

# Automated (AI) Planning

## Relaxation and Domain-Independent Heuristics

Carmel Domshlak

# Where heuristics come from?

## General idea

(Admissible) heuristic functions obtained as  
(optimal) cost functions of relaxed problems

## Examples

- Euclidian distance in Path Finding
- Manhattan distance in N-puzzle
- Spanning Tree in Traveling Salesman Problem
- Shortest Path in Job Shop Scheduling

Automated  
(AI) Planning

Introduction

Obtaining  
heuristics

Relaxation  
heuristics

Relaxation  
Heuristics

# Example

## 8-Puzzle

7	2	4
5		6
8	3	1

Start State

1	2	3
4	5	6
7	8	

Goal State

- A tile can move from square A to square B if A is adjacent to B and B is blank  $\leadsto$  solution distance  $h^*$
- A tile can move from square A to square B if A is adjacent to B  $\leadsto$  manhattan distance heuristic  $h^{MD}$
- A tile can move from square A to square B  $\leadsto$  misplaced tiles heuristic  $h^{MT}$

Here:  $h^*(s_0) = ?$ ,  $h^{MD}(s_0) = 14$ ,  $h^{MT}(s_0) = 6$

In general,  $h^* \geq h^{MD} \geq h^{MT}$ . (Why?)

# Example

## 8-Puzzle

7	2	4
5		6
8	3	1

Start State

1	2	3
4	5	6
7	8	

Goal State

- A tile can move from square A to square B if A is adjacent to B and B is blank  $\rightsquigarrow$  solution distance  $h^*$
- A tile can move from square A to square B if A is adjacent to B  $\rightsquigarrow$  manhattan distance heuristic  $h^{MD}$
- A tile can move from square A to square B  $\rightsquigarrow$  misplaced tiles heuristic  $h^{MT}$

Here:  $h^*(s_0) = ?$ ,  $h^{MD}(s_0) = 14$ ,  $h^{MT}(s_0) = 6$

In general,  $h^* \geq h^{MD} \geq h^{MT}$ . (Why?)

# Dominance relation between admissible heuristics

## Precision matters

Given two admissible heuristics  $h_1, h_2$ , if  $h_2(\sigma) \geq h_1(\sigma)$  for all search nodes  $\sigma$ , then  $h_2$  **dominates**  $h_1$  and is better for optimizing search

## Typical search costs (unit-cost action)

$h^*(I) = 14$  BFS  $\approx 1,700,000$  nodes

$A^*(h^{MT}) \approx 560$  nodes

$A^*(h^{MD}) \approx 115$  nodes

$h^*(I) = 24$  BFS  $\approx 27,000,000,000$  nodes

$A^*(h^{MT}) \approx 40,000$  nodes

$A^*(h^{MD}) \approx 1,650$  nodes

# Dominance relation between admissible heuristics

## Precision matters

Given two admissible heuristics  $h_1, h_2$ , if  $h_2(\sigma) \geq h_1(\sigma)$  for all search nodes  $\sigma$ , then  $h_2$  **dominates**  $h_1$  and is better for optimizing search

## Combining admissible heuristics

For any admissible heuristics  $h_1, \dots, h_k$ ,

$$h(\sigma) = \max_{i=1}^k \{h_i(\sigma)\}$$

is also admissible and dominates all individual  $h_i$

Later we'll see that **max** is just a special case of something more general.

# Are we solver?

## General idea

(Admissible) heuristic functions obtained as (optimal) cost functions of relaxed problems

- OK, but heuristic is **yet another input** to our agent!
- Satisfactory for general solvers?
- Satisfactory in special purpose solvers?

## Towards domain-independent agents

- How to get heuristics **automatically**?
- Can such automatically derived heuristics **dominate** the domain-specific heuristics crafted by hand?

# Are we solver?

## General idea

(Admissible) heuristic functions obtained as (optimal) cost functions of relaxed problems

- OK, but heuristic is **yet another input** to our agent!
- Satisfactory for general solvers?
- Satisfactory in special purpose solvers?

## Towards domain-independent agents

- How to get heuristics **automatically**?
- Can such automatically derived heuristics **dominate** the domain-specific heuristics crafted by hand?



# A simple heuristic for deterministic planning

STRIPS (Fikes & Nilsson, 1971) used the number of state variables that differ in current state  $s$  and a STRIPS goal  $G = \{g_1, \dots, g_k\}$ :

$$h(s) := |G \setminus s|.$$

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

$\rightsquigarrow$  **STRIPS heuristic** (properties?)

# Criticism of the STRIPS heuristic

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 actions!

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

# Coming up with heuristics in a principled way

## 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 (separately and *together*).

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

# Relaxations for planning

- Relaxation is a general technique for heuristic design:
  - **Straight-line heuristic** (route planning): Ignore the fact that one must stay on roads.
  - **Manhattan heuristic** (15-puzzle): Ignore the fact that one cannot move through occupied tiles.
- We want to apply the idea of relaxations to planning.
- Informally, we want to ignore **bad side effects** of applying actions.

## Example (8-puzzle)

If we move a tile from  $x$  to  $y$ , then the **good effect** is (in particular) that  $x$  is now free.

The **bad effect** is that  $y$  is not free anymore, preventing us for moving tiles through it.

# Relaxed planning tasks: idea

In STRIPS, good and bad effects are easy to distinguish:

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

Idea for the heuristic: **Ignore all delete effects.**

# Relaxed planning tasks: idea

In STRIPS, good and bad effects are easy to distinguish:

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

Idea for the heuristic: **Ignore all delete effects.**

# Relaxed planning tasks

## Definition (relaxation of actions)

The **relaxation**  $a^+$  of a STRIPS action  $a = \langle \text{pre}(a), \text{add}(a), \text{del}(a) \rangle$  is the action  $a^+ = \langle \text{pre}(a), \text{add}(a), \emptyset \rangle$ .

## Definition (relaxation of planning tasks)

The **relaxation**  $\Pi^+$  of a STRIPS planning task  $\Pi = \langle P, A, I, G \rangle$  is the planning task  $\Pi^+ := \langle P, \{a^+ \mid a \in A\}, I, G \rangle$ .

## Definition (relaxation of action sequences)

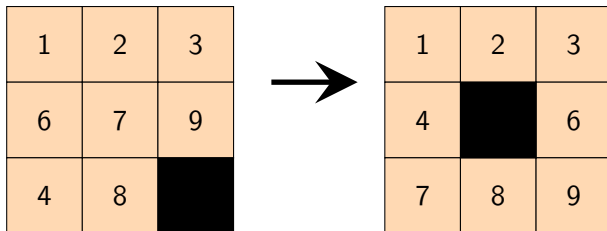
The **relaxation** of an action sequence  $\pi = a_1 \dots a_n$  is the action sequence  $\pi^+ := a_1^+ \dots a_n^+$ .

# Relaxed planning tasks: terminology

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

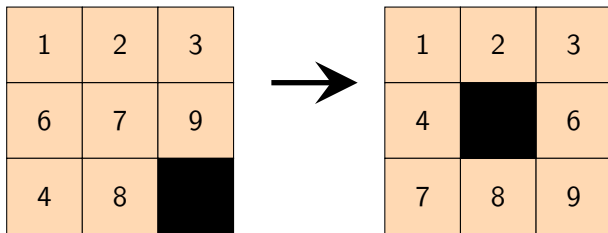


# Example: 8-Puzzle



- Real problem:
  - A tile can move from square A to square B if A is adjacent to B and B is blank
- Monotonically relaxed problem:
  - A tile can move from square A to square B if A is adjacent to B and B is blank (!!!)
  - In effect ...

# Example: 8-Puzzle



- A tile can move from square A to square B if A is adjacent to B and B is blank - solution distance  $h^*$
- A tile can move from square A to square B if A is adjacent to B - manhattan distance heuristic  $h^{MD}$
- A tile can move from square A to square B if A is adjacent to B and B is blank; in effect, the tile is at both A and B, and both A and B are blank -  $h^+$

Here:  $h^*(s_0) = 8$ ,  $h^{MD}(s_0) = 6$ ,  $h^+(s_0) = ???$

# Example: 8-Puzzle

1	2	3
6	7	9
4	8	



1	2	3
4		6
7	8	9

Optimal MD plan:

- 1  $move(t_9, p_6, p_9)$
- 2  $move(t_7, p_5, p_8)$
- 3  $move(t_6, p_4, p_5)$
- 4  $move(t_6, p_5, p_6)$
- 5  $move(t_4, p_7, p_4)$
- 6  $move(t_7, p_8, p_7)$

Optimal relaxed plan:

- 1  $move(t_9, p_6, p_9)$
- 2  $move(t_8, p_8, p_9)$
- 3  $move(t_7, p_5, p_8)$
- 4  $move(t_6, p_4, p_5)$
- 5  $move(t_6, p_5, p_6)$
- 6  $move(t_4, p_7, p_4)$
- 7  $move(t_7, p_8, p_7)$

Automated  
(AI) Planning

Introduction

Obtaining  
heuristics

Relaxation  
heuristics

The relaxation  
lemma

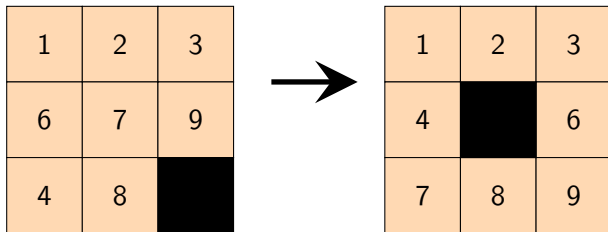
Greedy algorithm

Optimality

Discussion

Relaxation  
Heuristics

# Example: 8-Puzzle



Optimal MD plan:

