Towards better relaxed plans

Why does the greedy algorithm compute low-quality plans?
- It may apply many operators which are not goal-directed.

How can this problem be fixed?
- Reaching the goal of a relaxed planning task is most easily achieved with forward search.
- Analyzing relevance of an operator for achieving a goal (or subgoal) is most easily achieved with backward search.

Idea: Use a forward-backward algorithm that first finds a path to the goal greedily, then prunes it to a relevant subplan.

Relaxed plan steps

How to decide which operators to apply in forward direction?
- We avoid such a decision by applying all applicable operators simultaneously.

Definition (plan step)
A plan step is a set of operators \( \omega = \{ (\chi_1, e_1), \ldots, (\chi_n, e_n) \} \).
In the special case of all operators of \( \omega \) being relaxed, we further define:
- Plan step \( \omega \) is applicable in state \( s \) iff \( s \models \chi_i \) for all \( i \in \{1, \ldots, n\} \).
- The result of applying \( \omega \) to \( s \), in symbols \( \text{app}_\omega(s) \), is defined as the state \( s' \) with \( \text{on}(s') = \text{on}(s) \cup \bigcup_{i=1}^n \mathcal{E}_i[s] \).

general semantics for plan steps \( \rightsquigarrow \) much later
Applying relaxed plan steps: examples

In all cases, \( s = \{a \mapsto 0, b \mapsto 0, c \mapsto 1, d \mapsto 0\} \).
- \( \omega = \{(c,a), (b,b)\} \)
- \( \omega = \{(c,a), (c,a \succ b)\} \)
- \( \omega = \{(c,a \land b), (a,b \triangleright d)\} \)
- \( \omega = \{(c,a \land (b \triangleright d)), (c,b \land (a \triangleright d))\} \)

Serializations

Applying a relaxed plan step to a state is related to applying the operators in the step to a state in sequence.

Definition (serializatation)

A serialization of plan step \( \omega = \{o_1^+, \ldots, o_n^+\} \) is a sequence \( o_{\pi(1)}^+, \ldots, o_{\pi(n)}^+ \) where \( \pi \) is a permutation of \( \{1, \ldots, n\} \).

Lemma (conservativeness of plan step semantics)

If \( \omega \) is a plan step applicable in a state \( s \) of a relaxed planning task, then each serialization \( o_1, \ldots, o_n \) of \( \omega \) is applicable in \( s \) and \( \text{app}_{o_1, \ldots, o_n}(s) \) dominates \( \text{app}_\omega(s) \).

- Does equality hold for all/some serialization(s)?
- What if there are no conditional effects?
- What if we allowed general (unrelaxed) planning tasks?

Parallel plans

Definition (parallel plan)

A parallel plan for a relaxed planning task \( \langle A, I, O^+, \gamma \rangle \) is a sequence of plan steps \( \omega_1, \ldots, \omega_n \) of operators in \( O^+ \) with:
- \( s_0 := I \)
- For \( i = 1, \ldots, n \), step \( \omega_i \) is applicable in \( s_{i-1} \) and \( s_i := \text{app}_{\omega_i}(s_{i-1}) \).
- \( s_n \models \gamma \)

Remark: By ordering the operators within each single step arbitrarily, we obtain a (regular, non-parallel) plan.

Forward states, plan steps and sets

Idea: In the forward phase of the heuristic computation, apply plan step with all operators applicable initially, apply plan step with all operators applicable then, and so on.

