Planning Techniques

Dra. Mª Dolores Rodríguez Moreno

Universidad de Alcalá
ISG

Objectives

Specific Objectives

- Main structures used
- Main techniques

Source

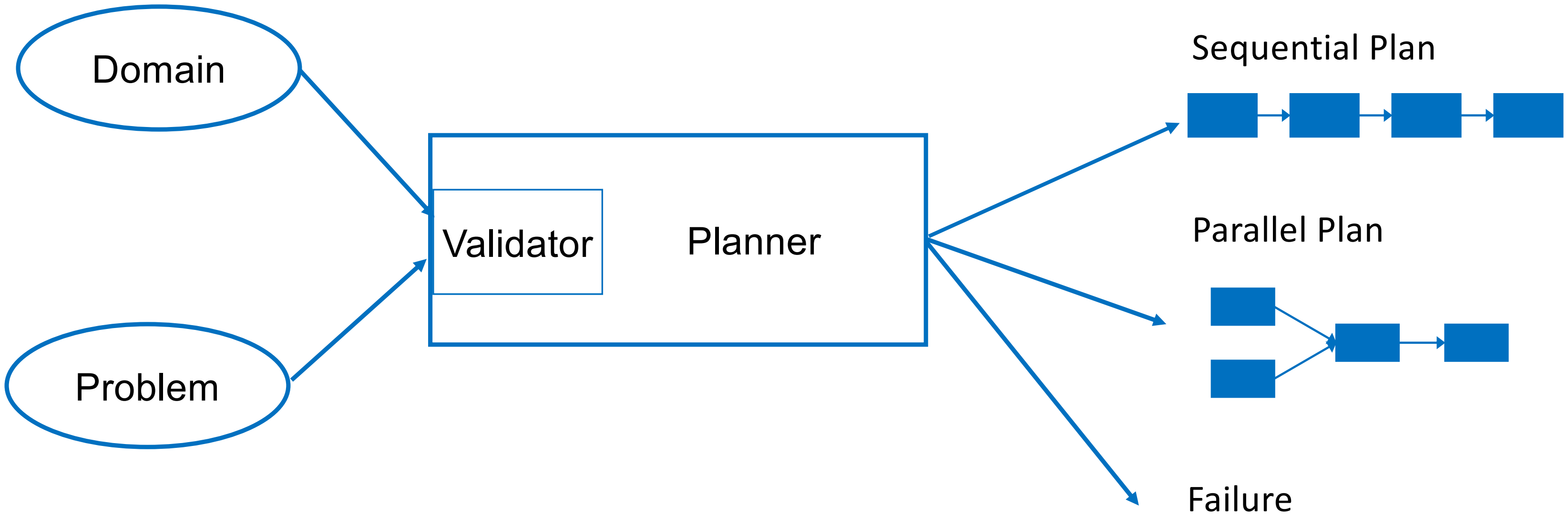
- Stuart Russell & Peter Norvig (2009). Artificial Intelligence: A Modern Approach. (3rd Edition). Ed. Pearsons
- Ghallab, Nau &Traverso (2004). Automated Planning: Theory & Practice. The Morgan Kaufmann Series in Artificial Intelligence
- Dana Nau's slides for Automated Planning. Licensed under License https://creativecommons.org/licenses/by-nc-sa/2.0/

Outline

- **Introduction**
- Classification of algorithms
- Planning techniques
- Conclusions

Introduction

- Nearly all planning procedures are search procedures



Outline

- Introduction
- **Classification of algorithms**
- Planning techniques
- Conclusions

Classification of algorithms

- Criteria for classifying algorithms:
  - How the search is performed (state/plan, progression/regression)
  - Goals ordering: (no)lineal
  - How plans are built: generative (no library plans) or case-based
  - Dealing with uncertainty: contingent and probabilistic
  - Using specific knowledge: domain (in)dependent

Classification of algorithms

- Criteria for classifying algorithms:
  - **How the search is performed (state/plan, progression/regression)**
  - Goals ordering: (no)lineal
  - How plans are built: generative (no library plans) or case-based
  - Dealing with uncertainty: contingent and probabilistic
  - Using specific knowledge: domain (in)dependent