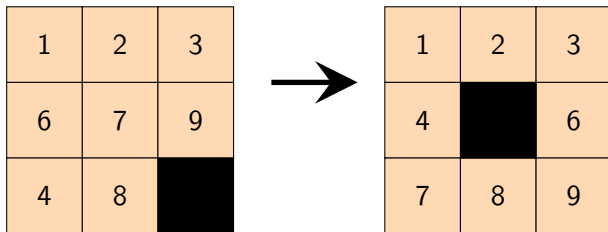
- 1  $move(t_9, p_6, p_9)$
- 2  $move(t_7, p_5, p_8)$
- 3  $move(t_6, p_4, p_5)$
- 4  $move(t_6, p_5, p_6)$
- 5  $move(t_4, p_7, p_4)$
- 6  $move(t_7, p_8, p_7)$

Optimal relaxed plan:

- 1  $move(t_9, p_6, p_9)$
- 2  $move(t_8, p_8, p_9)$
- 3  $move(t_7, p_5, p_8)$
- 4  $move(t_6, p_4, p_5)$
- 5  $move(t_6, p_5, p_6)$
- 6  $move(t_4, p_7, p_4)$
- 7  $move(t_7, p_8, p_7)$

So  $h^*(s_0) = 8$ ,  $h^{MD}(s_0) = 6$ ,  $h^+(s_0) = 7 (> h^{MD}!)$

## 8-Puzzle: $h^+$ vs. $h^{MD}$



$h^+$  dominates  $h^{MD}$

- The goal is given as a conjunction of  $at(t_i, p_j)$  atoms
- Achieving each single one of them takes at least as many steps as the respective tile's Manhattan distance
- Each action moves a single tile only

And we have just seen that  $h^+$  **strictly dominates**  $h^{MD}$

Automated  
(AI) Planning

Introduction

Obtaining  
heuristics

Relaxation  
heuristics

The relaxation  
lemma  
Greedy algorithm  
Optimality  
Discussion

Relaxation  
Heuristics

# Dominating states

The **on-set**  $on(s)$  of a state  $s$  is the set of atoms that are true in  $s$ . A state  $s'$  **dominates** another state  $s$  iff  $on(s) \subseteq on(s')$ .

## Lemma (relaxation)

*Let  $s$  be a state, let  $s'$  be a state that dominates  $s$ , and let  $\pi$  be an action 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 is immediate from  $app_{\pi^+}(s')$  dominating  $app_{\pi}(s)$ . Prove the rest by induction over the length of  $\pi$ .

# Consequences of the relaxation lemma

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

*Let  $\pi$  be an action sequence which 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 the original task.
- ~> Optimal relaxed plans are never longer than optimal plans for original tasks.

# Consequences of the relaxation lemma (ctd.)

## Corollary (relaxation preserves dominance)

*Let  $s$  be a state, let  $s'$  be a state that dominates  $s$ , and let  $\pi^+$  be a relaxed action 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.



# Monotonicity of relaxed planning tasks

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

## Lemma (monotonicity)

*Let  $a^+ = \langle \text{pre}(a), \text{add}(a), \emptyset \rangle$  be a relaxed action and let  $s$  be a state in which  $a^+$  is applicable.*

*Then  $\text{app}_{a^+}(s)$  dominates  $s$ .*

## Proof.

Since relaxed actions only have positive effects, we have  $\text{on}(s) \subseteq \text{on}(s) \cup \text{add}(a) = \text{on}(\text{app}_{a^+}(s))$ . □

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

Automated  
(AI) Planning

Introduction

Obtaining  
heuristics

Relaxation  
heuristics

The relaxation  
lemma

Greedy algorithm

Optimality

Discussion

Relaxation  
Heuristics

# Greedy algorithm for relaxed planning tasks

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

Greedy planning algorithm for  $\langle P, A^+, I, G \rangle$

$s := I$

$\pi^+ := \epsilon$

**forever:**

**if**  $G \subseteq s$ :

**return**  $\pi^+$

**else if** there is an action  $a^+ \in A^+$  applicable in  $s$

    with  $app_{a^+}(s) \neq s$ :

    Append such an action  $a^+$  to  $\pi^+$ .

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

**else:**

**return** unsolvable

Automated  
(AI) Planning

Introduction

Obtaining  
heuristics

Relaxation  
heuristics

The relaxation  
lemma

**Greedy algorithm**

Optimality

Discussion

Relaxation  
Heuristics

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

# Using the greedy algorithm as a heuristic

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?)

# 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 lengths as an estimate, even though it is NP-hard.  
 $\leadsto$   **$h^+$  heuristic** (*not that realistic. why?*)
- Do not actually solve the relaxed planning task, but compute an estimate of its difficulty in a different way.  
 $\leadsto$   **$h_{\max}$  heuristic,  $h_{\text{add}}$  heuristic**
- Compute a solution for relaxed planning tasks which is not necessarily optimal, but “reasonable”.  
 $\leadsto$   **$h_{\text{FF}}$  heuristic**

# 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 lengths as an estimate, even though it is NP-hard.  
 $\leadsto$   **$h^+$  heuristic** (*not that realistic. why?*)
- Do not actually solve the relaxed planning task, but compute an estimate of its difficulty in a different way.  
 $\leadsto$   **$h_{\max}$  heuristic,  $h_{\text{add}}$  heuristic**
- Compute a solution for relaxed planning tasks which is not necessarily optimal, but “reasonable”.  
 $\leadsto$   **$h_{\text{FF}}$  heuristic**

# 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 lengths as an estimate, even though it is NP-hard.  
 $\leadsto$   **$h^+$  heuristic** (*not that realistic. why?*)
- Do not actually solve the relaxed planning task, but compute an estimate of its difficulty in a different way.  
 $\leadsto$   **$h_{\max}$  heuristic,  $h_{\text{add}}$  heuristic**
- Compute a solution for relaxed planning tasks which is not necessarily optimal, but “reasonable”.  
 $\leadsto$   **$h_{\text{FF}}$  heuristic**

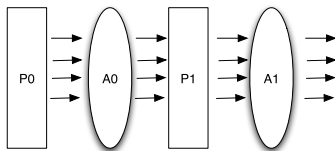
# Graphical “interpretation”: Relaxed planning graphs

- Build a layered **reachability graph**  $P_0, A_0, P_1, A_1, \dots$

$$P_0 = \{p \in I\}$$

$$A_i = \{a \in A \mid \text{pre}(a) \subseteq P_i\}$$

$$P_{i+1} = P_i \cup \{p \in \text{add}(a) \mid a \in A_i\}$$



- Terminate when  $G \subseteq P_i$



# Running example

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

$$a_1 = \langle \{a\}, \{b, c\}, \emptyset \rangle$$

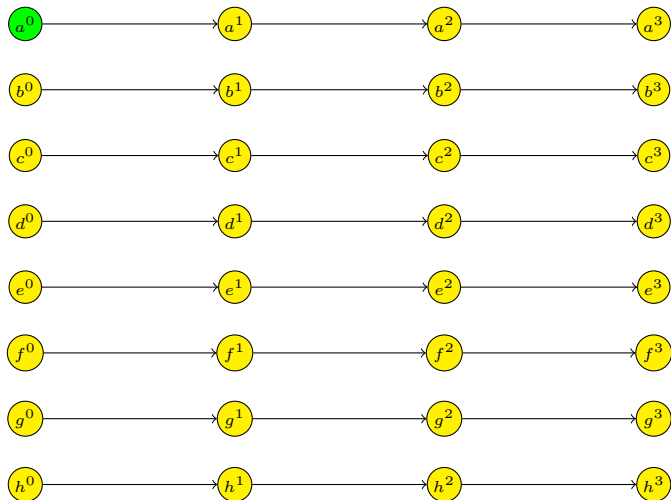
$$a_2 = \langle \{a, c\}, \{d\}, \emptyset \rangle$$