Definition (forward state/plan step/set)

Let \( \Pi^+ = \langle A, I, O^+, \gamma \rangle \) be a relaxed planning task.
The \( n \)-th forward state, in symbols \( s_n^F \) (\( n \in \mathbb{N}_0 \)),
the \( n \)-th forward plan step, in symbols \( \omega_n^F \) (\( n \in \mathbb{N}_1 \)), and
the \( n \)-th forward set, in symbols \( S_n^F \) (\( n \in \mathbb{N}_0 \)), are defined as:
- \( s_0^F := I \)
- \( \omega_n^F := \{ o \in O^+ \mid o \text{ applicable in } s_{n-1}^F \} \) for all \( n \in \mathbb{N}_1 \)
- \( s_n^F := \text{app}_{\omega_n^F}(s_{n-1}^F) \) for all \( n \in \mathbb{N}_1 \)
- \( S_n^F := \text{on}(s_n^F) \) for all \( n \in \mathbb{N}_0 \)
The max heuristic $h_{\text{max}}$

Definition (parallel forward distance)
The parallel forward distance of a relaxed planning task $\langle A, I, O^+, \gamma \rangle$ is the lowest number $n \in \mathbb{N}_0$ such that $s^F_n \models \gamma$, or $\infty$ if no forward state satisfies $\gamma$.

Remark: The parallel forward distance can be computed in polynomial time. (How?)

Definition (max heuristic $h_{\text{max}}$)
Let $\Pi = \langle A, I, O, \gamma \rangle$ be a planning task in positive normal form, and let $s$ be a state of $\Pi$.

The max heuristic estimate for $s$, $h_{\text{max}}(s)$, is the parallel forward distance of the relaxed planning task $\langle A, s, O^+, \gamma \rangle$.

Remark: $h_{\text{max}}$ is safe, goal-aware, admissible and consistent. (Why?)

So far, so good...

- We have seen how systematic computation of forward states leads to an admissible heuristic estimate.
- However, this estimate is very coarse.
- To improve it, we need to include backward propagation of information.

For this purpose, we use so-called relaxed planning graphs.

2 Relaxed planning graphs

- Introduction
- Construction
- Truth values

AND/OR dags

Definition (AND/OR dag)
An AND/OR dag $\langle V, A, \text{type} \rangle$ is a directed acyclic graph $\langle V, A \rangle$ with a label function $\text{type} : V \rightarrow \{\land, \lor\}$ partitioning nodes into AND nodes ($\text{type}(v) = \land$) and OR nodes ($\text{type}(v) = \lor$).

Note: AND nodes drawn as squares, OR nodes as circles.

Definition (truth values in AND/OR dags)
Let $G = \langle V, A, \text{type} \rangle$ be an AND/OR dag, and let $u \in V$ be a node with successor set $\{v_1, \ldots, v_k\} \subseteq V$.

The (truth) value of $u$, $\text{val}(u)$, is inductively defined as:

- If $\text{type}(u) = \land$, then $\text{val}(u) = \text{val}(v_1) \land \cdots \land \text{val}(v_k)$.
- If $\text{type}(u) = \lor$, then $\text{val}(u) = \text{val}(v_1) \lor \cdots \lor \text{val}(v_k)$.
Relaxed planning graphs

Let $\Pi^+$ be a relaxed planning task, and let $k \in \mathbb{N}_0$.

The relaxed planning graph of $\Pi^+$ for depth $k$, in symbols $\text{RPG}_k(\Pi^+)$, is an AND/OR dag that encodes
- which propositions can be made true in $k$ plan steps, and
- how they can be made true.

Its construction is a bit involved, so we present it in stages.

Running example

As a running example, consider the relaxed planning task $\langle A, I, \{o_1, o_2, o_3, o_4\}, \gamma \rangle$ with

$A = \{a, b, c, d, e, f, g, h\}$

$I = \{a \mapsto 1, b \mapsto 0, c \mapsto 1, d \mapsto 1, e \mapsto 0, f \mapsto 0, g \mapsto 0, h \mapsto 0\}$

$o_1 = \langle b \lor (c \land d), b \land ((a \land b) \triangleright e) \rangle$

