Planning and Optimization

D6. Delete Relaxation: h^{max} and h^{add}

Malte Helmert and Gabriele Röger

Universität Basel

October 28, 2024

Planning and Optimization

October 28, 2024 — D6. Delete Relaxation: h^{max} and h^{add}

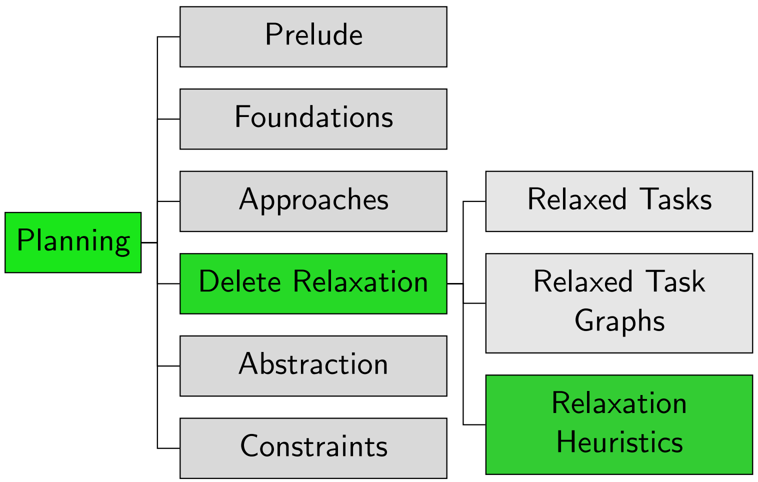
D6.1 Introduction

D6.2 h^{max} and h^{add}

D6.3 Properties of h^{max} and h^{add}

D6.4 Summary

Content of the Course



D6.1 Introduction

Delete Relaxation Heuristics

- In this chapter, we introduce heuristics based on delete relaxation.
- Their basic idea is to propagate information in relaxed task graphs, similar to the previous chapter.
- Unlike the previous chapter, we do not just propagate information about whether a given node is reachable, but estimates how expensive it is to reach the node.

Reminder: Running Example

We will use the same running example as in the previous chapter:
$\Pi = \langle V, I, \{o_1, o_2, o_3, o_4\}, \gamma\rangle$ with

$$
\begin{aligned}
V &= \{a, b, c, d, e, f, g, h\}\\
I &= \{a \mapsto \mathbf{T}, b \mapsto \mathbf{T}, c \mapsto \mathbf{F}, d \mapsto \mathbf{T},\\
&\quad\; e \mapsto \mathbf{F}, f \mapsto \mathbf{F}, g \mapsto \mathbf{F}, h \mapsto \mathbf{F}\}\\
o_1 &= \langle c \vee (a \wedge b), c \wedge ((c \wedge d) \rhd e), 1\rangle\\
o_2 &= \langle \top, f, 2\rangle\\
o_3 &= \langle f, g, 1\rangle\\
o_4 &= \langle f, h, 1\rangle\\
\gamma &= e \wedge (g \wedge h)
\end{aligned}
$$

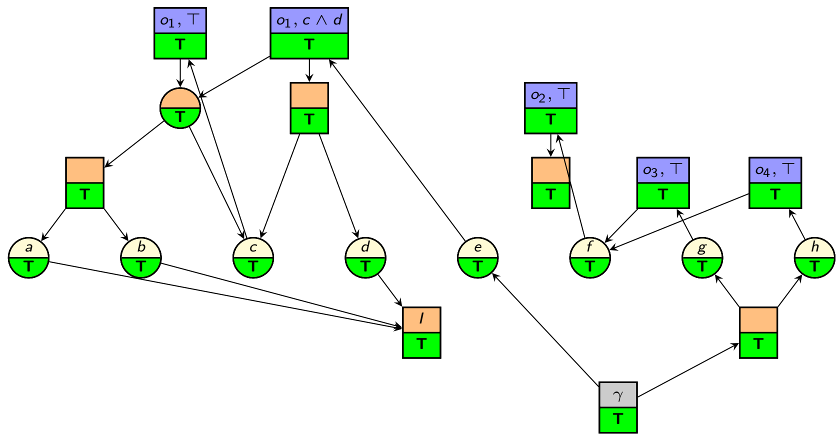
Algorithm for Reachability Analysis (Reminder)