$$a_3 = \langle \{b, c\}, \{e\}, \emptyset \rangle$$

$$a_4 = \langle \{b\}, \{f\}, \emptyset \rangle$$

$$a_5 = \langle \{d\}, \{g\}, \emptyset \rangle$$

# Running example: Relaxed planning graph



Automated  
(AI) Planning

Introduction

Obtaining  
heuristics

Relaxation  
heuristics

Relaxation  
Heuristics

Template

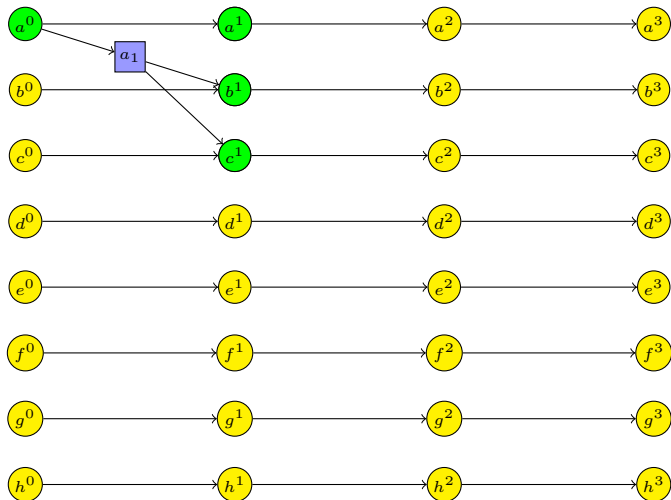
$h_{\max}$

$h_{\text{add}}$

$h_{\text{FF}}$

Comparison &  
practice

# Running example: Relaxed planning graph



Automated  
(AI) Planning

Introduction

Obtaining  
heuristics

Relaxation  
heuristics

Relaxation  
Heuristics

Template

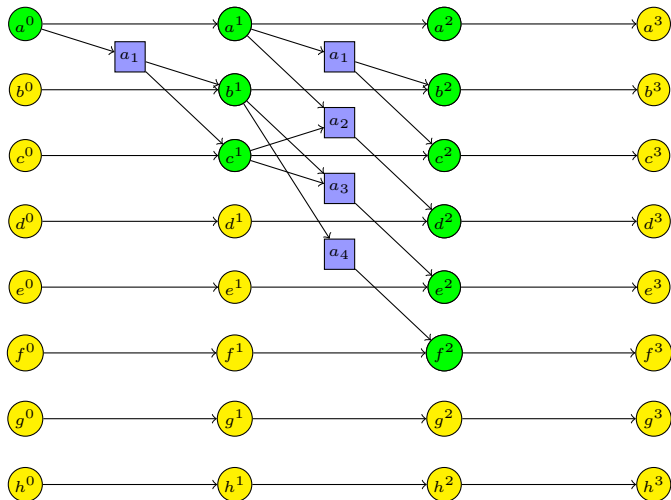
$h_{\max}$

$h_{\text{add}}$

$h_{\text{FF}}$

Comparison &  
practice

# Running example: Relaxed planning graph



Automated  
(AI) Planning

Introduction

Obtaining  
heuristics

Relaxation  
heuristics

Relaxation  
Heuristics

Template

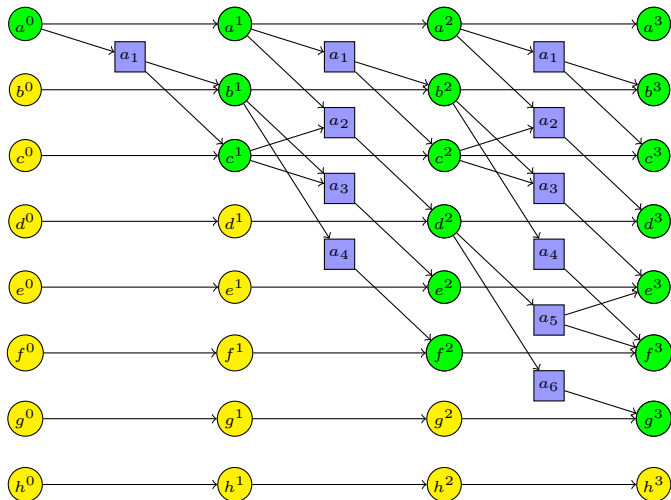
$h_{\max}$

$h_{\text{add}}$

$h_{\text{FF}}$

Comparison &  
practice

# Running example: Relaxed planning graph



Automated  
(AI) Planning

Introduction

Obtaining  
heuristics

Relaxation  
heuristics

Relaxation  
Heuristics

Template

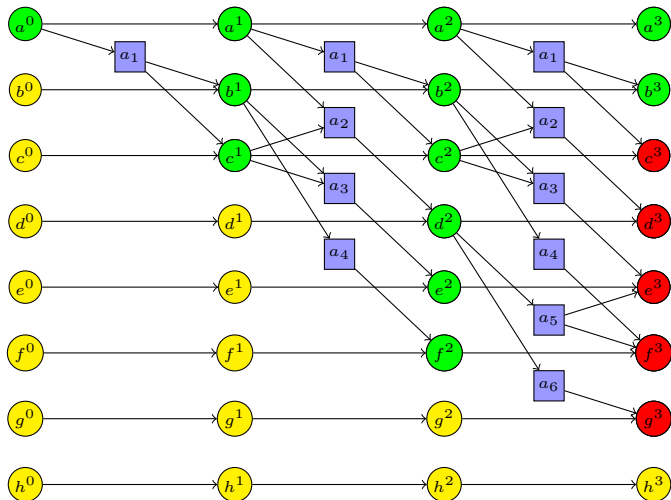
$h_{\max}$

$h_{\text{add}}$

$h_{\text{FF}}$

Comparison &  
practice

# Running example: Relaxed planning graph



Automated  
(AI) Planning

Introduction

Obtaining  
heuristics

Relaxation  
heuristics

Relaxation  
Heuristics

Template

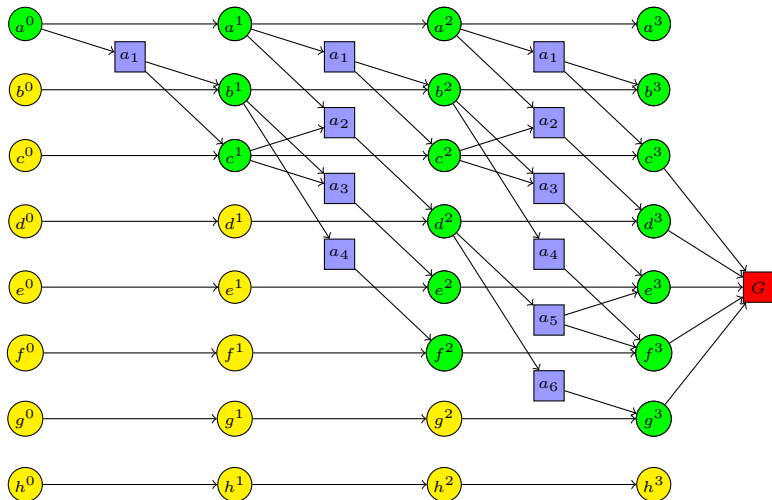
$h_{\max}$

$h_{\text{add}}$

$h_{\text{FF}}$

Comparison &  
practice

# Running example: Relaxed planning graph



Automated  
(AI) Planning

Introduction

Obtaining  
heuristics

Relaxation  
heuristics

Relaxation  
Heuristics

Template

$h_{\max}$

$h_{\text{add}}$

$h_{\text{FF}}$

Comparison &  
practice

# Generic relaxed planning graph heuristics

## Computing heuristics from relaxed planning graphs

```
def generic-rpg-heuristic( $\langle P, I, O, G \rangle, s$ ):  
     $\Pi^+ := \langle P, s, O^+, G \rangle$   
    for  $k \in \{0, 1, 2, \dots\}$ :  
         $rpg := RPG_k(\Pi^+)$   
        if  $G \subseteq P_k$ :  
            Annotate nodes of  $rpg$ .  
            if termination criterion is true:  
                return heuristic value from annotations  
        else if  $k = |P|$ :  
            return  $\infty$ 
```

↪ generic template for heuristic functions

↪ to get concrete heuristic: fill in highlighted parts

Automated  
(AI) Planning

Introduction

Obtaining  
heuristics

Relaxation  
heuristics

Relaxation  
Heuristics

Template

$h_{\max}$

$h_{\text{add}}$

$h_{\text{FF}}$

Comparison &  
practice



# Concrete examples for the generic heuristic

Many planning heuristics fit the generic template:

- max heuristic  $h_{\max}$
- additive heuristic  $h_{\text{add}}$
- FF heuristic  $h_{\text{FF}}$
- ...

Remarks:

- For all these heuristics, equivalent definitions that don't refer to relaxed planning graphs are possible.
- For some of these 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:  $h_{\max}$ ,  $h_{\text{add}}$
- 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

## Forward cost heuristics

### Computing annotations:

- Propagate cost values bottom-up using a combination rule for action nodes and a combination rule for proposition nodes.
- At **action nodes**, **add 1** after applying combination rule.

### Termination criterion:

- **stability**: terminate if  $P_k = P_{k-1}$  and cost for each proposition node  $p^k \in P_k$  equals cost for  $p^{k-1} \in P_{k-1}$

### Heuristic value:

- The heuristic value is the cost of the auxiliary goal node.
- Different forward cost heuristics only differ in their choice of combination rules.

# The max heuristic $h_{\max}$ (again)

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

Combination rule for action nodes:

- $cost(u) = \max(\{cost(v_1), \dots, cost(v_k)\})$   
(with  $\max(\emptyset) := 0$ )

Combination rule for proposition nodes:

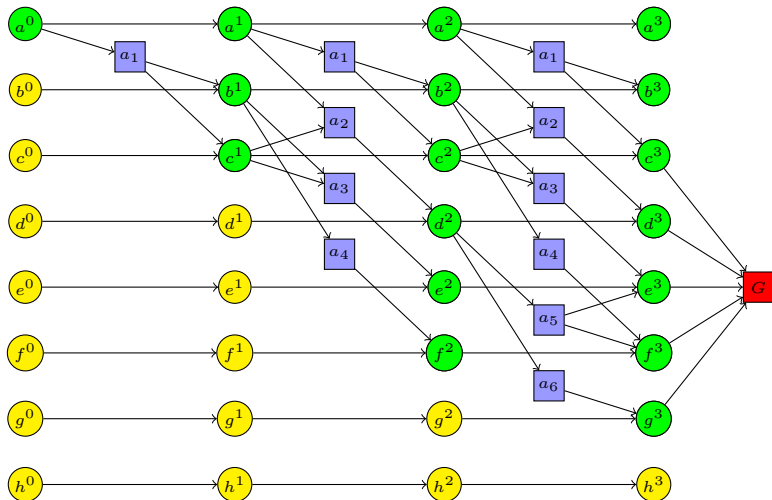
- $cost(u) = \min(\{cost(v_1), \dots, cost(v_k)\})$

In both cases,  $\{v_1, \dots, v_k\}$  is the set of immediate predecessors of  $u$ .

