

# Heuristic Search Planner Techniques

Dra. M<sup>a</sup> Dolores Rodríguez Moreno

# Objectives

## Specific Objectives

- Heuristic search planning techniques

## Source

- Stuart Russell & Peter Norvig (2009). Artificial Intelligence: A Modern Approach. Chapter 10. (3rd Edition). Ed. Pearsons
- Dana Nau's slides for Automated Planning. Licensed under License <https://creativecommons.org/licenses/by-nc-sa/2.0/>
- Jorg Hoffmann. FF: The Fast-Forward Planning System. AAAI'01

# Outline

- **Introduction**
- **Heuristics**
  - Max-cost and additive cost
  - Delete-Relaxation
  - Landmarks
- **FF**
- **Conclusions**

# Introduction

- Heuristic Search Planners (HSP) transform planning problems into heuristic search problems extracting heuristics functions, rather than enter them by hand
- Problems
  - Number of explored nodes is very high
  - Heuristic calculation in each step
- Examples of heuristics:
  - HSP: additive ( $h^{\text{add}}$ )
  - FF: delete-relaxation

# Outline

- Introduction
- **Heuristics**
  - Max-cost and additive cost
  - Delete-Relaxation
  - Landmarks
- FF
- Conclusions

# Heuristics

- Returns an estimate  $h(s)$  of the minimum cost  $h^*(s)$  of getting from the state  $s$  to a goal state
- If the algorithm always finds optimal solutions, then the heuristic function will be admissible
- Best way of producing them: RELAXATION
  - Weakening some of the constraints
- They can be domain-specific or domain-independent
  - Max-Cost and Additive Cost
  - Delete-Relaxation
  - Landmarks

# Outline

- Introduction
- Heuristics
  - **Max-cost and additive cost**
  - Delete-Relaxation
  - Landmarks
- FF
- Conclusions

# Max-cost heuristic

- The max-cost of a set of literals  $g = \{g_1, \dots, g_k\}$  is defined recursively as the largest max-cost of each  $g_i$  individually
- Computation of  $h^{\max}$  can be visualized as an And/Or search going backward from  $g$
- Admissible
- Planning: a goal (i.e., a set of literals such as  $g$  or the preconditions of an action) can be reached by achieving just one of the goal's literals, namely, the one that is the most expensive to achieve



# Additive heuristic

- $H^{\text{add}}$  is similar to  $h^{\text{max}}$  but adds the costs of each set of literals rather than taking their maximum
- Not admissible (better in practice)

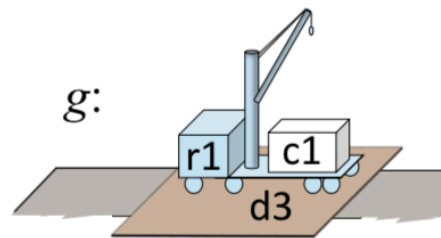
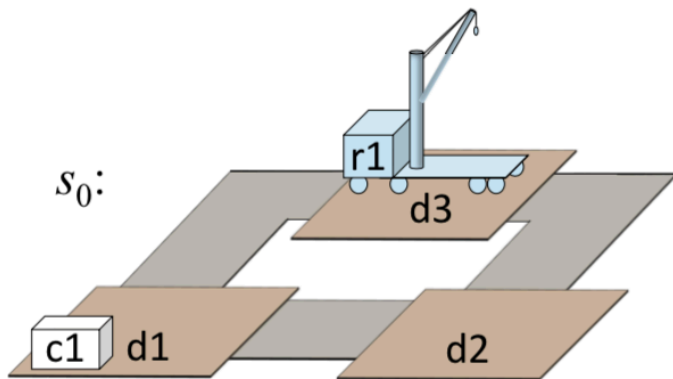
# Heuristic calculation: example (I)

$B = Robots \cup Docks \cup Containers \cup \{nil\};$   
 $Robots = \{r_1\};$   
 $Docks = \{d_1, d_2, d_3\};$   
 $Containers = \{c_1\}.$

$load(r, c, l)$   
 pre:  $cargo(r) = nil, loc(c) = l, loc(r) = l$   
 eff:  $cargo(r) \leftarrow c, loc(c) \leftarrow r$   
 cost: 1

$unload(r, c, l)$   
 pre:  $cargo(r) = c, loc(r) = l$   
 eff:  $cargo(r) \leftarrow nil, loc(c) \leftarrow l$   
 cost: 1

$move(r, d, e)$   
 pre:  $loc(r) = d$   
 eff:  $loc(r) \leftarrow e$   
 cost: 1

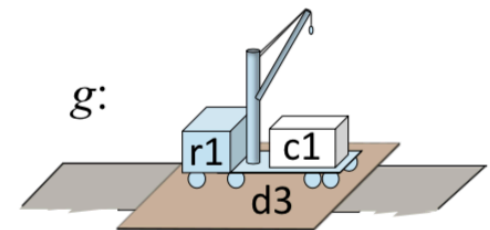
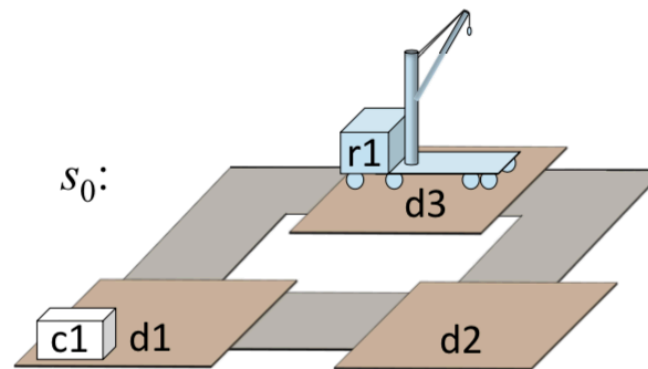


$s_0 = \{loc(r1) = d3, cargo(r1) = nil, loc(c1) = d1\};$   
 $g = \{loc(r1) = d3, loc(c1) = r1\}.$

## Heuristic calculation: example (II)

- Actions?

