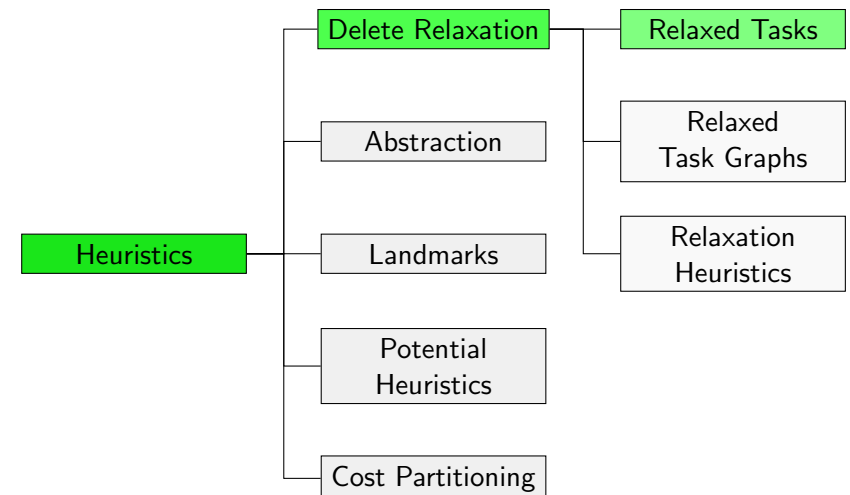
C3.4 Most/Least Conservative Valuations

C3.5 Summary

## Content of this Course



## Content of this Course: Heuristics



# The Story So Far

- ▶ A general way to come up with heuristics is to solve a **simplified** version of the real problem.
- ▶ **delete relaxation**: given a task in positive normal form, discard all delete effects
- ▶ A simple greedy algorithm **solves relaxed tasks efficiently** but usually generates plans of **poor quality**.

How hard is it to find **optimal** plans?

# C3.1 Optimal Relaxed Plans

# The Set Cover Problem

To obtain an admissible heuristic, we must compute 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:** Is there 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.*

# Complexity of Optimal Relaxed Planning (1)

**Theorem (Complexity of Optimal Relaxed Planning)**  
*The BCPLANEX problem restricted to delete-relaxed planning tasks is NP-complete.*

**Proof.**  
 For **membership in NP**, guess a plan and verify.  
 It is sufficient to check plans of length at most  $|V|$  where  $V$  is the set of state variables, so this can be done in nondeterministic polynomial time.  
 For **hardness**, we reduce from the set cover problem. ...

## Complexity of Optimal Relaxed Planning (2)

Proof (continued).

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

- ▶  $V = U$
- ▶  $I = \{v \mapsto \mathbf{F} \mid v \in V\}$
- ▶  $O^+ = \{\langle \top, \bigwedge_{v \in C_i} v, 1 \rangle \mid C_i \in C\}$
- ▶  $\gamma = \bigwedge_{v \in U} v$

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 cost 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.  $\square$

## C3.2 AND/OR Graphs

## Using Relaxations in Practice

How can we use relaxations for heuristic planning in practice?

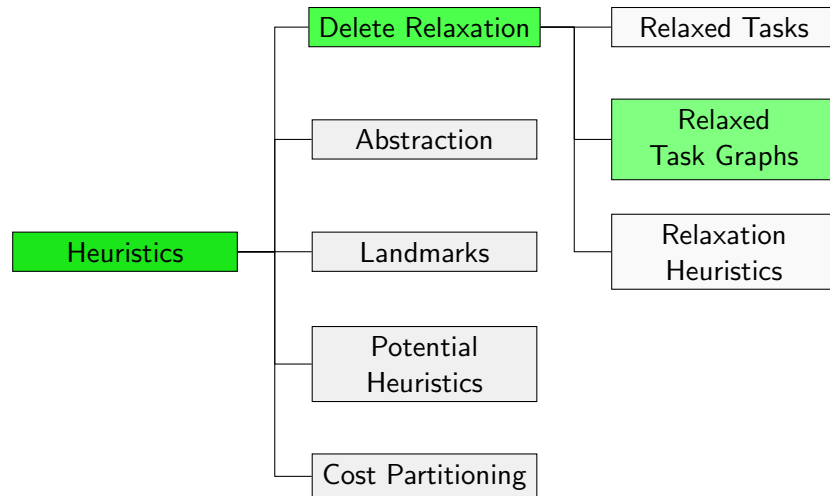
Different possibilities:

- ▶ Implement an **optimal planner** for relaxed planning tasks and use its solution costs as estimates, even though optimal relaxed planning is NP-hard.  
 $\rightsquigarrow$   **$h^+$  heuristic**
- ▶ Do not actually solve the relaxed planning task, but compute an approximation of its solution cost.  
 $\rightsquigarrow$   **$h^{\max}$  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**

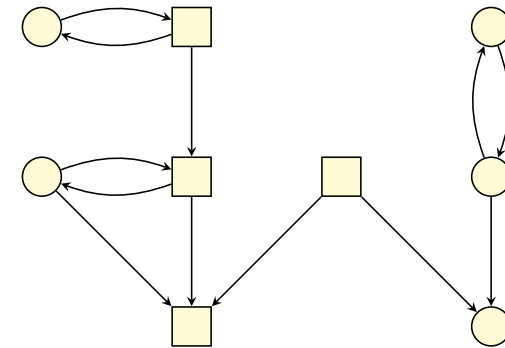
## AND/OR Graphs: Motivation

- ▶ Most relaxation heuristics we will consider can be understood in terms of computations on graphical structures called **AND/OR graphs**.
- ▶ We now introduce AND/OR graphs and study some of their major properties.
- ▶ In the next chapter, we will relate AND/OR graphs to relaxed planning tasks.

## Content of this Course: Heuristics



## AND/OR Graph Example



## AND/OR Graphs

### Definition (AND/OR Graph)

An **AND/OR graph**  $\langle N, A, type \rangle$  is a directed graph  $\langle N, A \rangle$  with a node label function  $type: N \rightarrow \{\wedge, \vee\}$  partitioning nodes into

- ▶ **AND nodes** ( $type(v) = \wedge$ ) and
- ▶ **OR nodes** ( $type(v) = \vee$ ).

We write  $succ(n)$  for the successors of node  $n \in N$ , i.e.,  $succ(n) = \{n' \in N \mid \langle n, n' \rangle \in A\}$ .

**Note:** We draw AND nodes as squares and OR nodes as circles.

## AND/OR Graph Valuations

### Definition (Consistent Valuations of AND/OR Graphs)

Let  $G$  be an AND/OR graph with nodes  $N$ .

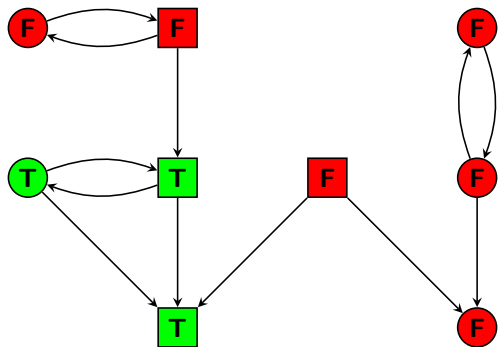
A **valuation** or **truth assignment** of  $G$  is a valuation  $\alpha: N \rightarrow \{\mathbf{T}, \mathbf{F}\}$ , treating the nodes as propositional variables.

We say that  $\alpha$  is **consistent** if

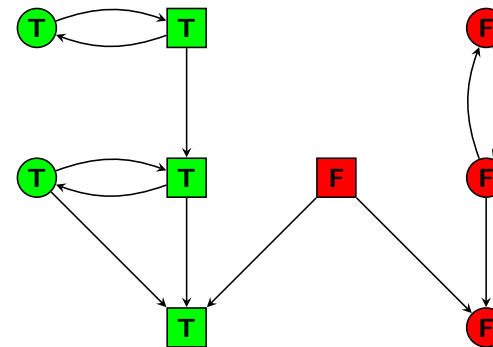
- ▶ for all AND nodes  $n \in N$ :  $\alpha \models n$  iff  $\alpha \models \bigwedge_{n' \in succ(n)} n'$ .
- ▶ for all OR nodes  $n \in N$ :  $\alpha \models n$  iff  $\alpha \models \bigvee_{n' \in succ(n)} n'$ .