Intuition:

- **Action rule:** If we have to achieve several preconditions, estimate this by the **most expensive** cost.
- **Proposition rule:** If we have a choice how to achieve a proposition, pick the **cheapest** possibility.

# Running example: $h_{\max}$



Automated  
(AI) Planning

Introduction

Obtaining  
heuristics

Relaxation  
heuristics

Relaxation  
Heuristics

Template

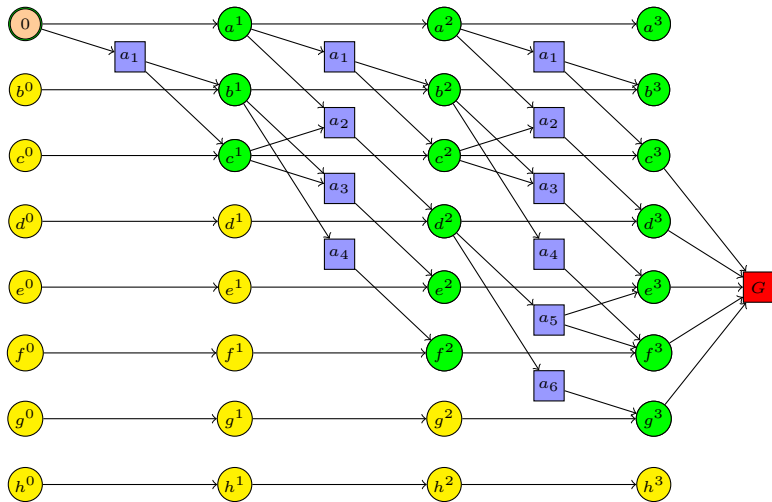
$h_{\max}$

$h_{\text{add}}$

$h_{\text{FF}}$

Comparison &  
practice

# Running example: $h_{\max}$



Automated (AI) Planning

Introduction

Obtaining heuristics

Relaxation heuristics

Relaxation Heuristics

Template

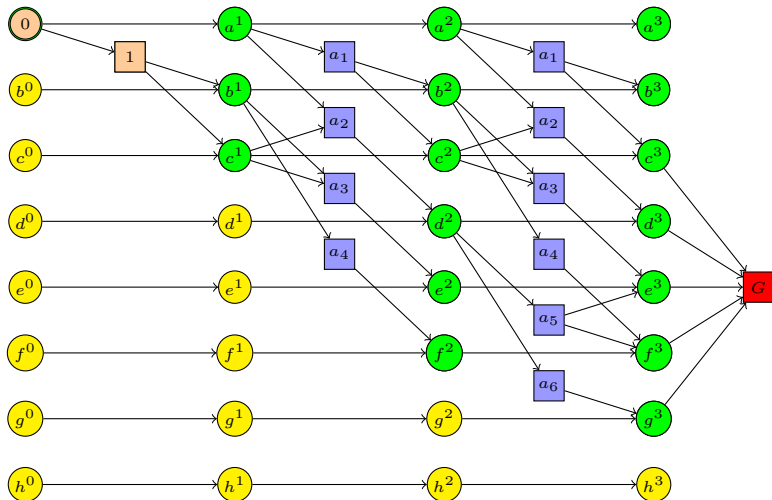
$h_{\max}$

$h_{\text{add}}$

$h_{\text{FF}}$

Comparison & practice

# Running example: $h_{\max}$



Automated  
(AI) Planning

Introduction

Obtaining  
heuristics

Relaxation  
heuristics

Relaxation  
Heuristics

Template

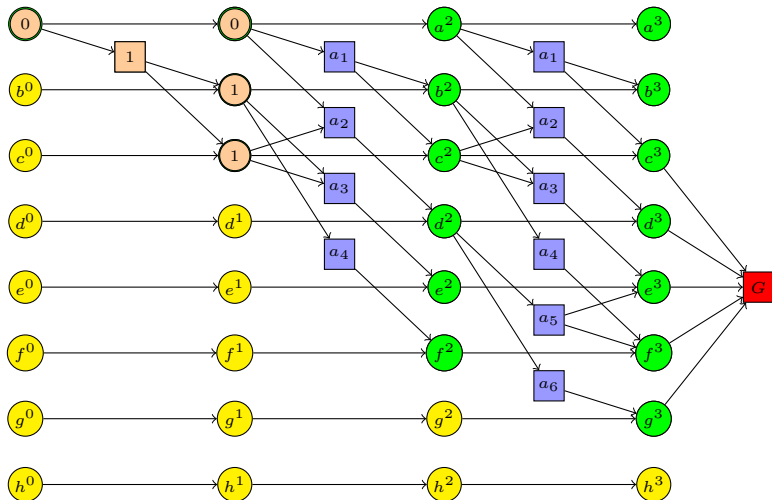
$h_{\max}$

$h_{\text{add}}$

$h_{\text{FF}}$

Comparison &  
practice

# Running example: $h_{\max}$



Automated  
(AI) Planning

Introduction

Obtaining  
heuristics

Relaxation  
heuristics

Relaxation  
Heuristics

Template

$h_{\max}$

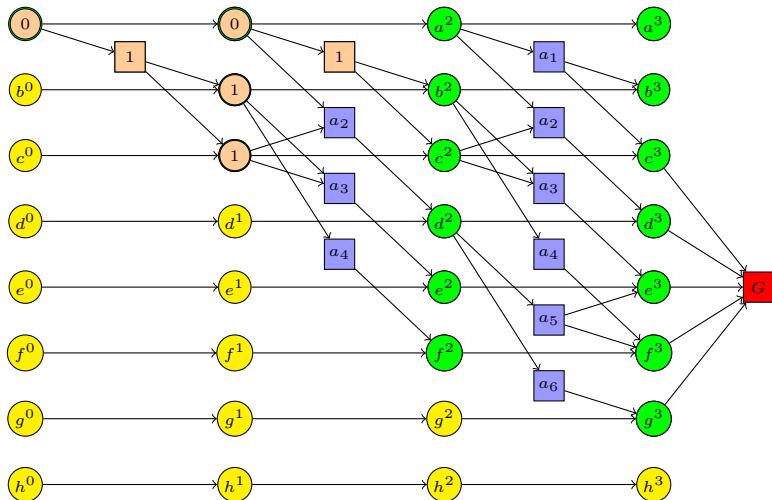
$h_{\text{add}}$

$h_{\text{FF}}$

Comparison &  
practice



# Running example: $h_{\max}$



Automated  
(AI) Planning

Introduction

Obtaining  
heuristics

Relaxation  
heuristics

Relaxation  
Heuristics

Template

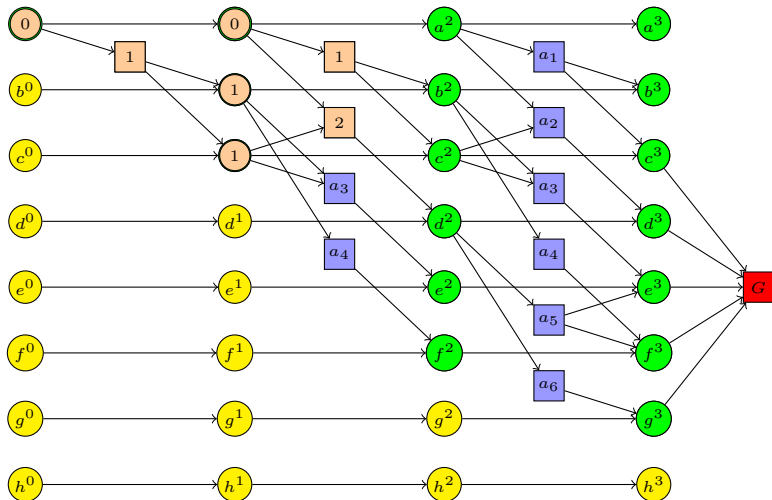
$h_{\max}$

$h_{\text{add}}$

$h_{\text{FF}}$

Comparison &  
practice

# Running example: $h_{\max}$



Automated  
(AI) Planning

Introduction

Obtaining  
heuristics

Relaxation  
heuristics

Relaxation  
Heuristics

Template

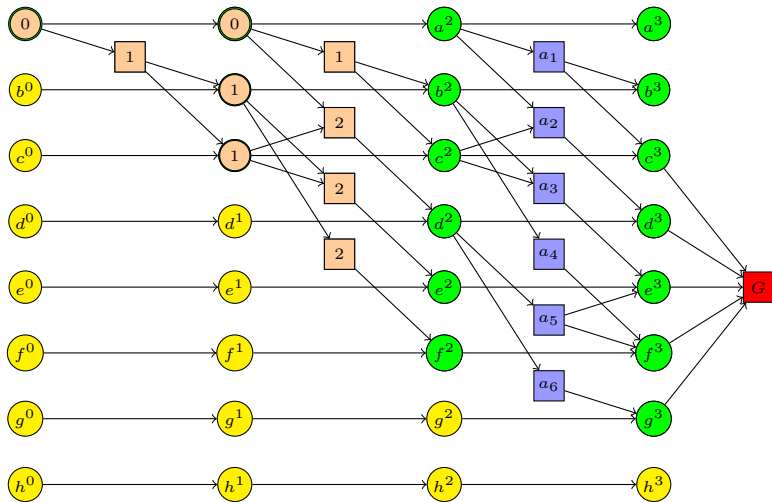
$h_{\max}$

$h_{\text{add}}$