- $a_1 = \text{move}(r_1, d_3, d_1)$
- $a_2 = \text{move}(r_1, d_3, d_2)$

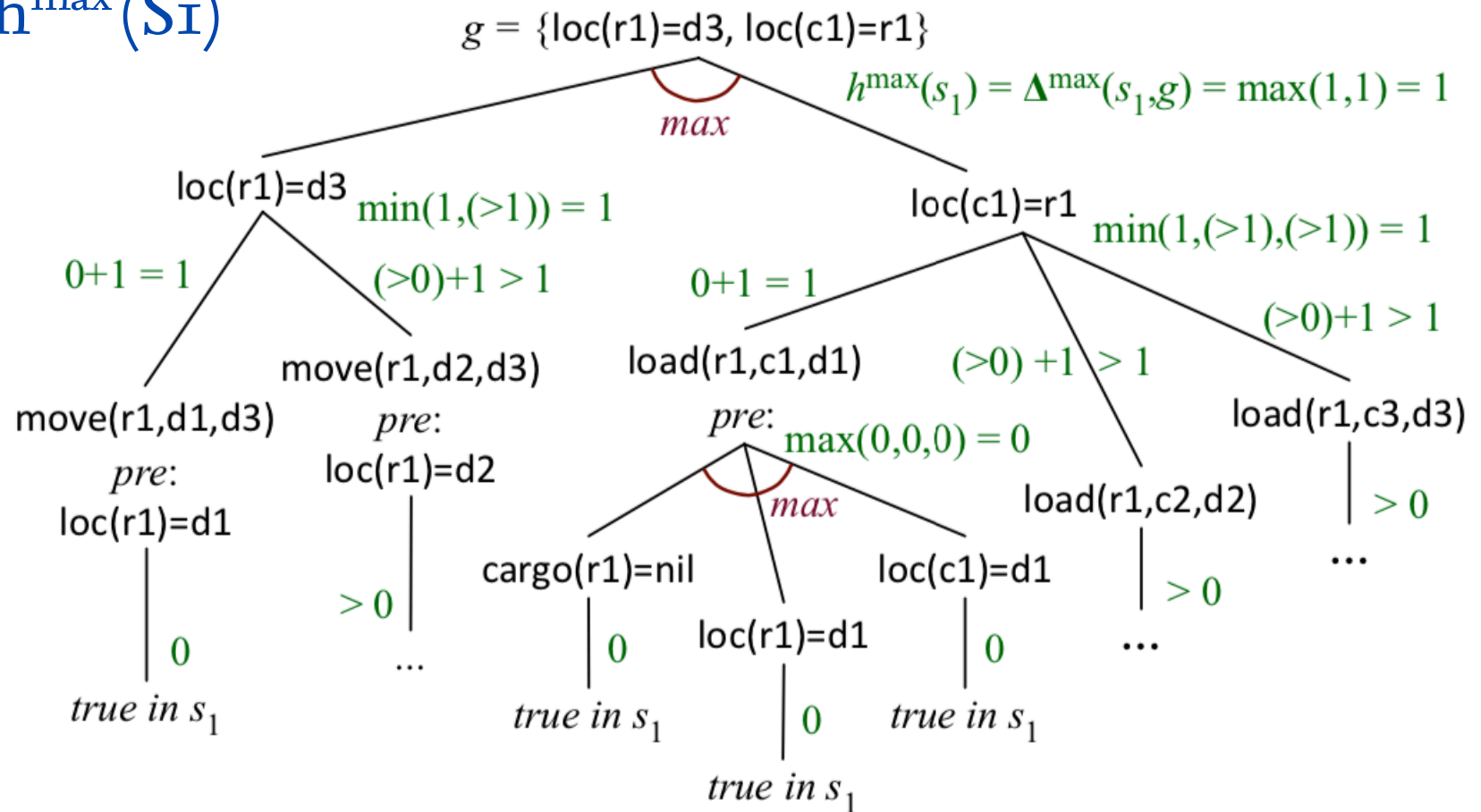


$s_1 = \gamma(s_0, a_1) = \{\text{loc}(r_1) = d_1, \text{carg}(r_1) = \text{nil}, \text{loc}(c_1) = d_1\}$