$o_2 = \langle \top, f \rangle$

$o_3 = \langle f, g \rangle$

$o_4 = \langle f, h \rangle$

$\gamma = e \land (g \land h)$

Running example: forward sets and plan steps

$I = \{a \mapsto 1, b \mapsto 0, c \mapsto 1, d \mapsto 1, e \mapsto 0, f \mapsto 0, g \mapsto 0, h \mapsto 0\}$

$o_1 = \langle b \lor (c \land d), b \land ((a \land b) \triangleright e) \rangle$

$o_2 = \langle \top, f \rangle, \quad o_3 = \langle f, g \rangle, \quad o_4 = \langle f, h \rangle$

$S^F_0 = \{a, c, d\}$

$\omega^F_1 = \{o_1, o_2\}$

$S^F_1 = \{a, b, c, d, f\}$

$\omega^F_2 = \{o_1, o_2, o_3, o_4\}$

$S^F_2 = \{a, b, c, d, e, f, g, h\}$

$\omega^F_3 = \omega^F_2$

$S^F_3 = S^F_2$ etc.

Components of relaxed planning graphs

A relaxed planning graph consists of four kinds of components:
- **Proposition nodes** represent the truth value of propositions after applying a certain number of plan steps.
- **Idle arcs** represent the fact that state variables, once true, remain true.
- **Operator subgraphs** represent the possibility and effect of applying a given operator in a given plan step.
- **The goal subgraph** represents the truth value of the goal condition after $k$ plan steps.
Let $\Pi^+ = \langle A, I, O^+, \gamma \rangle$ be a relaxed planning task, let $k \in \mathbb{N}_0$.

For each $i \in \{0, \ldots, k\}$, $\text{RPG}_k(\Pi^+)$ contains one proposition layer which consists of:
- a proposition node $a^i$ for each state variable $a \in A$.

Node $a^i$ is an AND node if $i = 0$ and $I \models a$. Otherwise, it is an OR node.

Relaxed planning graph: proposition layers

For each proposition node $a^i$ with $i \in \{1, \ldots, k\}$, $\text{RPG}_k(\Pi^+)$ contains an arc from $a^i$ to $a^{i-1}$ (idle arcs).

Intuition: If a state variable is true in step $i$, one of the possible reasons is that it was already previously true.
Relaxed planning graph: operator subgraphs

For each \( i \in \{1, \ldots, k\} \) and each operator \( o^+ = (\chi, e^+) \in O^\pm \), \( \text{RPG}_k(\Pi^+) \) contains a subgraph called an \textit{operator subgraph} with the following parts:

- one formula node \( n^+_\varphi \) for each formula \( \varphi \) which is a subformula of \( \chi \) or of some effect condition in \( e^+ \):
  - If \( \varphi = a \) for some atom \( a \), \( n^+_\varphi \) is the proposition node \( a^\perp \).
  - If \( \varphi = \top \), \( n^+_\varphi \) is a new AND node without outgoing arcs.
  - If \( \varphi = \bot \), \( n^+_\varphi \) is a new OR node without outgoing arcs.
  - If \( \varphi = (\varphi' \land \varphi''), n^+_\varphi \) is a new AND node with outgoing arcs to \( n^+_\varphi' \) and \( n^+_\varphi'' \).
  - If \( \varphi = (\varphi' \lor \varphi''), n^+_\varphi \) is a new OR node with outgoing arcs to \( n^+_\varphi' \) and \( n^+_\varphi'' \).

Relaxed planning graph: goal subgraph

\( \text{RPG}_k(\Pi^+) \) contains a subgraph called a \textit{goal subgraph} with the following parts:

- for each conditional effect \( (\chi' \supset a) \) in \( e^+ \), an effect node \( o^+_{\chi'} \) (an AND node) with outgoing arcs to the precondition formula node \( n^+_\chi' \) and effect condition formula node \( n^+_\varphi' \), and incoming arc from proposition node \( a^\prime \):
  - unconditional effects \( a \) (effects which are not part of a conditional effect) are treated the same, except that there is no arc to an effect condition formula node.
  - effects with identical condition (including groups of unconditional effects) share the same effect node.
  - the effect node for unconditional effects is denoted by \( o^+_{\chi'} \).