$h_{\text{FF}}$

Comparison &  
practice

# Running example: $h_{\max}$



Automated (AI) Planning

Introduction

Obtaining heuristics

Relaxation heuristics

Relaxation Heuristics

Template

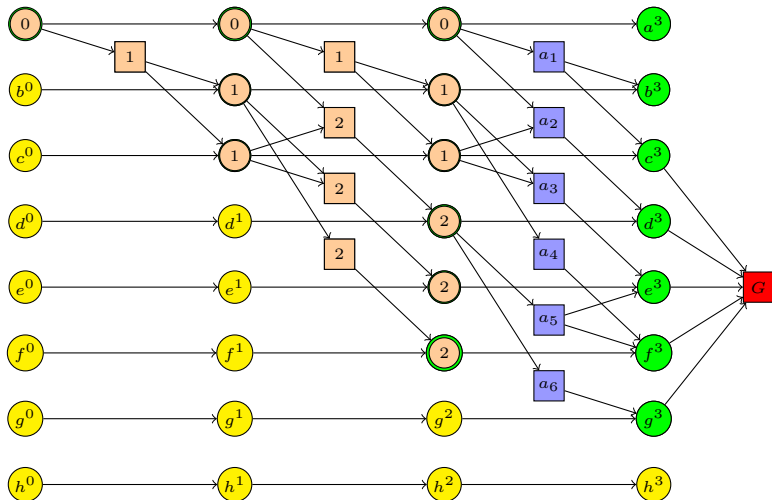
$h_{\max}$

$h_{\text{add}}$

$h_{\text{FF}}$

Comparison & practice

# Running example: $h_{\max}$



Automated  
(AI) Planning

Introduction

Obtaining  
heuristics

Relaxation  
heuristics

Relaxation  
Heuristics

Template

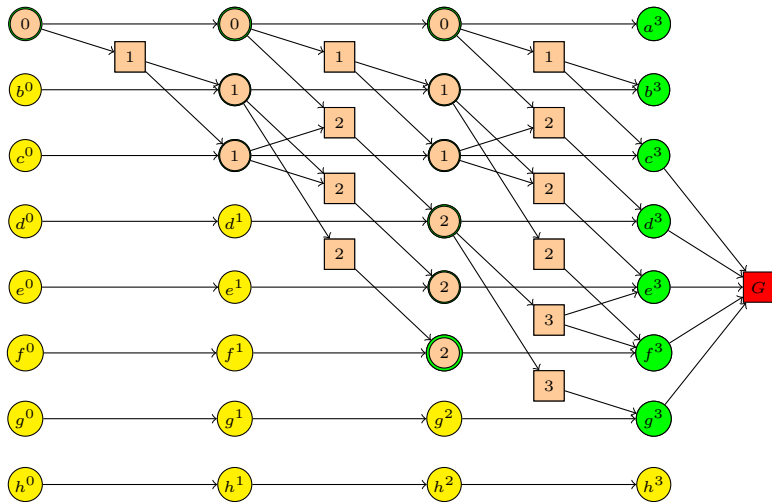
$h_{\max}$

$h_{\text{add}}$

$h_{\text{FF}}$

Comparison &  
practice

# Running example: $h_{\max}$



Automated (AI) Planning

Introduction

Obtaining heuristics

Relaxation heuristics

Relaxation Heuristics

Template

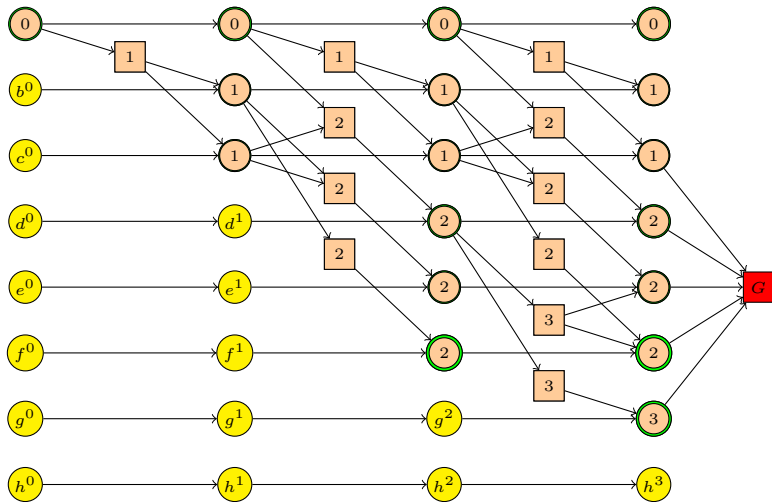
$h_{\max}$

$h_{\text{add}}$

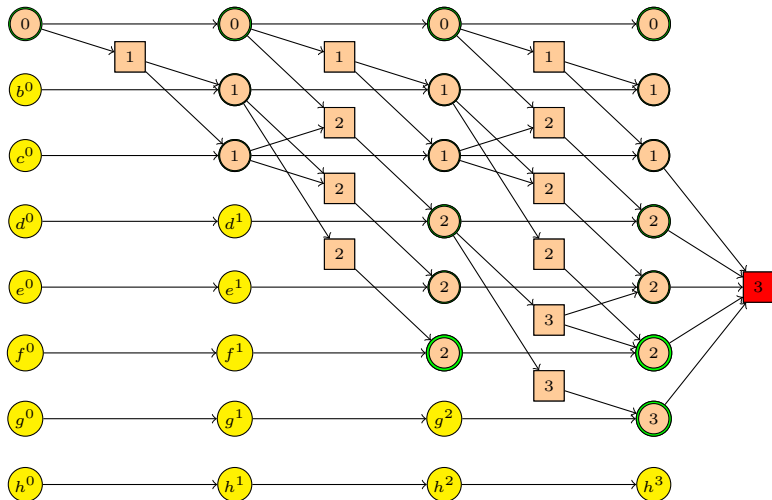
$h_{\text{FF}}$

Comparison & practice

# Running example: $h_{\max}$



# Running example: $h_{\max}$



Automated  
(AI) Planning

Introduction

Obtaining  
heuristics

Relaxation  
heuristics

Relaxation  
Heuristics

Template

$h_{\max}$

$h_{\text{add}}$

$h_{\text{FF}}$

Comparison &  
practice

# The additive heuristic

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

Combination rule for action nodes:

- $cost(u) = cost(v_1) + \dots + cost(v_k)$   
(with  $\sum(\emptyset) := 0$ )

Combination rule for proposition nodes:

- $cost(u) = \min(\{cost(v_1), \dots, cost(v_k)\})$

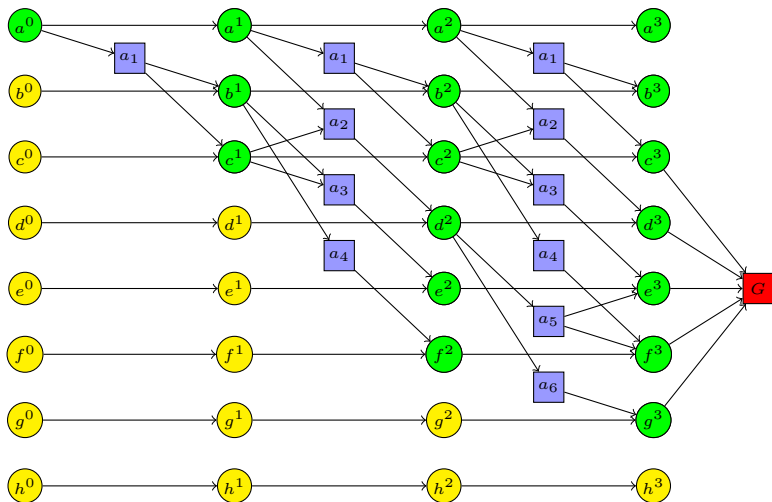
In both cases,  $\{v_1, \dots, v_k\}$  is the set of immediate predecessors of  $u$ .

Intuition:

- **Action rule:** If we have to achieve several preconditions, estimate this by the cost of achieving **each in isolation**.
- **Proposition rule:** If we have a choice how to achieve a proposition, pick the **cheapest** possibility.