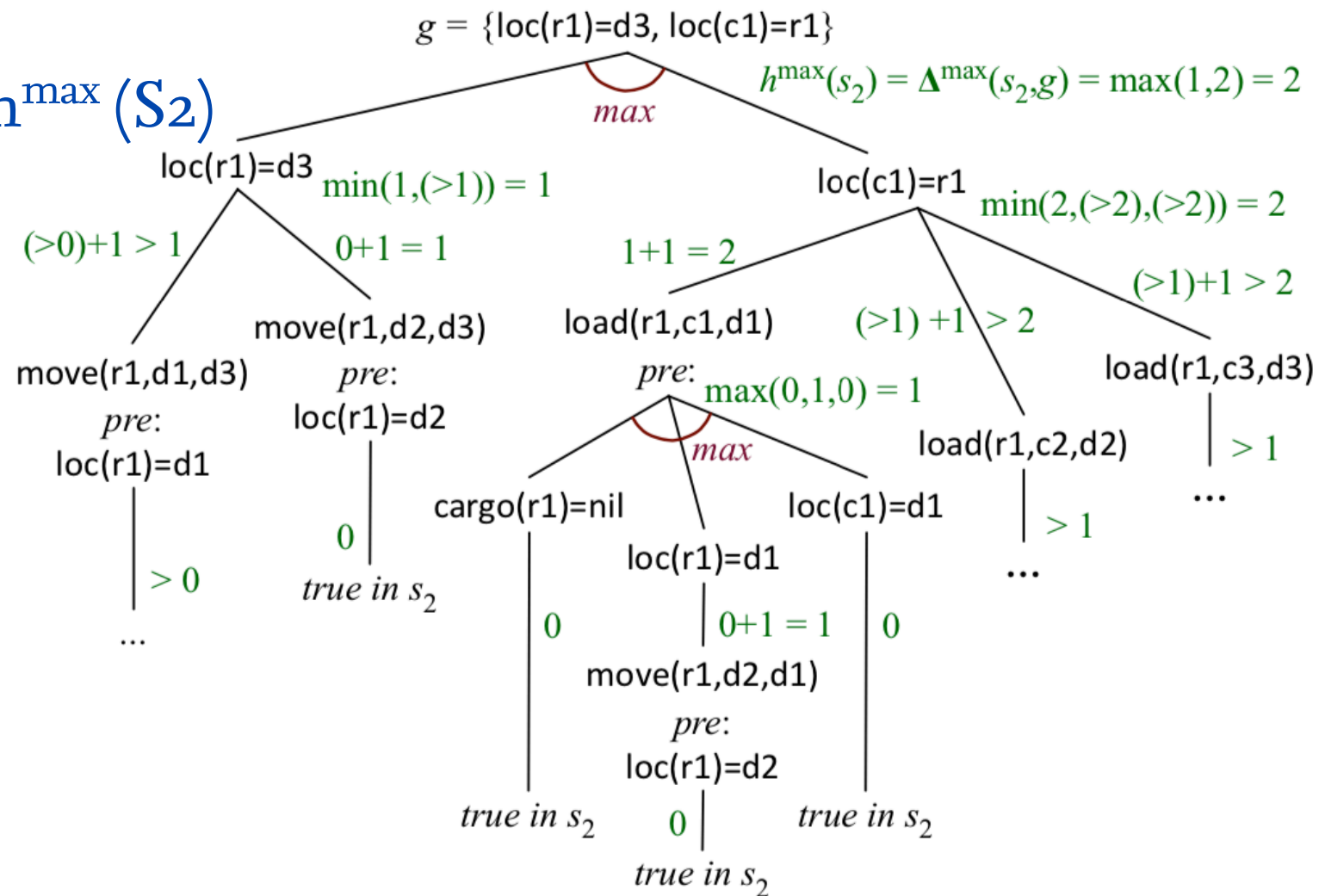
$s_2 = \gamma(s_0, a_2) = \{\text{loc}(r_1) = d_2, \text{carg}(r_1) = \text{nil}, \text{loc}(c_1) = d_1\}$

Can you calculate the value of  $h^{\text{add}}$  and  $h^{\text{max}}$ ?