Note that  $\bigwedge_{n' \in \emptyset} n' = \mathbf{T}$  and  $\bigvee_{n' \in \emptyset} n' = \perp$ .

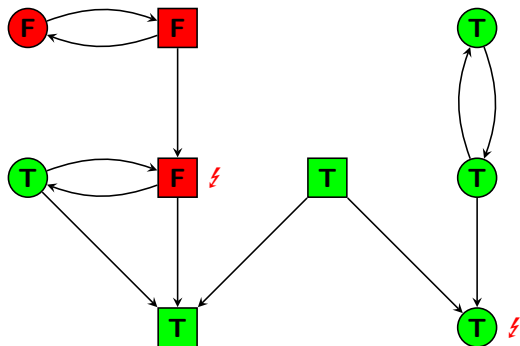
## Example: A Consistent Valuation



## Example: Another Consistent Valuation



## Example: An Inconsistent Valuation



## How Do We Find Consistent Valuations?

If we want to use valuations of AND/OR graphs algorithmically, a number of questions arise:

- ▶ Do consistent valuations **exist** for every AND/OR graph?
- ▶ Are they **unique**?
- ▶ If not, how are different consistent valuations **related**?
- ▶ Can consistent valuations be **computed efficiently**?

Our example shows that the answer to the second question is “no”. In the rest of this chapter, we address the remaining questions.

## C3.3 Forced Nodes

## Forced Nodes

### Definition (Forced True/False Nodes)

Let  $G$  be an AND/OR graph.

A node  $n$  of  $G$  is called **forced true** if  $\alpha(n) = \mathbf{T}$  for all consistent valuations  $\alpha$  of  $G$ .

A node  $n$  of  $G$  is called **forced false** if  $\alpha(n) = \mathbf{F}$  for all consistent valuations  $\alpha$  of  $G$ .

How can we efficiently determine that nodes are forced true/false?

$\rightsquigarrow$  We begin by looking at some simple rules.

## Rules for Forced True Nodes

### Proposition (Rules for Forced True Nodes)

Let  $n$  be a node in an AND/OR graph.

**Rule T-( $\wedge$ ):** If  $n$  is an AND node and *all* of its successors are forced true, then  $n$  is forced true.

**Rule T-( $\vee$ ):** If  $n$  is an OR node and *at least one* of its successors is forced true, then  $n$  is forced true.

#### Remarks:

- ▶ These are “if, then” rules.  
Would they also be correct as “if and only if” rules?
- ▶ For the first rule, it is easy to see that the answer is “yes”.
- ▶ For the second rule, this is not so easy. (Why not?)

## Rules for Forced False Nodes

### Proposition (Rules for Forced False Nodes)

Let  $n$  be a node in an AND/OR graph.

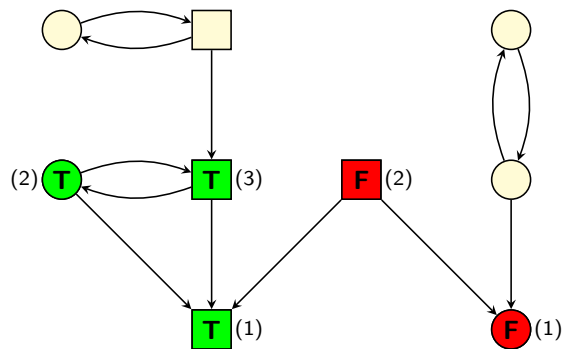
**Rule F-( $\wedge$ ):** If  $n$  is an AND node and *at least one* of its successors is forced false, then  $n$  is forced false.