# Running example: $h_{\text{add}}$



Automated  
(AI) Planning

Introduction

Obtaining  
heuristics

Relaxation  
heuristics

Relaxation  
Heuristics

Template

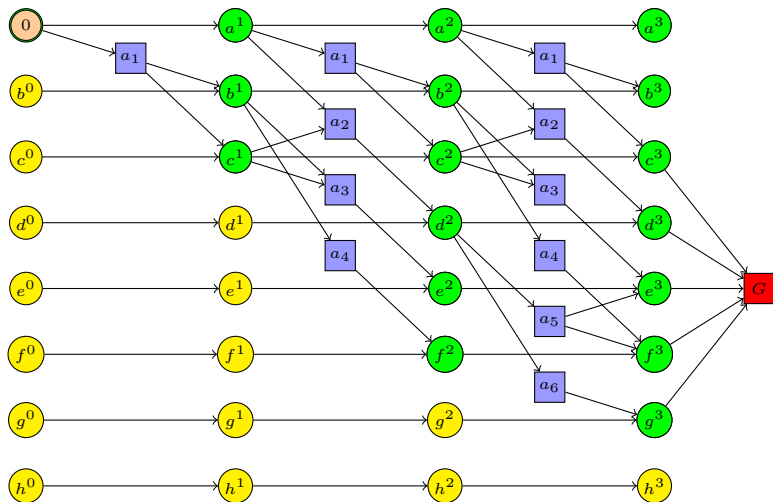
$h_{\text{max}}$

$h_{\text{add}}$

$h_{\text{FF}}$

Comparison &  
practice

# Running example: $h_{\text{add}}$



Automated  
(AI) Planning

Introduction

Obtaining  
heuristics

Relaxation  
heuristics

Relaxation  
Heuristics

Template

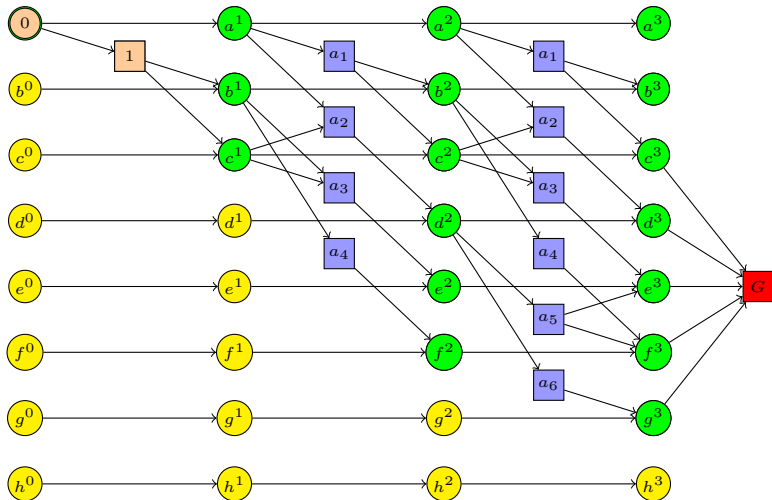
$h_{\text{max}}$

$h_{\text{add}}$

$h_{\text{FF}}$

Comparison &  
practice

# Running example: $h_{\text{add}}$



Automated  
(AI) Planning

Introduction

Obtaining  
heuristics

Relaxation  
heuristics

Relaxation  
Heuristics

Template

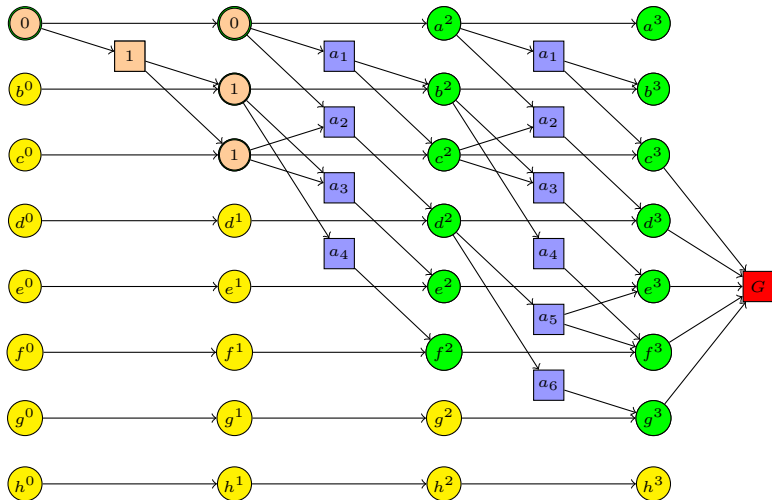
$h_{\text{max}}$

$h_{\text{add}}$

$h_{\text{FF}}$

Comparison &  
practice

# Running example: $h_{add}$



Automated  
(AI) Planning

Introduction

Obtaining  
heuristics

Relaxation  
heuristics

Relaxation  
Heuristics

Template

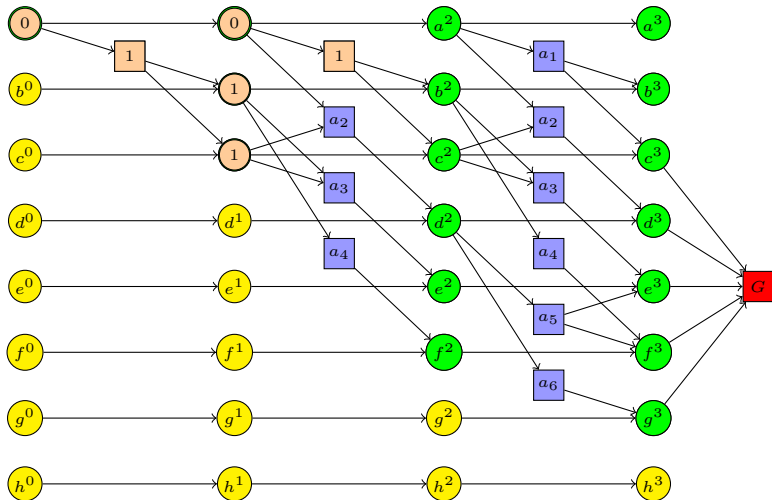
$h_{max}$

$h_{add}$

$h_{FF}$

Comparison &  
practice

# Running example: $h_{\text{add}}$



Automated  
(AI) Planning

Introduction

Obtaining  
heuristics

Relaxation  
heuristics

Relaxation  
Heuristics

Template

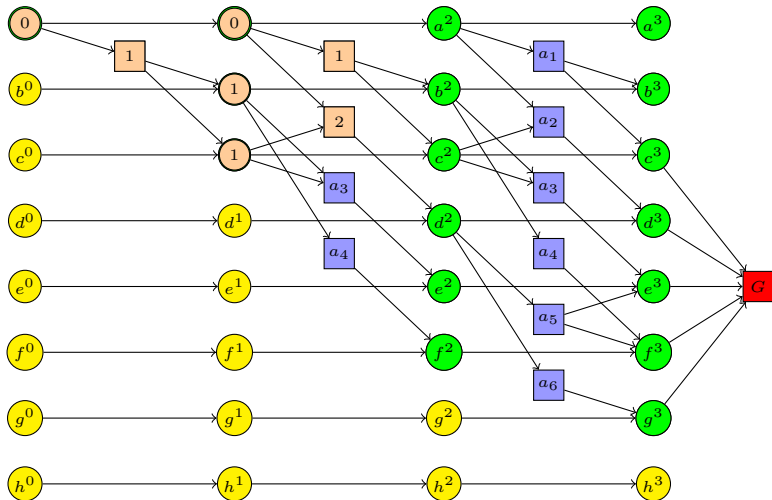
$h_{\text{max}}$

$h_{\text{add}}$

$h_{\text{FF}}$

Comparison &  
practice

# Running example: $h_{\text{add}}$



Automated  
(AI) Planning

Introduction

Obtaining  
heuristics

Relaxation  
heuristics

Relaxation  
Heuristics

Template

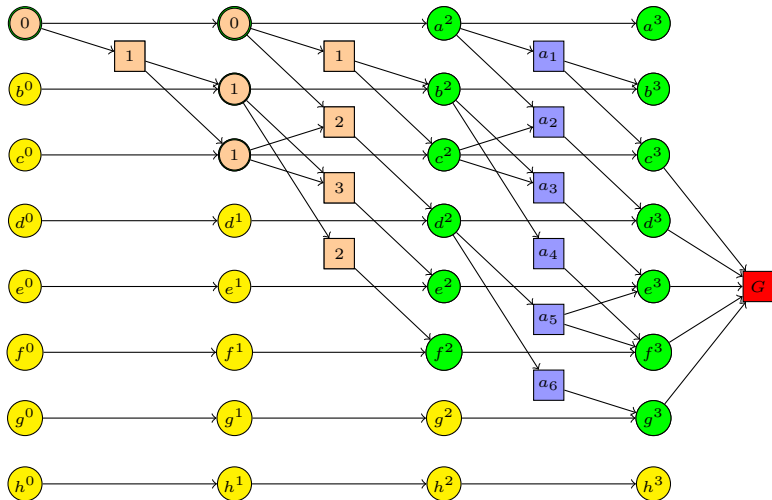
$h_{\text{max}}$

$h_{\text{add}}$

$h_{\text{FF}}$

Comparison &  
practice

# Running example: $h_{\text{add}}$



Automated  
(AI) Planning

Introduction

Obtaining  
heuristics

Relaxation  
heuristics

Relaxation  
Heuristics

Template

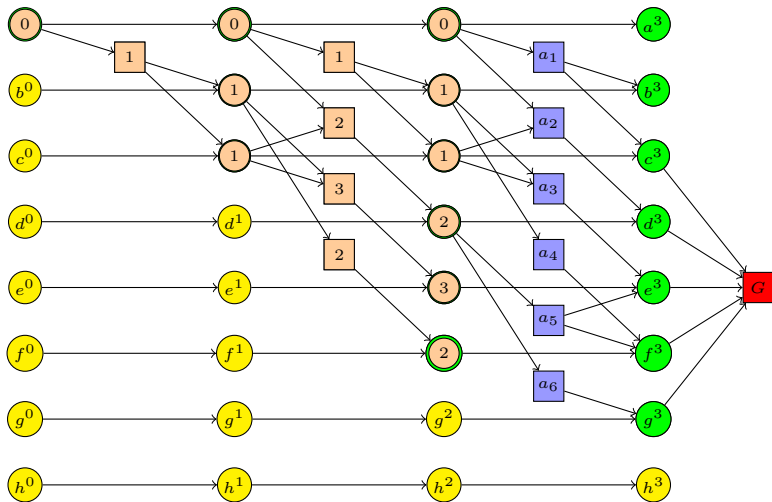
$h_{\text{max}}$

$h_{\text{add}}$

$h_{\text{FF}}$

Comparison &  
practice

# Running example: $h_{add}$



Automated  
(AI) Planning

Introduction

Obtaining  
heuristics

Relaxation  
heuristics

Relaxation  
Heuristics

Template

$h_{max}$

$h_{add}$

$h_{FF}$

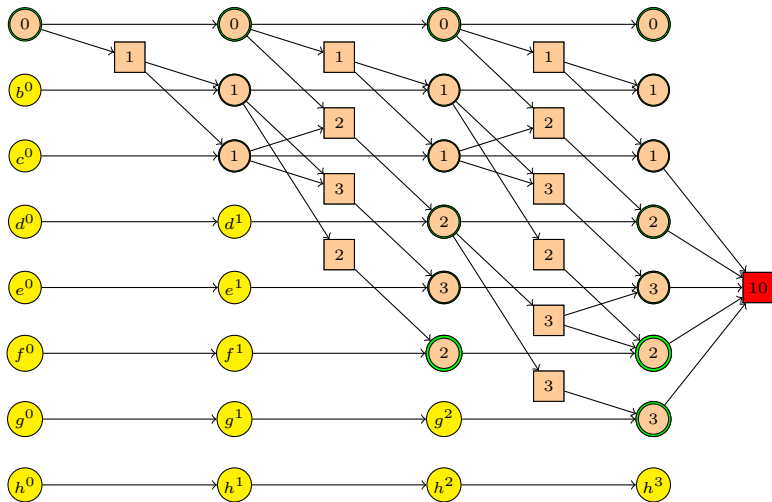
Comparison &  
practice







# Running example: $h_{add}$



Automated  
(AI) Planning

Introduction

Obtaining  
heuristics

Relaxation  
heuristics

Relaxation  
Heuristics

Template

$h_{max}$

$h_{add}$

$h_{FF}$

Comparison &  
practice

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

- $h_{\text{add}}$  is **safe** and **goal-aware**.
- Unlike  $h_{\text{max}}$ ,  $h_{\text{add}}$  is a **very informative** heuristic in many planning domains.  
Q: Intuitively, when it will be informative?
- 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.

# FF heuristic: fitting the template

## The FF heuristic $h_{FF}$

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 an action node is **justified** if all its true immediate predecessors are marked, and that a proposition node is **justified** if at least one of its immediate predecessors is marked.

...

# FF heuristic: fitting the template (ctd.)

## The FF heuristic $h_{FF}$ (ctd.)

Computing annotations:

- ...

Apply these rules until **all marked nodes are justified**:

- 1 Mark all immediate predecessors of a marked unjustified ACTION node.
- 2 Mark the immediate predecessor of a marked unjustified PROP node with only one immediate predecessor.
- 3 Mark an immediate predecessor of a marked unjustified PROP node connected via an idle arc.
- 4 Mark any immediate predecessor of a marked unjustified PROP node.

The rules are given in priority order: earlier rules are preferred if applicable.

# FF heuristic: fitting the template (ctd.)

## The FF heuristic $h_{FF}$ (ctd.)

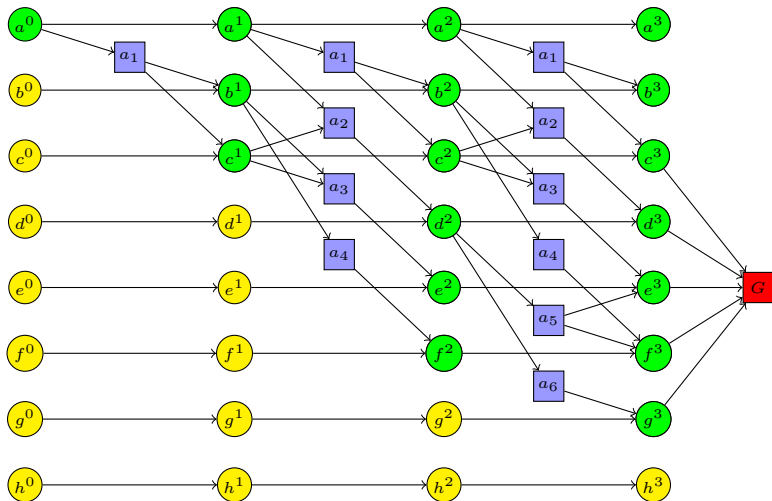
Termination criterion:

- **Always terminate** at first layer where goal node is true.

Heuristic value:

- The heuristic value is the **number of marked action nodes**.