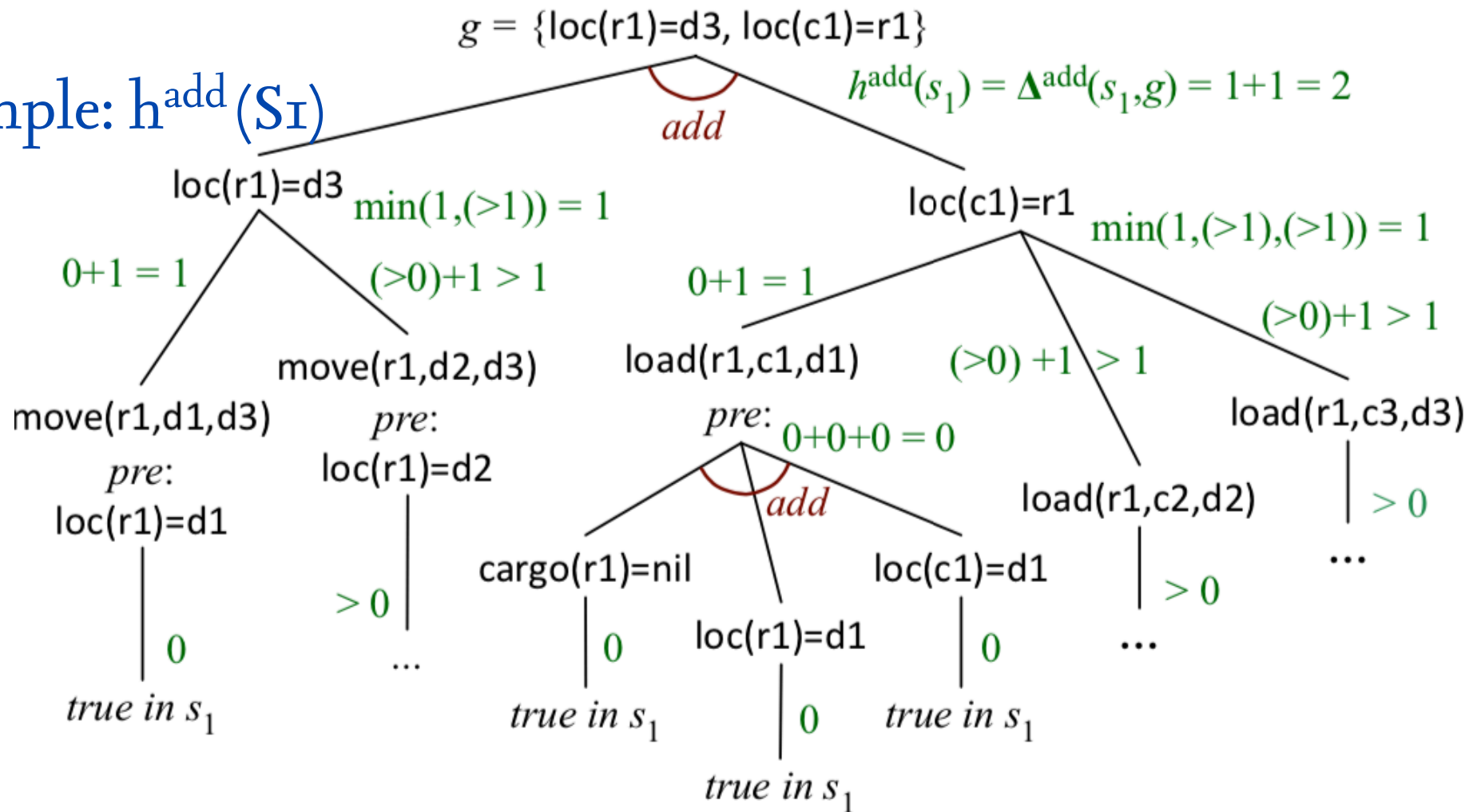
# Example: $h^{\max}(S_I)$



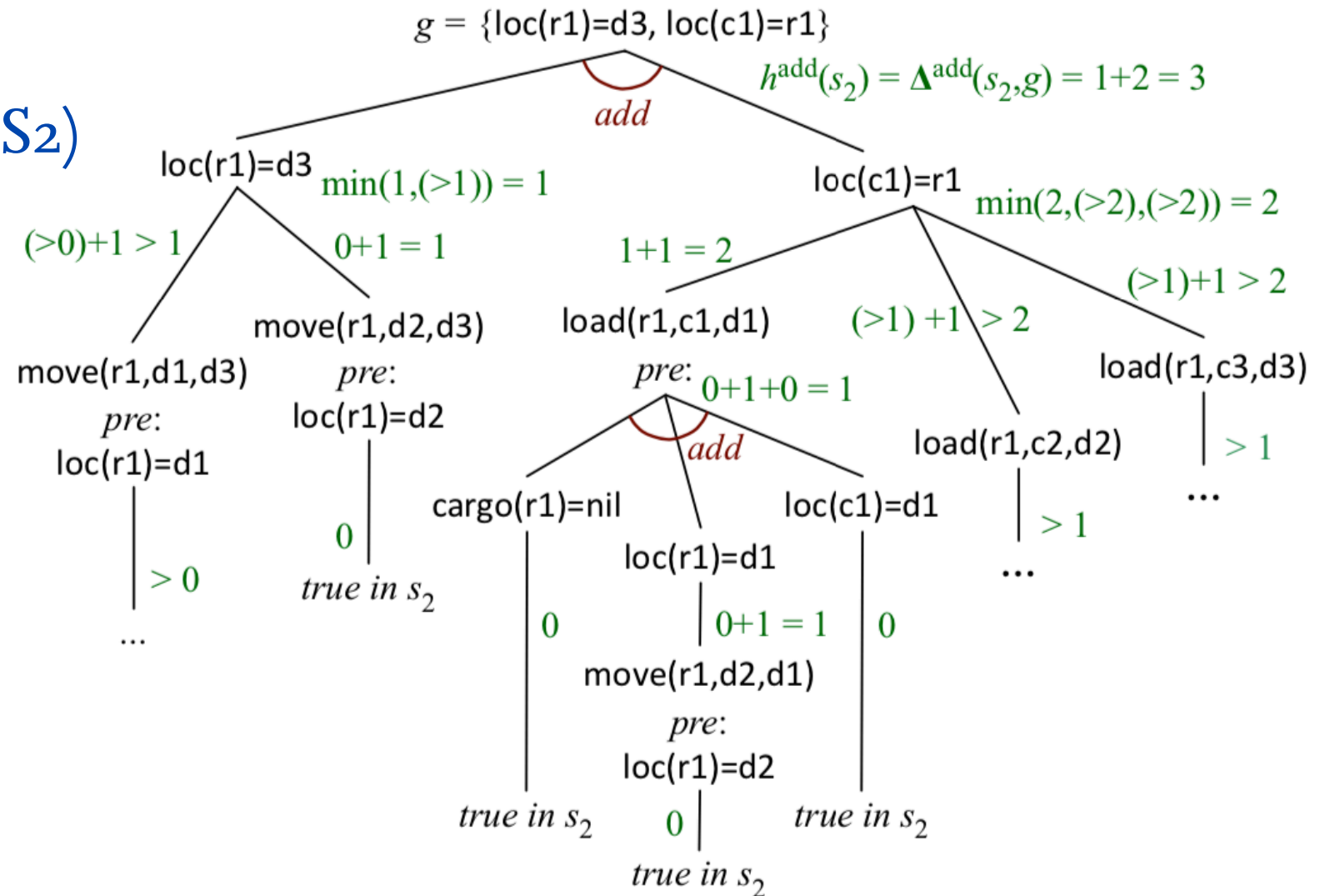
## Example: $h^{\max}(S_2)$



Example:  $h^{\text{add}}(S_I)$



# Example: $h^{\text{add}}(S_2)$



# Outline

- Introduction
- Heuristics
  - Max-cost and additive cost
  - **Delete-Relaxation**
  - Landmarks
- FF
- Conclusions



# Delete-relaxation

- $h^+$ : applies new actions that never removes old atoms from a state (simply adds new ones)
- Admissible
- Use in combination with a relaxed GP

# Outline

- Introduction
- Heuristics
  - Max-cost and additive cost
  - Delete-Relaxation
  - **Landmarks**
- FF
- Conclusions