- reachability analysis in RTGs = computing all forced true nodes = computing the most conservative assignment
- Here is an algorithm that achieves this:

Reachability Analysis

Associate a *reachable* attribute with each node.
for all nodes n:
    $n.\mathit{reachable} := \mathbf{false}$
while no fixed point is reached:
    Choose a node n.
    if n is an AND node:
        $n.\mathit{reachable} := \bigwedge_{n' \in succ(n)} n'.\mathit{reachable}$
    if n is an OR node:
        $n.\mathit{reachable} := \bigvee_{n' \in succ(n)} n'.\mathit{reachable}$

Reachability Analysis: Example (Reminder)



D6.2 h^{max} and h^{add}

Associating Costs with RTG Nodes

Basic intuitions for associating costs with RTG nodes:

- To apply an operator, we must pay its cost.
- To make an OR node true, it is sufficient to make one of its successors true.
  - ⇝ Therefore, we estimate the cost of an OR node as the minimum of the costs of its successors.
- To make an AND node true, all its successors must be made true first.
  - ⇝ We can be optimistic and estimate the cost as the maximum of the successor node costs.
  - ⇝ Or we can be pessimistic and estimate the cost as the sum of the successor node costs.
  - ⇝ We will prove later that this is indeed optimistic/pessimistic.

h^{max} Algorithm

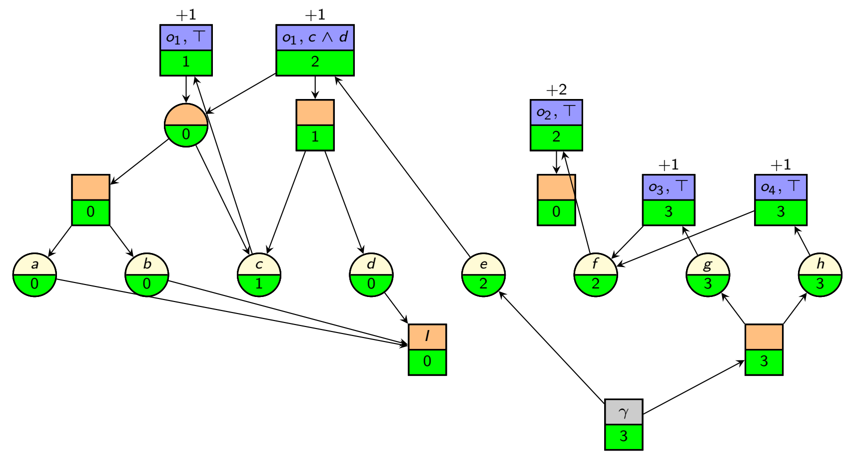
(Differences to reachability analysis algorithm highlighted.)

Computing h^{max} Values

Associate a *cost* attribute with each node.
for all nodes n:
    $n.cost := \infty$
while no fixed point is reached:
    Choose a node n.
    if n is an AND node that is not an effect node:
        $n.cost := \max_{n' \in succ(n)} n'.cost$
    if n is an effect node for operator o:
        $n.cost := cost(o) + \max_{n' \in succ(n)} n'.cost$
    if n is an OR node:
        $n.cost := \min_{n' \in succ(n)} n'.cost$

The overall heuristic value is the cost of the goal node, $n_\gamma.cost$.