# Running example: $h_{FF}$



Automated  
(AI) Planning

Introduction

Obtaining  
heuristics

Relaxation  
heuristics

Relaxation  
Heuristics

Template

$h_{\max}$

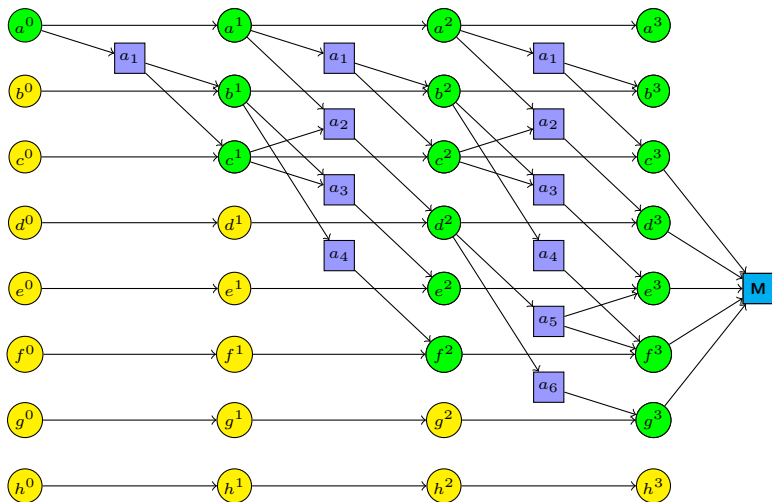
$h_{\text{add}}$

$h_{FF}$

Comparison &  
practice



# Running example: $h_{FF}$



Automated  
(AI) Planning

Introduction

Obtaining  
heuristics

Relaxation  
heuristics

Relaxation  
Heuristics

Template

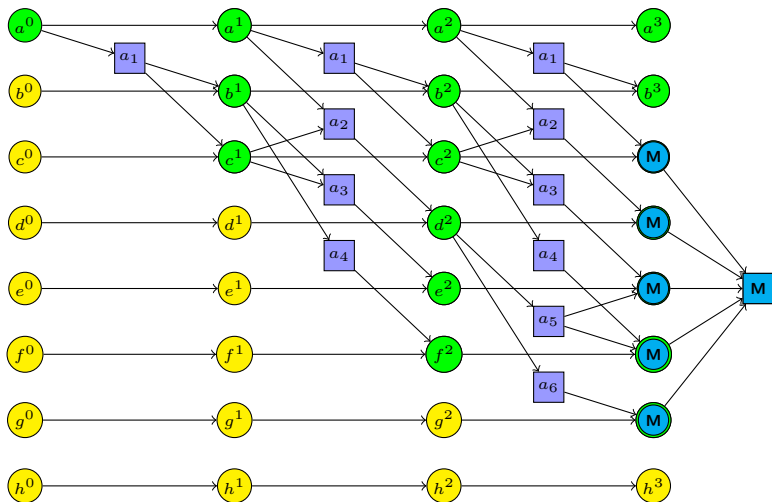
$h_{\max}$

$h_{\text{add}}$

$h_{FF}$

Comparison &  
practice

# Running example: $h_{FF}$



Automated  
(AI) Planning

Introduction

Obtaining  
heuristics

Relaxation  
heuristics

Relaxation  
Heuristics

Template

$h_{\max}$

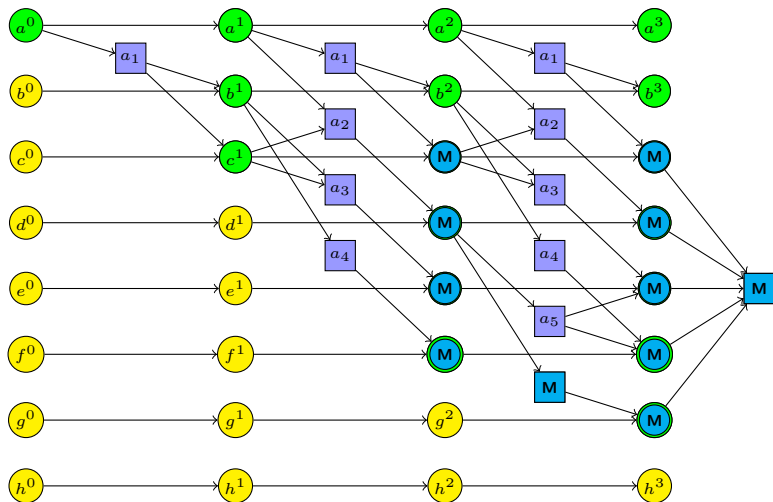
$h_{\text{add}}$

$h_{FF}$

Comparison &  
practice



# Running example: $h_{FF}$



Automated  
(AI) Planning

Introduction

Obtaining  
heuristics

Relaxation  
heuristics

Relaxation  
Heuristics

Template

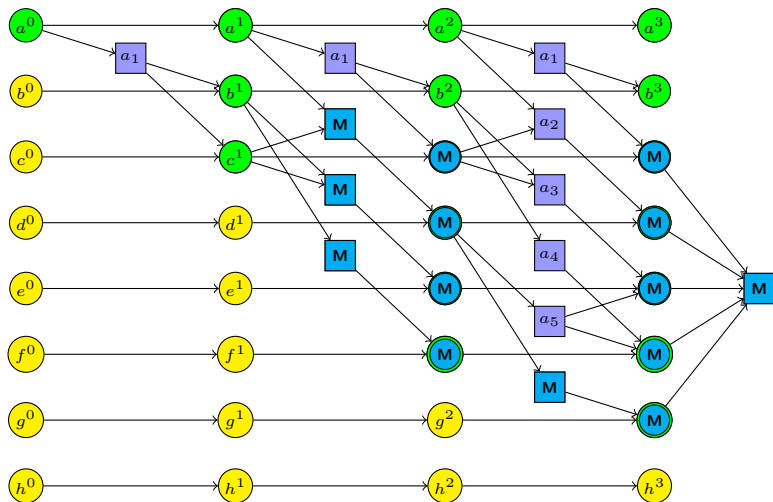
$h_{\max}$

$h_{\text{add}}$

$h_{FF}$

Comparison &  
practice

# Running example: $h_{FF}$



Automated  
(AI) Planning

Introduction

Obtaining  
heuristics

Relaxation  
heuristics

Relaxation  
Heuristics

Template

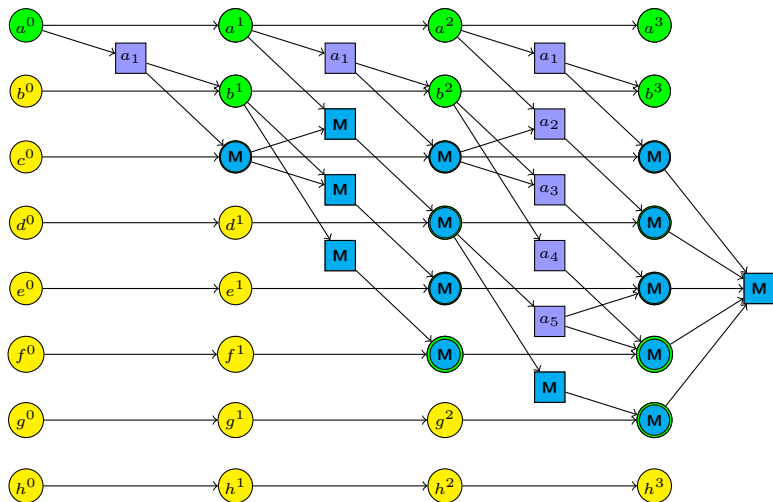
$h_{\max}$

$h_{\text{add}}$

$h_{FF}$

Comparison &  
practice

# Running example: $h_{FF}$



Automated  
(AI) Planning

Introduction

Obtaining  
heuristics

Relaxation  
heuristics

Relaxation  
Heuristics

Template

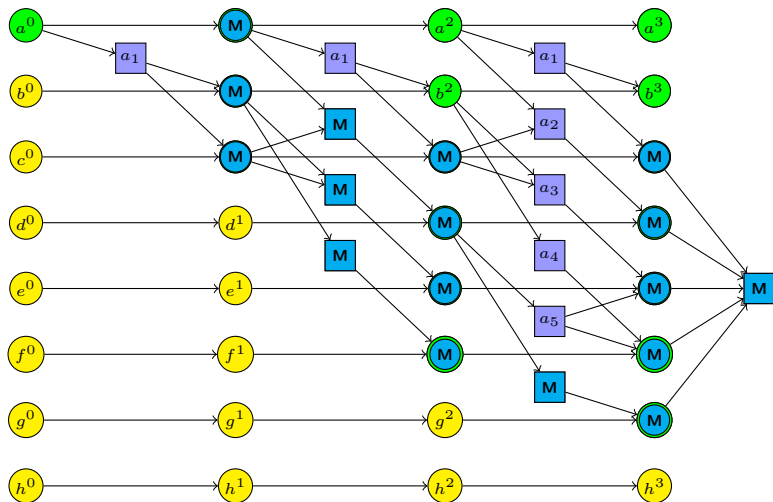
$h_{\max}$

$h_{\text{add}}$

$h_{FF}$

Comparison &  
practice

# Running example: $h_{FF}$



Automated  
(AI) Planning

Introduction

Obtaining  
heuristics

Relaxation  
heuristics

Relaxation  
Heuristics

Template

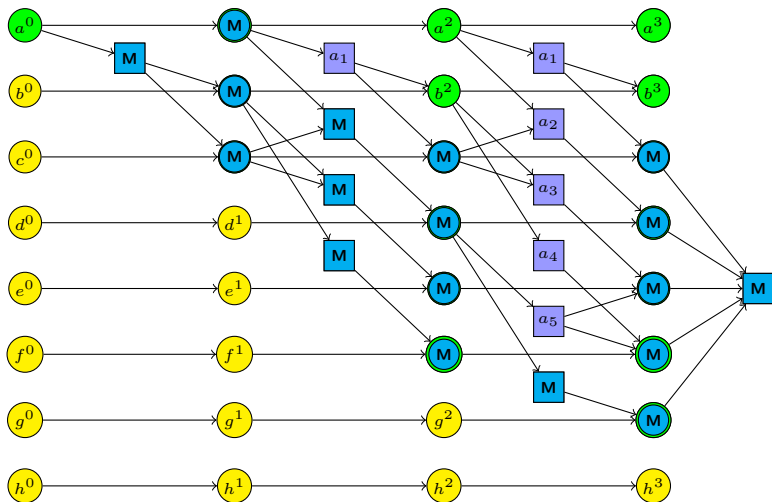
$h_{\max}$

$h_{\text{add}}$

$h_{FF}$

Comparison &  
practice

# Running example: $h_{FF}$



Automated  