# Landmarks

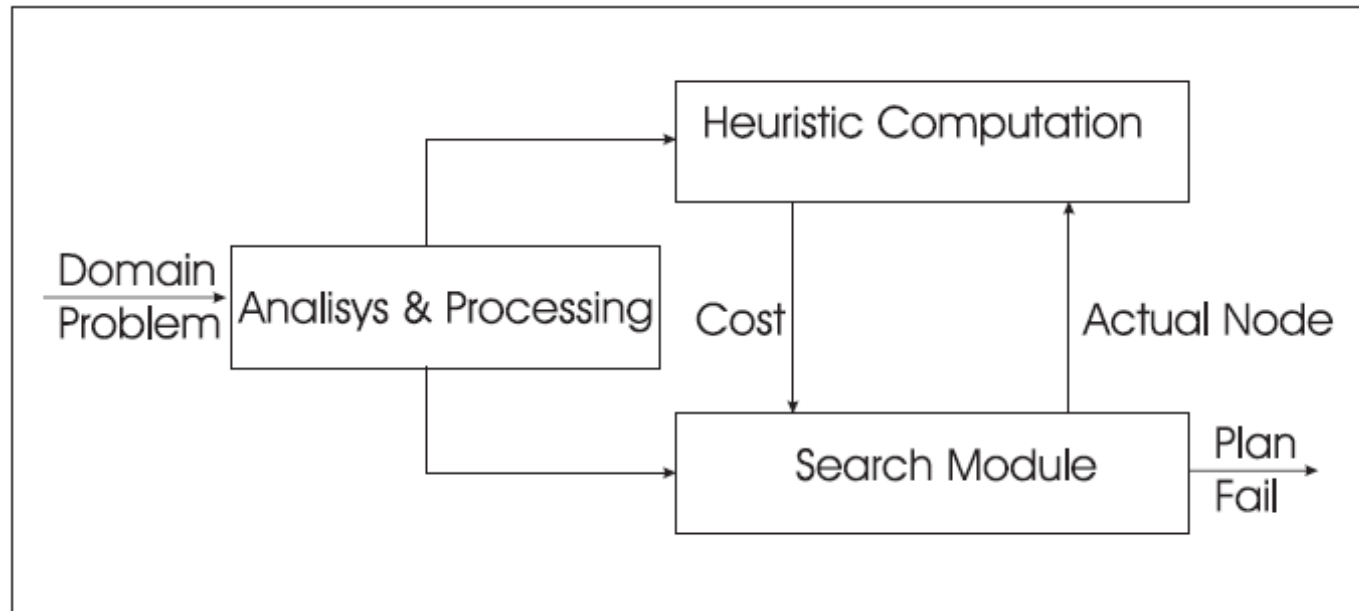
- To compute  $h^{sl}(s_1)$ , we count the number of landmarks between  $s_1$  and  $g$
- If we start in  $s_1$ , then every solution plan must include a state in which  $cargo(r_1) = c_1$ 
  - This is the only landmark for  $s_1$
  - $h^{sl}(s_1) = 1$
- If we start in state  $s_2$ , then the landmark computation will find two landmarks:
  - $cargo(s_1) = c_1$
  - $loc(r_1) = d_1$
  - $h^{sl}(s_2) = 2$

# Outline

- Introduction
- Heuristics
  - Max-cost and additive cost
  - Delete-Relaxation
  - Landmarks
- **FF: Fast-Forward**
- Conclusions

# FF (I)

- General architecture

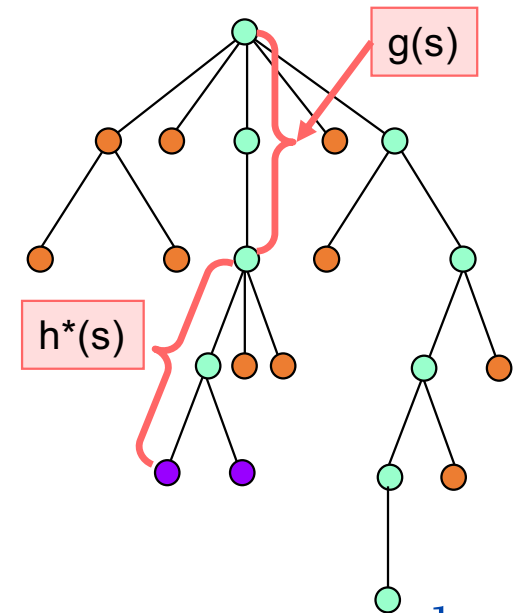


## FF (II)

- **The Analysis & Processing module.** Analyze and process all the information from the domain and the initial state (table or vector with all the possible operators instantiated)
- **The Heuristic Computation module.** Computes the cost of applying a determined node in the search process
- **The Search module.** Depends on the heuristic computation, uses a search algorithm or a combination of them

# FF search: A\*

- For every state  $s$ , let
  - $g(s)$  = cost of the path from  $s_0$  to  $s$
  - $h^*(s)$  = least cost of all paths from  $s$  to goal nodes
  - $f^*(s) = g(s) + h^*(s)$  = least cost of all paths from  $s_0$  to goal nodes that go through  $s$
- Suppose  $h(s)$  is an estimate of  $h^*(s)$ 
  - Let  $f(s) = g(s) + h(s)$
  - $h$  is *admissible* if for every state  $s$ ,  $0 \leq h(s) \leq h^*(s)$
  - If  $h$  is admissible then  $f$  is a lower bound on  $f^*$  and A\* guarantees optimality
- In combination with Enforced Hill Climbing