Operator subgraph for \( o_1 = \langle b \lor (c \land d), b \land ((a \land b) \supset e) \rangle \) for layer \( i = 1 \).

\[ \begin{align*}
    a^1 & \quad n_{a}^1 \\
    b^2 & \quad n_{b}^1 \\
    c^3 & \quad n_{c}^1 \\
    d^2 & \quad n_{d}^1 \\
    e^1 & \quad n_{e}^1 
\end{align*} \]
Relaxed planning graph: goal subgraphs

Goal subgraph for $\gamma = e \land (g \land h)$ and depth $k = 2$:

Connection to forward sets and plan steps

Theorem (relaxed planning graph truth values)
Let $\Pi^+ = \langle A, I, O^+, \gamma \rangle$ be a relaxed planning task. Then the truth values of the nodes of its depth-$k$ relaxed planning graph $\text{RPG}_k(\Pi^+)$ relate to the forward sets and forward plan steps of $\Pi^+$ as follows:

- **Proposition nodes:**
  For all $a \in A$ and $i \in \{0, \ldots, k\}$, $\text{val}(a^i) = 1$ iff $a \in S_F^i$.

- **(Unconditional) effect nodes:**
  For all $o \in O^+$ and $i \in \{1, \ldots, k\}$, $\text{val}(o^i) = 1$ iff $o \in \omega_F^i$.

- **Goal nodes:**
  $\text{val}(n^i_{\gamma}) = 1$ iff the parallel forward distance of $\Pi^+$ is at most $k$.

(We omit the straight-forward proof.)
Parallel plans
Relaxed planning graphs
Introduction
Construction
Truth values
Relaxation
heuristics
Summary

Remark: Relaxed planning graphs have historically been defined for STRIPS tasks only. In this case, we can simplify:
- Only one effect node per operator: STRIPS does not have conditional effects.
- Because each operator has only one effect node, effect nodes are called operator nodes in relaxed planning graphs for STRIPS.
- No goal nodes: The test whether all goals are reached is done by the algorithm that evaluates the AND/OR dag.
- No formula nodes: Operator nodes are directly connected to their preconditions.

⇒ Relaxed planning graphs for STRIPS are layered digraphs and only have proposition and operator nodes.

November 22nd, 2013 B. Nebel, R. Mattmüller – AI Planning 30 / 62

3 Relaxation heuristics

- Generic template for relaxation heuristics
- The max heuristic \( h_{\text{max}} \)
- The additive heuristic \( h_{\text{add}} \)
- The set-additive heuristic \( h_{\text{sa}} \)
- Incremental computation
- The FF heuristic \( h_{\text{FF}} \)
- Comparison & relaxation heuristics in practice

November 22nd, 2013 B. Nebel, R. Mattmüller – AI Planning 32 / 62

Computing parallel forward distances from RPGs

So far, relaxed planning graphs offer us a way to compute parallel forward distances:

Parallel forward distances from relaxed planning graphs

```python
def parallel-forward-distance(\( \Pi^+ \)):  
    Let \( A \) be the set of state variables of \( \Pi^+ \)
    for \( k \in \{0,1,2,\ldots\} \):
        \( \text{rpg} := \text{RPG}_k(\Pi^+) \)
        Evaluate truth values for \( \text{rpg} \)
        if goal node of \( \text{rpg} \) has value 1:
            return \( k \)
        else if \( k = |A| \):
            return \( \infty \)
```

November 22nd, 2013 B. Nebel, R. Mattmüller – AI Planning 33 / 62

Remarks on the algorithm

- The relaxed planning graph for depth \( k \geq 1 \) can be built incrementally from the one for depth \( k - 1 \):
  - Add new layer \( k \).
  - Move goal subgraph from layer \( k - 1 \) to layer \( k \).
- Similarly, all truth values up to layer \( k - 1 \) can be reused.
- Thus, overall computation with maximal depth \( m \) requires time \( O(\|\text{RPG}_m(\Pi^+)\|) = O((m+1) \cdot \|\Pi^+\|) \).
- This is not a very efficient way of computing parallel forward distances (and wouldn’t be used in practice).
- However, it allows computing additional information for the relaxed planning graph nodes along the way, which can be used for heuristic estimates.