**Rule F-( $\vee$ ):** If  $n$  is an OR node and *all* of its successors are forced false, then  $n$  is forced false.

#### Remarks:

- ▶ Analogous comments as in the case of forced true nodes apply.
- ▶ This time, it is the first rule for which it is not obvious if a corresponding “if and only if” rule would be correct.

## Example: Applying the Rules for Forced Nodes



## Completeness of Rules for Forced Nodes

### Theorem

If  $n$  is a node in an AND/OR graph that is forced true, then this can be derived by a sequence of applications of Rule  $\mathbf{T}-(\wedge)$  and Rule  $\mathbf{T}-(\vee)$ .

### Theorem

If  $n$  is a node in an AND/OR graph that is forced false, then this can be derived by a sequence of applications of Rule  $\mathbf{F}-(\wedge)$  and Rule  $\mathbf{F}-(\vee)$ .

We prove the result for **forced true** nodes.

The result for forced false nodes can be proved analogously.

## Completeness of Rules for Forced Nodes: Proof (1)

### Proof.

- ▶ Let  $\alpha$  be a valuation where  $\alpha(n) = \mathbf{T}$  iff there exists a sequence  $\rho_n$  of applications of Rules  $\mathbf{T}-(\wedge)$  and Rule  $\mathbf{T}-(\vee)$  that derives that  $n$  is forced true.
- ▶ Because the rules are monotonic, there exists a sequence  $\rho$  of rule applications that derives that  $n$  is forced true for **all**  $n \in on(\alpha)$ . (Just concatenate all  $\rho_n$  to form  $\rho$ .)
- ▶ By the correctness of the rules, we know that all nodes reached by  $\rho$  are forced true. It remains to show that none of the nodes **not** reached by  $\rho$  is forced true.
- ▶ We prove this by showing that  $\alpha$  is **consistent**, and hence no nodes with  $\alpha(n) = \mathbf{F}$  can be forced true.

...

## Completeness of Rules for Forced Nodes: Proof (2)

### Proof (continued).

#### Case 1: nodes $n$ with $\alpha(n) = \mathbf{T}$

- ▶ In this case,  $\rho$  must have reached  $n$  in one of the derivation steps. Consider this derivation step.
- ▶ If  $n$  is an AND node,  $\rho$  must have reached all successors of  $n$  in previous steps, and hence  $\alpha(n') = \mathbf{T}$  for all successors  $n'$ .
- ▶ If  $n$  is an OR node,  $\rho$  must have reached at least one successor of  $n$  in a previous step, and hence  $\alpha(n') = \mathbf{T}$  for at least one successor  $n'$ .
- ▶ In both cases,  $\alpha$  is consistent for node  $n$ .

...

## Completeness of Rules for Forced Nodes: Proof (3)

Proof (continued).

Case 2: nodes  $n$  with  $\alpha(n) = \mathbf{F}$

- ▶ In this case, by definition of  $\alpha$  no sequence of derivation steps reaches  $n$ . In particular,  $\rho$  does not reach  $n$ .
- ▶ If  $n$  is an AND node, there must exist some  $n' \in \text{succ}(n)$  which  $\rho$  does not reach. Otherwise,  $\rho$  could be extended using Rule  $\mathbf{T}-(\wedge)$  to reach  $n$ . Hence,  $\alpha(n') = \mathbf{F}$  for some  $n' \in \text{succ}(n)$ .
- ▶ If  $n$  is an OR node, there cannot exist any  $n' \in \text{succ}(n)$  which  $\rho$  reaches. Otherwise,  $\rho$  could be extended using Rule  $\mathbf{T}-(\vee)$  to reach  $n$ . Hence,  $\alpha(n') = \mathbf{F}$  for all  $n' \in \text{succ}(n)$ .
- ▶ In both cases,  $\alpha$  is consistent for node  $n$ .