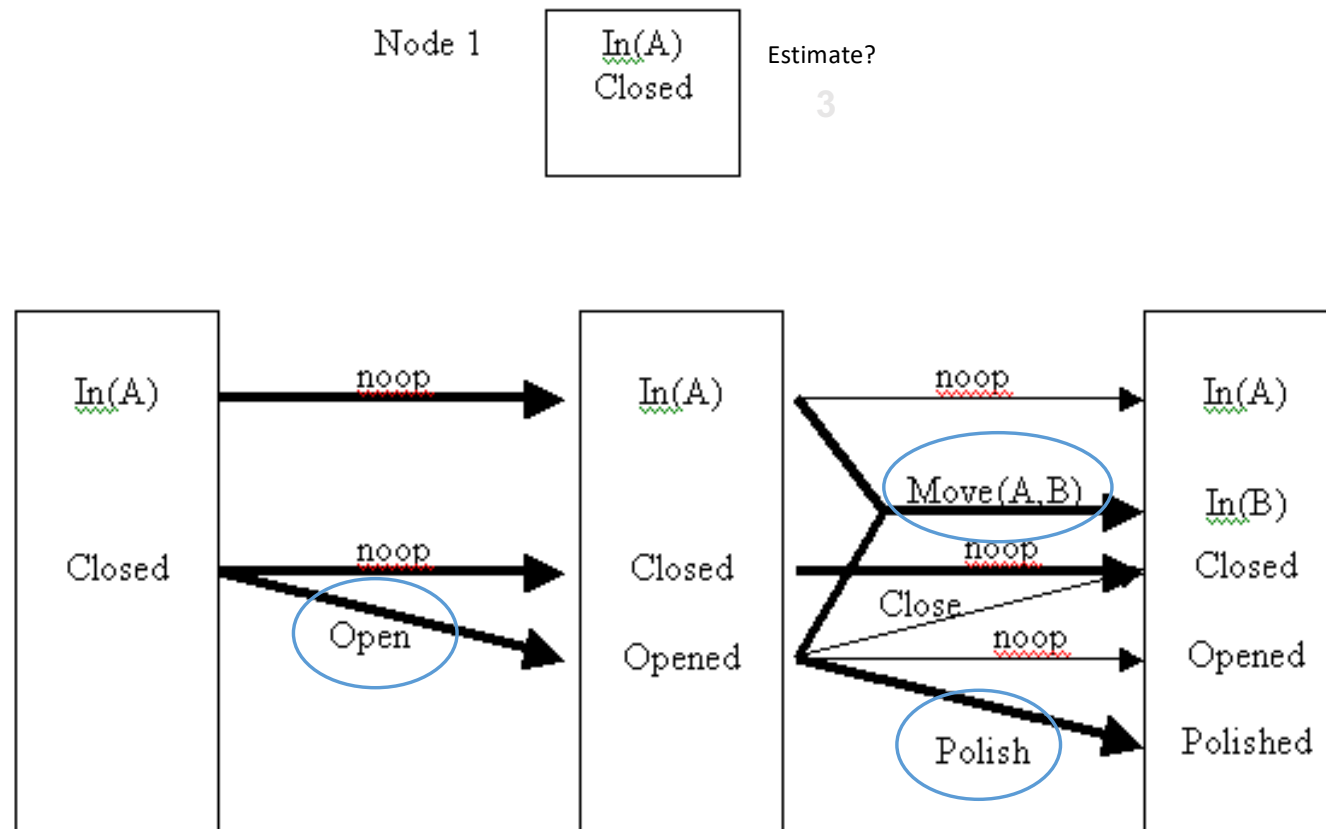
FF: h

<p><b>Move</b> <u>in(x,y)</u></p> <p><b>Pre:</b> <u>in(x)</u> <u>opened</u></p> <p><b>Add:</b> <u>in(y)</u></p> <p><b>Del:</b> <u>in(x)</u></p> <p><b>Open</b></p> <p><b>Pre:</b> closed</p> <p><b>Add:</b> opened</p> <p><b>Del:</b> closed</p>	<p><b>Close</b></p> <p><b>Pre:</b> opened</p> <p><b>Add:</b> closed</p> <p><b>Del:</b> opened</p> <p><b>Polish</b></p> <p><b>Pre:</b> opened</p> <p><b>Add:</b> polished</p> <p><b>Del:</b></p>
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

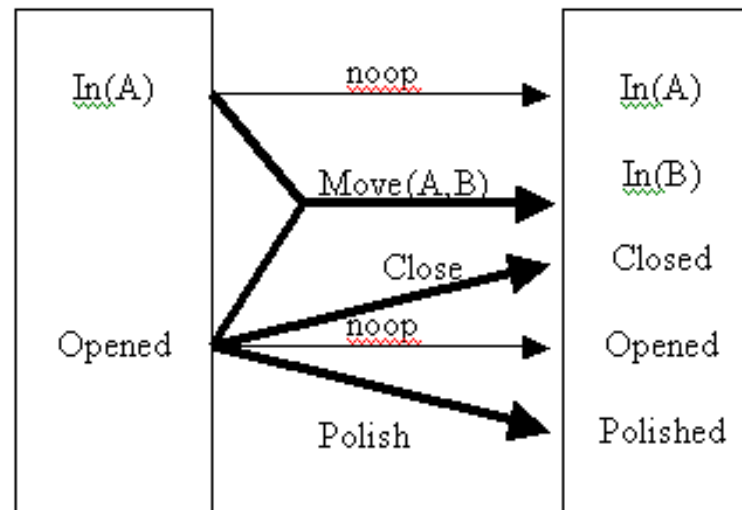
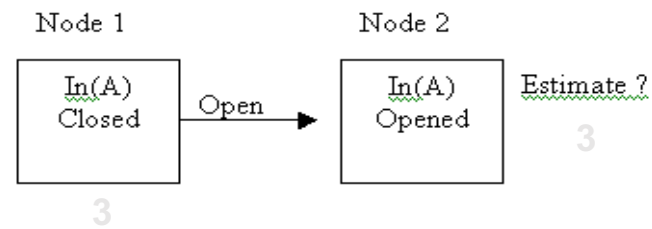
$\underline{\text{In}}(\text{A})$   
 Closed
  $\longrightarrow$ 
 $\underline{\text{In}}(\text{B})$   
 Closed  
 Polished