November 22nd, 2013 B. Nebel, R. Mattmüller – AI Planning 34 / 62
Generic relaxed planning graph heuristics

Computing heuristics from relaxed planning graphs

```python
def generic-rpg-heuristic((A, I, O, γ), s):
    Π⁺ := (A, s, O⁺, γ)
    for k ∈ {0, 1, 2, ...}:
        rpg := RPG_k(Π⁺)
        Evaluate truth values for rpg.
        if goal node of rpg has value 1:
            Annotate true nodes of rpg.
        if termination criterion is true:
            return heuristic value from annotations
        else if k = |A|:
            return ∞
```

⇝ generic template for heuristic functions
⇝ to get concrete heuristic: fill in highlighted parts

Concrete examples for the generic heuristic

Many planning heuristics fit the generic template:
- additive heuristic $h_{add}$ (Bonet, Loerincs & Geffner, 1997)
- max heuristic $h_{max}$ (Bonet & Geffner, 1999)
- FF heuristic $h_{FF}$ (Hoffmann & Nebel, 2001)
- cost-sharing heuristic $h_{cs}$ (Mirkis & Domshlak, 2007)
- not covered in this course
- set-additive heuristic $h_{sa}$ (Keyder & Geffner, 2008)

Remarks:
- For all these heuristics, equivalent definitions that don’t refer to relaxed planning graphs are possible.
- Historically, such equivalent definitions have mostly been used for $h_{max}$, $h_{add}$ and $h_{sa}$.
- For those heuristics, the most efficient implementations do not use relaxed planning graphs explicitly.

Forward cost heuristics

The simplest relaxed planning graph heuristics are forward cost heuristics.
- Examples: $h_{max}$, $h_{add}$
- Here, node annotations are cost values (natural numbers).
- The cost of a node estimates how expensive (in terms of required operators) it is to make this node true.

Forward cost heuristics: fitting the template

Computing annotations:
- Propagate cost values bottom-up using a combination rules for OR nodes and for AND nodes.
- At effect nodes, add 1 after applying combination rule.
Termination criterion:
- stability: terminate if cost for proposition node $a^k$ equals cost for $a^{k-1}$ for all true propositions $a$ in layer $k$
Heuristic value:
- The heuristic value is the cost of the goal node.
- Different forward cost heuristics only differ in their choice of combination rules.
The max heuristic $h_{\text{max}}$ (again)

Forward cost heuristics: max heuristic $h_{\text{max}}$

Combination rule for AND nodes:

\[
\text{cost}(u) = \max \{\text{cost}(v_1), \ldots, \text{cost}(v_k)\}
\]
(with $\max(\emptyset) := 0$)

Combination rule for OR nodes:

\[
\text{cost}(u) = \min \{\text{cost}(v_1), \ldots, \text{cost}(v_k)\}
\]

In both cases, $\{v_1, \ldots, v_k\}$ is the set of true successors of $u$.

Intuition:

- **AND rule**: If we have to achieve several conditions, estimate this by the most expensive cost.
- **OR rule**: If we have a choice how to achieve a condition, pick the cheapest possibility.