h^{max}: Example



⇝ $h^{max}(I) = 3$

h^{add} Algorithm

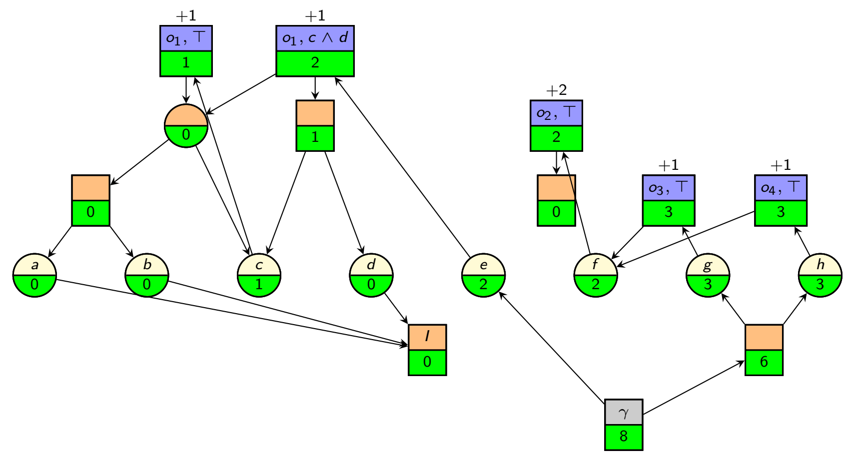
(Differences to h^{max} algorithm highlighted.)

Computing h^{add} Values

Associate a *cost* attribute with each node.
for all nodes n:
  $n.cost := \infty$
while no fixed point is reached:
  Choose a node n.
  if n is an AND node that is not an effect node:
    $n.cost := \sum_{n' \in succ(n)} n'.cost$
  if n is an effect node for operator o:
    $n.cost := cost(o) + \sum_{n' \in succ(n)} n'.cost$
  if n is an OR node:
    $n.cost := \min_{n' \in succ(n)} n'.cost$

The overall heuristic value is the cost of the goal node, $n_\gamma.cost$.

h^{add}: Example



$\rightsquigarrow h^{add}(I) = 8$

h^{max} and h^{add}: Definition

We can now define our first non-trivial efficient planning heuristics:

h^{max} and h^{add} Heuristics

Let $\Pi = \langle V, I, O, \gamma \rangle$ be a propositional planning task in positive normal form.

The h^{max} heuristic value of a state s, written $h^{max}(s)$, is obtained by constructing the RTG for $\Pi_s^+ = \langle V, s, O^+, \gamma \rangle$ and then computing $n_\gamma.cost$ using the h^{max} value algorithm for RTGs.

The h^{add} heuristic value of a state s, written $h^{add}(s)$, is computed in the same way using the h^{add} value algorithm for RTGs.

Notation: we will use the same notation $h^{max}(n)$ and $h^{add}(n)$ for the h^{max}/h^{add} values of RTG nodes

D6.3 Properties of h^{max} and h^{add}

Understanding h^{max} and h^{add}

We want to understand h^{max} and h^{add} better:

- Are they well-defined?
- How can they be efficiently computed?
- Are they safe?
- Are they admissible?
- How do they compare to the optimal solution cost for a delete-relaxed task (h^+)?

Well-Definedness of h^{max} and h^{add} (1)

Are h^{max} and h^{add} well-defined?

- The algorithms for computing h^{max} and h^{add} values do not specify in which order the RTG nodes should be selected.
- It turns out that the order does not affect the final result.
  $\rightsquigarrow$ The h^{max} and h^{add} values are well-defined.
- To show this, we must show
  - that their computation always terminates, and
  - that all executions terminate with the same result.
- For time reasons, we only provide a proof sketch.

Well-Definedness of h^{max} and h^{add} (2)

Theorem

The fixed point algorithms for computing h^{max} and h^{add} values produce a well-defined result.

Proof Sketch.

Let $V_0, V_1, V_2, \dots$ be the vectors of cost values during a given execution of the algorithm.

Termination: Note that $V_i \geq V_{i+1}$ for all i.
It is not hard to prove that each node value can only decrease a finite number of times: first from ∞ to some finite value, and then a finite number of additional times. ...

Well-Definedness of h^{max} and h^{add} (3)

Proof Sketch (continued).

Uniqueness of result: Let $V_0 \geq V_1 \geq V_2 \geq \dots \geq V_n$ be the finite sequence of cost value vectors until termination during a given execution of the algorithm.