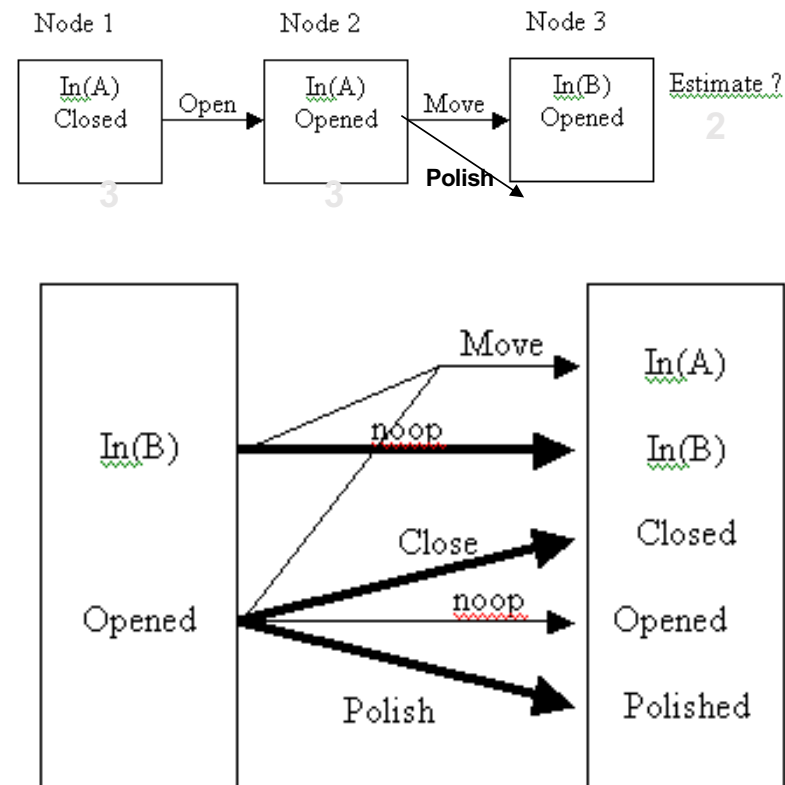
FF: h



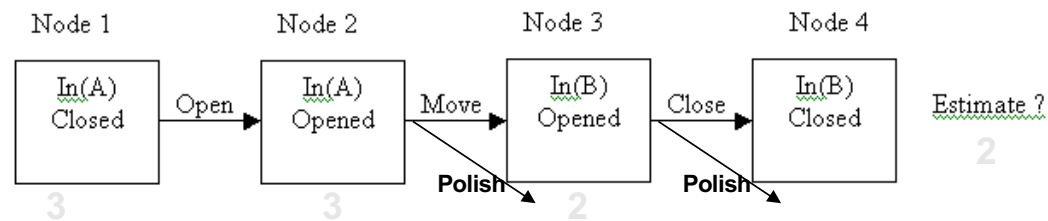
FF: h



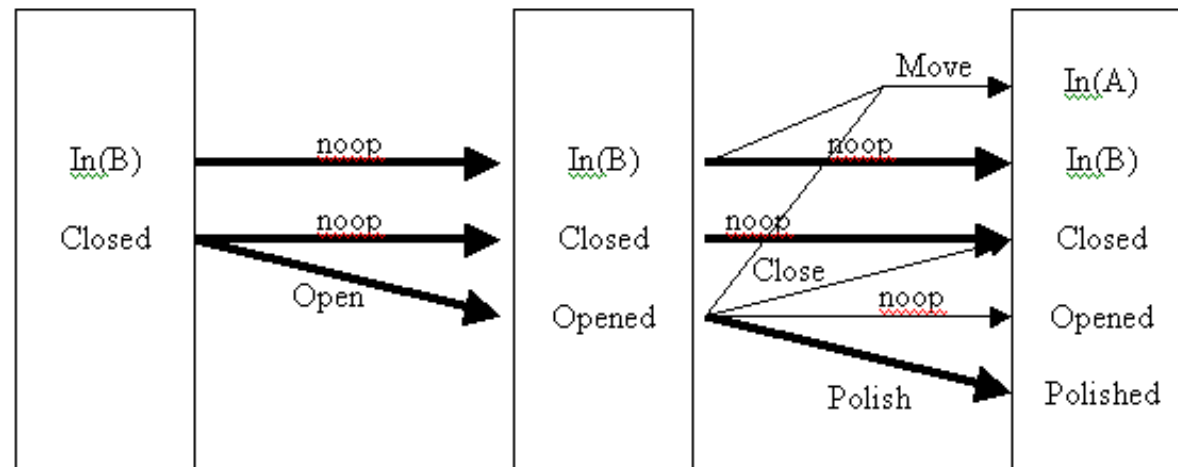
FF: h



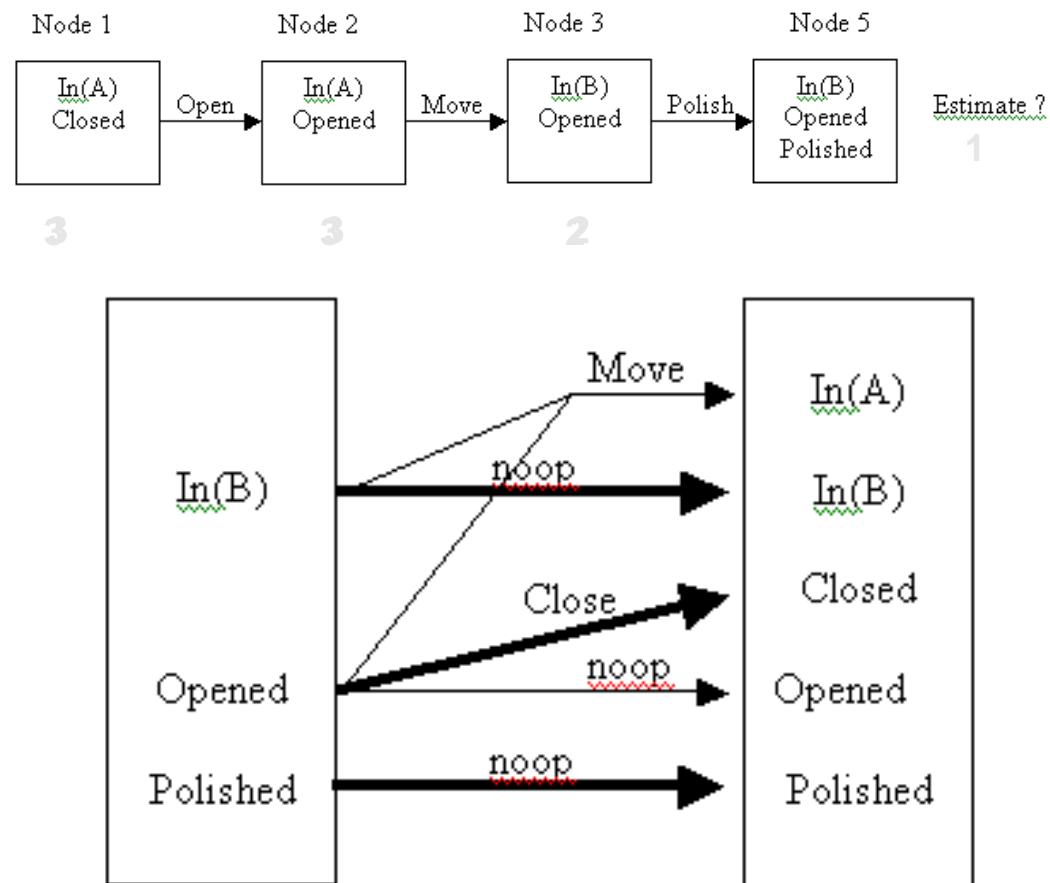
FF: h



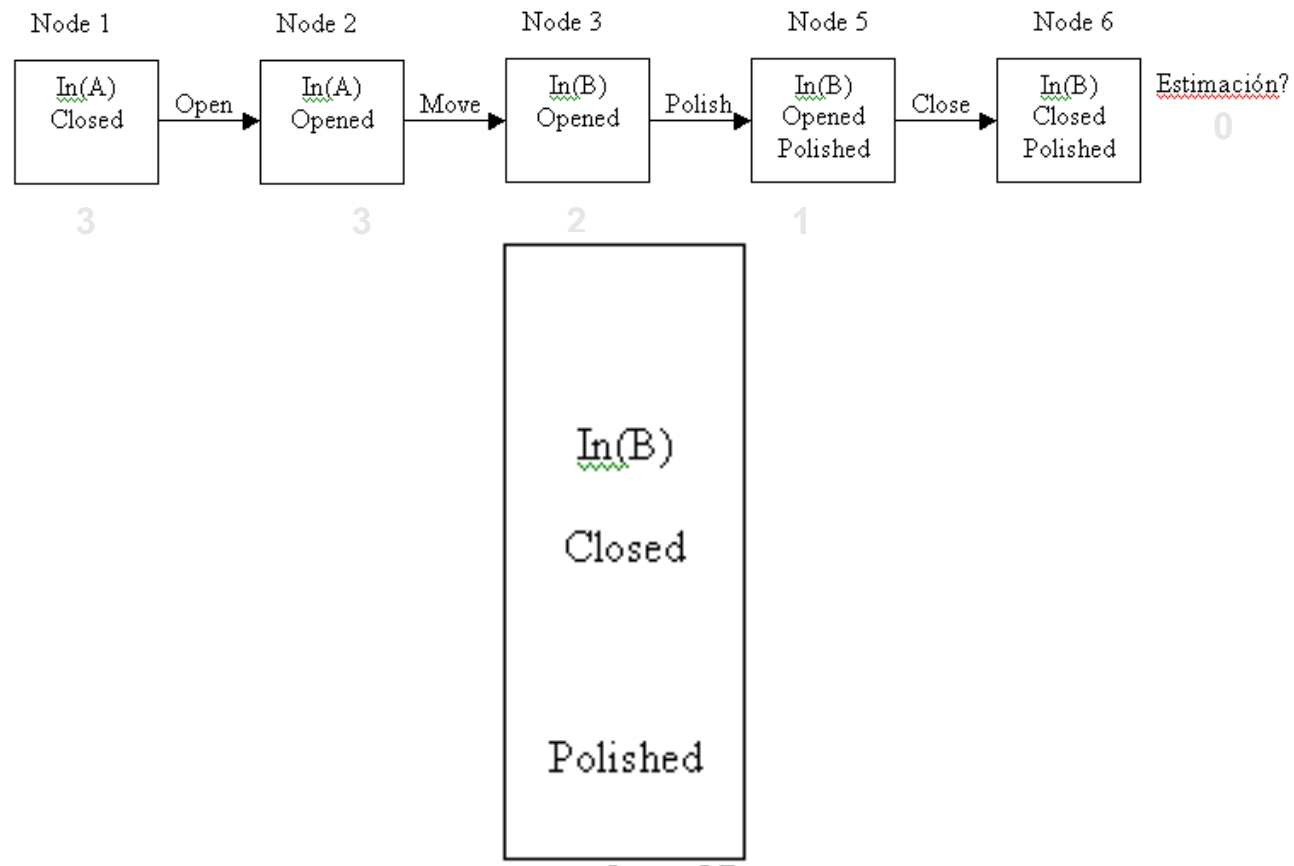
It does not improve the previous result, try Polish



# FF: h



## FF: h



# Outline

- Introduction
- Heuristics
  - Max-cost and additive cost
  - Delete-Relaxation
  - Landmarks
- FF: Fast-Forward
- **Conclusions**

# Conclusions

- HSP transforms planning problems into heuristic search problems
- FF: representative
  - h: relaxed GP (ignore delete list) and use the length as the heuristic
  - Is the heuristic admissible?
  - Performs forward state-space search ( $A^*$ /EHC)