Remarks on $h_{\text{max}}$

- The definition of $h_{\text{max}}$ as a forward cost heuristic is equivalent to our earlier definition in this chapter.
- Unlike the earlier definition, it generalizes to an extension where every operator has an associated non-negative cost (rather than all operators having cost 1).
- In the case without costs (and only then), it is easy to prove that the goal node has the same cost in all graphs $\text{RPG}_k(\Pi^+)$ where it is true. (Namely, the cost is equal to the lowest value of $k$ for which the goal node is true.)
- We can thus terminate the computation as soon as the goal becomes true, without waiting for stability.
- The same is not true for other forward-propagating heuristics ($h_{\text{add}}, h_{\text{CS}}, h_{\text{sa}}$).

The additive heuristic

Forward cost heuristics: additive heuristic $h_{\text{add}}$

Combination rule for AND nodes:

\[
\text{cost}(u) = \text{cost}(v_1) + \ldots + \text{cost}(v_k)
\]
(with $\sum(\emptyset) := 0$)

Combination rule for OR nodes:

\[
\text{cost}(u) = \min \{\text{cost}(v_1), \ldots, \text{cost}(v_k)\}
\]

In both cases, $\{v_1, \ldots, v_k\}$ is the set of true successors of $u$.

Intuition:

- **AND rule**: If we have to achieve several conditions, estimate this by the cost of achieving each in isolation.
- **OR rule**: If we have a choice how to achieve a condition, pick the cheapest possibility.
Running example: $h_{\text{add}}$

Remarks on $h_{\text{add}}$

- It is important to test for stability in computing $h_{\text{add}}$! (The reason for this is that, unlike $h_{\text{max}}$, cost values of true propositions can decrease from layer to layer.)
- Stability is achieved after layer $|A|$ in the worst case.
- $h_{\text{add}}$ is safe and goal-aware.
- Unlike $h_{\text{max}}$, $h_{\text{add}}$ is a very informative heuristic in many planning domains.
- The price for this is that it is not admissible (and hence also not consistent), so not suitable for optimal planning.
- In fact, it almost always overestimates the $h^+$ value because it does not take positive interactions into account.

The set-additive heuristic

- We now discuss a refinement of the additive heuristic called the set-additive heuristic $h_{\text{sa}}$.
- The set-additive heuristic addresses the problem that $h_{\text{add}}$ does not take positive interactions into account.
- Like $h_{\text{max}}$ and $h_{\text{add}}$, $h_{\text{sa}}$ is calculated through forward propagation of node annotations.
- However, the node annotations are not cost values, but sets of operators (kind of).
- The idea is that by taking set unions instead of adding costs, operators needed only once are counted only once.

Disclaimer: There are some quite subtle differences between the $h_{\text{sa}}$ heuristic as we describe it here and the “real” heuristic of Keyder & Geffner. We do not want to discuss this in detail, but please note that such differences exist.

Operators needed several times

- The original $h_{\text{sa}}$ heuristic as described in the literature is defined for STRIPS tasks and propagates sets of operators.
- This is fine because in relaxed STRIPS tasks, each operator need only be applied once.
- The same is not true in general: in our running example, operator $o_1$ must be applied twice in the relaxed plan.
- In general, it only makes sense to apply an operator again in a relaxed planning task if a previously unsatisfied effect condition has been made true.
- For this reason, we keep track of operator/effect condition pairs rather than just plain operators.
Set-additive heuristic: fitting the template

The set-additive heuristic $h_{sa}$

Computing annotations:
- Annotations are sets of operator/effect condition pairs, computed bottom-up.
  - Combination rule for AND nodes:
    \[ ann(u) = ann(v_1) \cup \cdots \cup ann(v_k) \] (with $\emptyset \cup \emptyset = \emptyset$)
  - Combination rule for OR nodes:
    \[ ann(u) = ann(v) \] for some $v$ minimizing $|ann(v)|$
  - In case of several minimizers, use any tie-breaking rule.

In both cases, $\{v_1, \ldots, v_k\}$ is the set of true successors of $u$. At effect nodes, add the corresponding operator/effect condition pair to the set after applying combination rule.