□

## Remarks on Forced Nodes

Notes:

- ▶ The theorem shows that we can compute all forced nodes by applying the rules repeatedly until a fixed point is reached.
- ▶ In particular, this also shows that the order of rule application does not matter: we always end up with the same result.
- ▶ In an efficient implementation, the sets of forced nodes can be computed in linear time in the size of the AND/OR graph.
- ▶ The proof of the theorem also shows that every AND/OR graph has a consistent valuation, as we explicitly construct one in the proof.

## C3.4 Most/Least Conservative Valuations

### Most and Least Conservative Valuation

Definition (Most and Least Conservative Valuation)

Let  $G$  be an AND/OR graph with nodes  $N$ .

The **most conservative valuation**  $\alpha_{\text{mcv}}^G : N \rightarrow \{\mathbf{T}, \mathbf{F}\}$  and the **least conservative valuation**  $\alpha_{\text{lcv}}^G : N \rightarrow \{\mathbf{T}, \mathbf{F}\}$  of  $G$  are defined as:

$$\alpha_{\text{mcv}}^G(n) = \begin{cases} \mathbf{T} & \text{if } n \text{ is forced true} \\ \mathbf{F} & \text{otherwise} \end{cases}$$

$$\alpha_{\text{lcv}}^G(n) = \begin{cases} \mathbf{F} & \text{if } n \text{ is forced false} \\ \mathbf{T} & \text{otherwise} \end{cases}$$

Note:  $\alpha_{\text{mcv}}^G$  is the valuation constructed in the previous proof.



## Properties of Most/Least Conservative Valuations

### Theorem (Properties of Most/Least Conservative Valuations)

Let  $G$  be an AND/OR graph. Then:

- ①  $\alpha_{\text{mcv}}^G$  is consistent.
- ②  $\alpha_{\text{lcv}}^G$  is consistent.
- ③ For all consistent valuations  $\alpha$  of  $G$ ,  
 $\text{on}(\alpha_{\text{mcv}}^G) \subseteq \text{on}(\alpha) \subseteq \text{on}(\alpha_{\text{lcv}}^G)$ .

## Properties of MCV/LCV: Proof

### Proof.

Part 1. was shown in the preceding proof. We showed that the valuation  $\alpha$  considered in this proof is consistent and satisfies  $\alpha(n) = \mathbf{T}$  iff  $n$  is forced true, which implies  $\alpha = \alpha_{\text{mcv}}^G$ .

The proof of Part 2. is analogous, using the rules for forced false nodes instead of forced true nodes.

Part 3 follows directly from the definitions of forced nodes,  $\alpha_{\text{mcv}}^G$  and  $\alpha_{\text{lcv}}^G$ . □

## Properties of MCV/LCV: Consequences

This theorem answers our remaining questions about the existence, uniqueness, structure and computation of consistent valuations:

- ▶ Consistent valuations always exist and can be efficiently computed.
- ▶ All consistent valuations lie between the most and least conservative one.
- ▶ There is a unique consistent valuation iff  $\alpha_{\text{mcv}}^G = \alpha_{\text{lcv}}^G$ , or equivalently iff each node is forced true or forced false.

## C3.5 Summary

## Summary I

- ▶ For an informative heuristic, we would ideally want to find **optimal relaxed plans**.
- ▶ The solution cost of an optimal relaxed plan is the estimate of the  $h^+$  heuristic.
- ▶ However, the bounded-cost plan existence problem for relaxed planning tasks is **NP-complete**.
- ▶ Other relaxation heuristics can be understood in terms of computations on **AND/OR graphs**.

## Summary II

- ▶ **AND/OR graphs** are directed graphs with **AND nodes** and **OR nodes**.
- ▶ We can assign **truth values** to AND/OR graph nodes.
- ▶ Such valuations are called **consistent** if they match the intuitive meaning of “AND” and “OR”.
- ▶ Consistent valuations always exist.
- ▶ Consistent valuations can be computed efficiently.
- ▶ All consistent valuations fall between two extremes:
  - ▶ the **most conservative valuation**, where only nodes that are **forced to be true** are true
  - ▶ the **least conservative valuation**, where all nodes that are **not forced to be false** are true