(AI) Planning

Introduction

Obtaining  
heuristics

Relaxation  
heuristics

Relaxation  
Heuristics

Template

$h_{\max}$

$h_{\text{add}}$

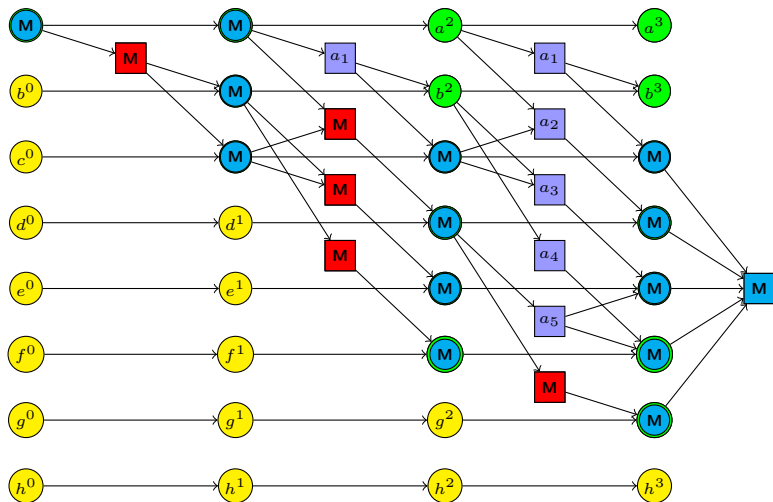
$h_{FF}$

Comparison &  
practice





# Running example: $h_{FF}$



Automated  
(AI) Planning

Introduction

Obtaining  
heuristics

Relaxation  
heuristics

Relaxation  
Heuristics

Template

$h_{\max}$

$h_{\text{add}}$

$h_{FF}$

Comparison &  
practice

# Remarks on $h_{FF}$

- Like  $h_{add}$ ,  $h_{FF}$  is **safe** and **goal-aware**, but neither **admissible** nor **consistent**.
- Always more accurate than  $h_{add}$  with respect to  $h^+$ .
  - Marked actions define a **relaxed plan**.
- $h_{FF}$  can be computed in **linear time**.
  - The  $h_{FF}$  value depends on tie-breaking when the marking rules allow several possible choices, so  $h_{FF}$  is **not well-defined** without specifying the tie-breaking rule.
  - The best implementations of FF use additional rules of thumb to try to reduce the size of the generated relaxed plan.

# Comparison of relaxation heuristics

## Relationship between relaxation heuristics

Let  $s$  be a state of planning task  $\langle P, I, O, G \rangle$ . Then:

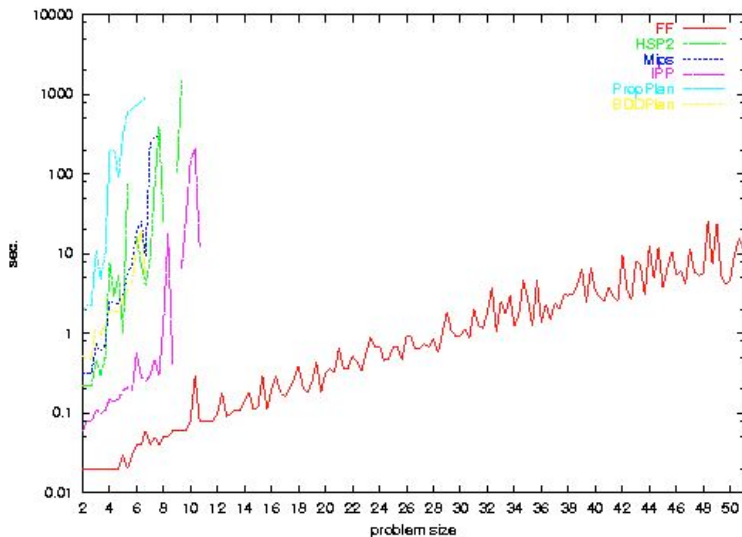
- $h_{\max}(s) \leq h^+(s) \leq h^*(s)$
- $h_{\max}(s) \leq h^+(s) \leq h_{\text{FF}}(s) \leq h_{\text{add}}(s)$
- $h^*$  and  $h_{\text{FF}}$  are pairwise incomparable
- $h^*$  and  $h_{\text{add}}$  are incomparable

Moreover,  $h^+$ ,  $h_{\max}$ ,  $h_{\text{add}}$ , 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.

# Does the heuristic really matter?

Example: The 2nd Planning Competition; Schedule domain



Automated  
(AI) Planning

Introduction

Obtaining  
heuristics

Relaxation  
heuristics

Relaxation  
Heuristics

Template

$h_{max}$

$h_{add}$

$h_{FF}$

Comparison &  
practice