Running example: $h_{sa}$

Remarks on $h_{sa}$

- The same remarks for stability as for $h_{add}$ apply.
- Like $h_{add}$, $h_{sa}$ is safe and goal-aware, but neither admissible nor consistent.
- $h_{sa}$ is generally better informed than $h_{add}$, but significantly more expensive to compute.
- The $h_{sa}$ value depends on the tie-breaking rule used, so $h_{sa}$ is not well-defined without specifying the tie-breaking rule.
- The operators contained in the goal node annotation, suitably ordered, define a relaxed plan for the task.
  - Operators mentioned several times in the annotation must be added as many times in the relaxed plan.
One nice property of forward-propagating heuristics is that they allow *incremental computation:*

- when evaluating several states in sequence which only differ in a few state variables, can
- start computation from previous results and
- keep track only of what needs to be recomputed
- typical use case: depth-first style searches (e.g., IDA*)
- rarely exploited in practice

**Incremental computation example:**

<table>
<thead>
<tr>
<th>Change value of e to 1.</th>
<th>Recompute outdated values.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result for {a → 1, b → 0, c → 1, d → 1, e → 0, f → 0, g → 0, h → 0}</td>
<td>&quot;Incremental computation example: h\text{add}&quot;</td>
</tr>
</tbody>
</table>

---

*November 22nd, 2013 B. Nebel, R. Mattmüller – AI Planning*
Incremental computation example: $h_{\text{add}}$

Recompute outdated values.

November 22nd, 2013  B. Nebel, R. Mattmüller – AI Planning  51 / 62
Heuristic estimate $h_{\text{FF}}$

- $h_{\text{sa}}$ is more expensive to compute than the other forward propagating heuristics because we must propagate sets.
- It is possible to get the same advantage over $h_{\text{add}}$ combined with efficient propagation.
- Key idea of $h_{\text{FF}}$: perform a backward propagation that selects a sufficient subset of nodes to make the goal true (called a solution graph in AND/OR dag literature).
- The resulting heuristic is almost as informative as $h_{\text{sa}}$, yet computable as quickly as $h_{\text{add}}$.

Note: Our presentation inverts the historical order. The set-additive heuristic was defined after the FF heuristic (sacrificing speed for even higher informativeness).

FF heuristic: fitting the template

The FF heuristic $h_{\text{FF}}$

Computing annotations:
- Annotations are Boolean values, computed top-down.
- A node is marked when its annotation is set to 1 and unmarked if it is set to 0. Initially, the goal node is marked, and all other nodes are unmarked.
- We say that a true AND node is justified if all its true successors are marked, and that a true OR node is justified if at least one of its true successors is marked.

...
Comparison of relaxation heuristics

**Theorem (relationship between relaxation heuristics)**

Let \( s \) be a state of planning task \( \langle A, I, O, \gamma \rangle \). Then:

- \( h_{\text{max}}(s) \leq h^+(s) \leq h^*(s) \)
- \( h_{\text{max}}(s) \leq h^+(s) \leq h_{\text{sa}}(s) \leq h_{\text{add}}(s) \)
- \( h_{\text{max}}(s) \leq h^+(s) \leq h_{\text{FF}}(s) \leq h_{\text{add}}(s) \)
- \( h^*, h_{\text{FF}} \) and \( h_{\text{sa}} \) are pairwise incomparable
- \( h^* \) and \( h_{\text{add}} \) are incomparable

Moreover, \( h^+, h_{\text{max}}, h_{\text{add}}, h_{\text{sa}} \) and \( h_{\text{FF}} \) assign \( \infty \) to the same set of states.

**Note:** For inadmissible heuristics, dominance is in general neither desirable nor undesirable. For relaxation heuristics, the objective is usually to get as close to \( h^+ \) as possible.