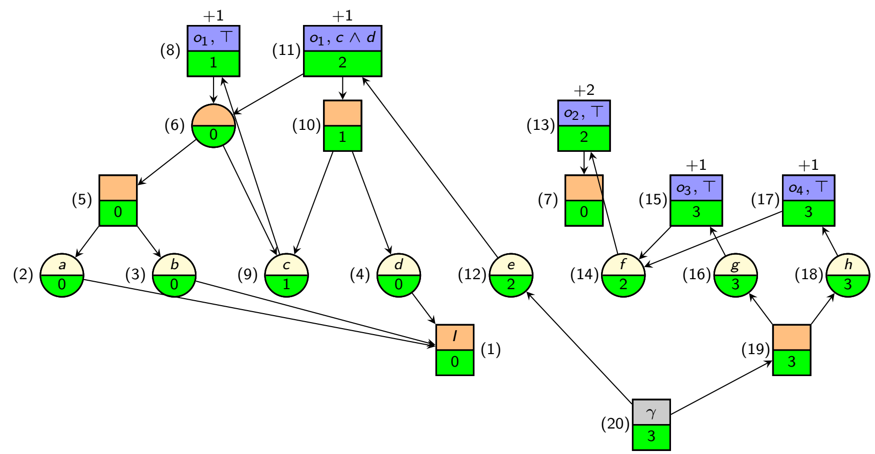
- View the consistency conditions of all nodes (e.g., $n.cost = \min_{n' \in succ(n)} n'.cost$ for all OR nodes n) as a system of equations **E**.
- V_n must be a solution to **E** (otherwise no fixed point is reached with V_n).
- For all $i \in \{0, \dots, n\}$, show by induction over i that $V_i \geq S$ for all solutions S to **E**.
- It follows that V_n is the unique maximum solution to **E** and hence well-defined.

□

Efficient Computation of $h^{\max}$ and h^{add}

- If nodes are poorly chosen, the $h^{\max}/h^{\text{add}}$ algorithm can update the same node many times until it reaches its final value.
- However, there is a simple strategy that prevents this: in every iteration, pick a node with minimum new value among all nodes that can be updated to a new value.
- With this strategy, no node is updated more than once. (We omit the proof, which is not complicated.)
- Using a suitable priority queue data structure, this allows computing the $h^{\max}/h^{\text{add}}$ values of an RTG with nodes N and arcs A in time $O(|N| \log |N| + |A|)$.

$h^{\max}$: Example of Efficient Computation



$\rightsquigarrow h^{\max}(I) = 3$

Efficient Computation of $h^{\max}$ and h^{add}: Remarks

- In the following chapters, we will always assume that we are using this efficient version of the $h^{\max}$ and h^{add} algorithm.
- In particular, we will assume that all reachable nodes of the relaxed task graph are processed exactly once (and all unreachable nodes not at all), so that it makes sense to speak of certain nodes being processed after others etc.

Heuristic Quality of $h^{\max}$ and h^{add}

This leaves us with the questions about the heuristic quality of $h^{\max}$ and h^{add}:

- Are they safe?
- Are they admissible?
- How do they compare to the optimal solution cost for a delete-relaxed task?

It is easy to see that $h^{\max}$ and h^{add} are safe: they assign ∞ iff a node is unreachable in the delete relaxation.

In our running example, it seems that $h^{\max}$ is prone to underestimation and h^{add} is prone to overestimation.

We will study this further in the next chapter.

D6.4 Summary

Summary

- $h^{\max}$ and h^{add} values estimate how expensive it is to reach a state variable, operator effect or formula (e.g., the goal).
- They are computed by propagating cost information in relaxed task graphs:
  - At OR nodes, choose the cheapest alternative.
  - At AND nodes, maximize or sum the successor costs.
  - At effect nodes, also add the operator cost.
- $h^{\max}$ and h^{add} values can serve as heuristics.
- They are well-defined and can be computed efficiently by computing them in order of increasing cost along the RTG.