Remarks on \( h_{\text{FF}} \)

- Like \( h_{\text{add}} \) and \( h_{\text{sa}} \), \( h_{\text{FF}} \) is safe and goal-aware, but neither admissible nor consistent.
- Its informativeness can be expected to be slightly worse than for \( h_{\text{sa}} \), but is usually not far off.
- Unlike \( h_{\text{sa}} \), \( h_{\text{FF}} \) can be computed in linear time.
- Similar to \( h_{\text{sa}} \), the operators corresponding to the marked operator/effect condition pairs define a relaxed plan.
- Similar to \( h_{\text{sa}} \), the \( h_{\text{FF}} \) value depends on tie-breaking when the marking rules allow several possible choices, so \( h_{\text{FF}} \) is not well-defined without specifying the tie-breaking rule.
- The implementation in FF uses additional rules of thumb to try to reduce the size of the generated relaxed plan.

Relaxation heuristics in practice: HSP

**Example (HSP)**

HSP (Bonet & Geffner) was one of the four top performers at the 1st International Planning Competition (IPC-1998).

**Key ideas:**

- hill climbing search using \( h_{\text{add}} \)
- on plateaus, keep going for a number of iterations, then restart
- use a closed list during exploration of plateaus

**Literature:** Bonet, Loerincs & Geffner (1997), Bonet & Geffner (2001)
Relaxation heuristics in practice: FF

Example (FF)

FF (Hoffmann & Nebel) won the 2nd International Planning Competition (IPC-2000).

Key ideas:

- **enforced hill-climbing** search using $h_{FF}$
- **helpful action pruning**: in each search node, only consider successors from operators that add one of the atoms marked in proposition layer 1
- **goal ordering**: in certain cases, FF recognizes and exploits that certain subgoals should be solved one after the other

If main search fails, FF performs greedy best-first search using $h_{FF}$ without helpful action pruning or goal ordering.


Relaxation heuristics in practice: Fast Downward

Example (Fast Downward)

Fast Downward (Helmert & Richter) won the satisficing track of the 4th International Planning Competition (IPC-2004).

Key ideas:

- **greedy best-first search** using $h_{FF}$ and causal graph heuristic (not relaxation-based)
- **search enhancements:**
  - multi-heuristic best-first search
  - deferred evaluation of heuristic estimates
  - preferred operators (similar to FF’s helpful actions)


Relaxation heuristics in practice: SGPlan

Example (SGPlan)

SGPlan (Wah, Hsu, Chen & Huang) won the satisficing track of the 5th International Planning Competition (IPC-2006).

Key ideas:

- FF
- problem decomposition techniques
- domain-specific techniques

Literature: Chen, Wah & Hsu (2006)

Relaxation heuristics in practice: LAMA

Example (LAMA)

LAMA (Richter & Westphal) won the satisficing track of the 6th International Planning Competition (IPC-2008).

Key ideas:

- **Fast Downward**
- **landmark pseudo-heuristic** instead of causal graph heuristic (“somewhat” relaxation-based)
- anytime variant of Weighted A* instead of greedy best-first search

Parallel plans

Relaxed planning graphs are AND/OR dags. They encode which propositions can be made true in $\Pi^+$ and how.
- Closely related to forward sets and forward plan steps, based on the notion of parallel relaxed plans.
- They can be constructed and evaluated efficiently, in time $O((m+1)||\Pi^+||)$ for planning task $\Pi$ and depth $m$.

By annotating RPG nodes with appropriate information, we can compute many useful heuristics.
- Examples: max heuristic $h_{\text{max}}$, additive heuristic $h_{\text{add}}$, set-additive heuristic $h_{\text{sa}}$ and FF heuristic $h_{\text{FF}}$
  - Of these, only $h_{\text{max}}$ admissible (but not very accurate).
  - The others are much more informative. The set-additive heuristic is the most sophisticated one.
  - The FF heuristic is often similarly informative. It offers a good trade-off between accuracy and computation time.