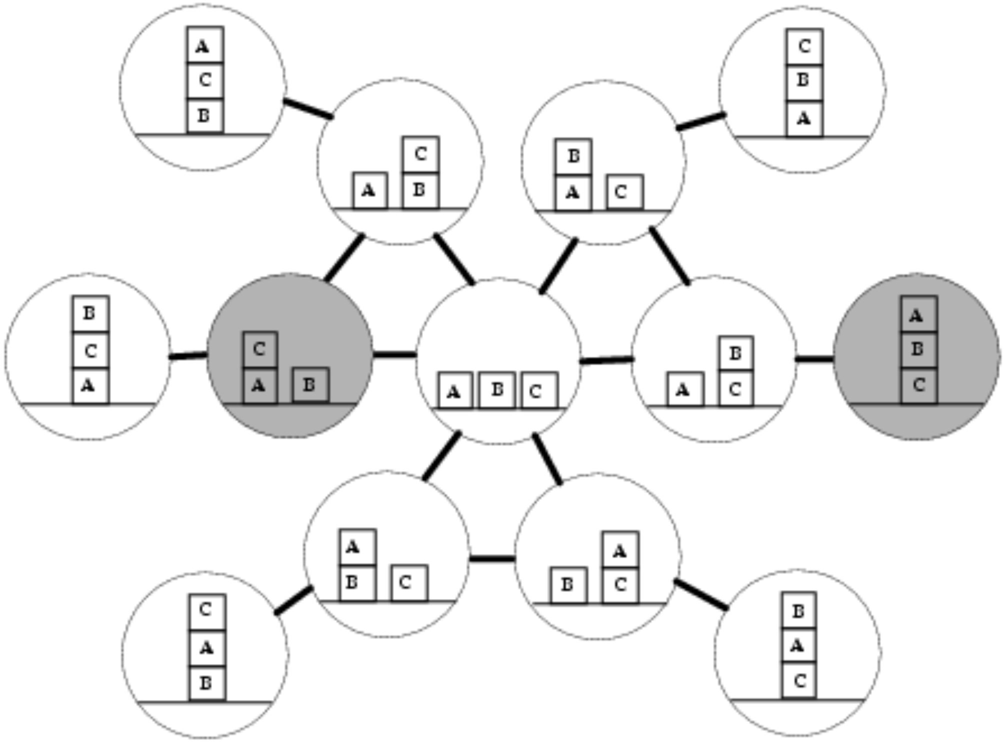
Classification of algorithms: search

- **State-space search in planning**
  - Each node represents a state of the world
  - A plan is a path through the space
- Plan-space search in planning
  - Each node is a set of partially-instantiated operators, plus some constraints
  - Impose more and more constraints, until we get a plan

State Space Search (SSS) (I)

- The easiest way to build a planner is to convert it in a search problem through the state space (SSS)
  - Each node in the tree/graph represents a state of the world
  - Each arc connects worlds that are achieved by executing an action
- Once the planning problem is converted → we can apply any algorithm studied

State Space Search (SSS) (II)



State Space Search (SSS) (III)

- You can incrementally generate the set of reachable states from the initial state by a sequence of actions: *progression*
- The states that are achieved in these sequences can be calculated
  - Checking if the goal has been achieved
  - Checking if the preconditions are satisfied
- Instead of looking forward from the initial state, you can search backward from the goals: *regression*

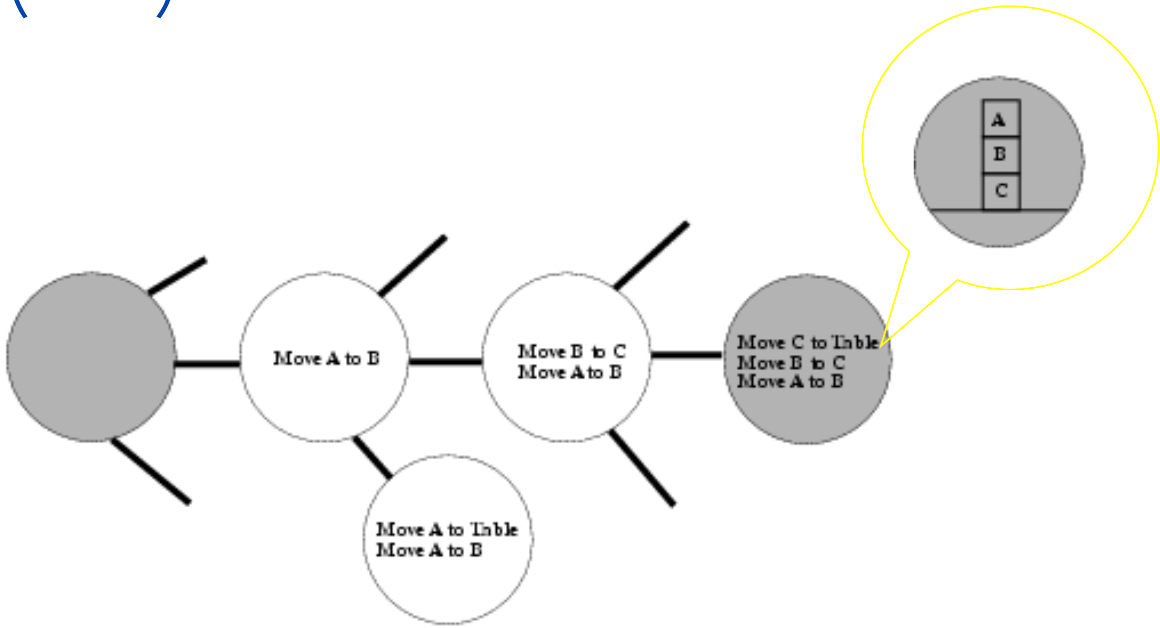
Classification of algorithms: search

- State-space search in planning
  - Each node represents a state of the world
  - A plan is a path through the space
- **Plan-space search in planning**
  - Each node is a set of partially-instantiated operators, plus some constraints
  - Impose more and more constraints, until we get a plan

Plan Space Search (PSS)

- In 1974, the planner NOAH (Sacerdoti) was built, the search was conducted through the space of plans (PSS) :
  - Each node in the tree/graph represents a partial plan
  - Each arc represents refining plan operations (add an action to the plan)
  - A set of constraints
- The initial node is a NULL plan and the final node represents the solution plan for the goal
- While SSS has to return the solution path from the initial to the goal states, the goal state in PSS is the solution

Plan Space Search (PSS)



Outline

- Introduction
- Classification of algorithms
- **Planning techniques**
  - Total Order Planners (TO)
  - Partial Order Planners (POP)
  - Hierarchical Task Network (HTN)
  - Graph-based Planners (GP)
  - SAT-based planners
  - Heuristic Search Planners
- Conclusions

Planning techniques

- TO: the solution is a totally ordered sequence of actions (States / Plans)
- PO: search at the space plans. Implements a "least Commitment approach": only the essential decisions orders are saved
- HTN: network of tasks and constraints
- **Graph-based**: the search structure is a Planning Graph
- SAT: takes as input a problem, guess the length of the plan and generates propositional clauses
- **Heuristic Search Planners**: transform planning problems in heuristic problems

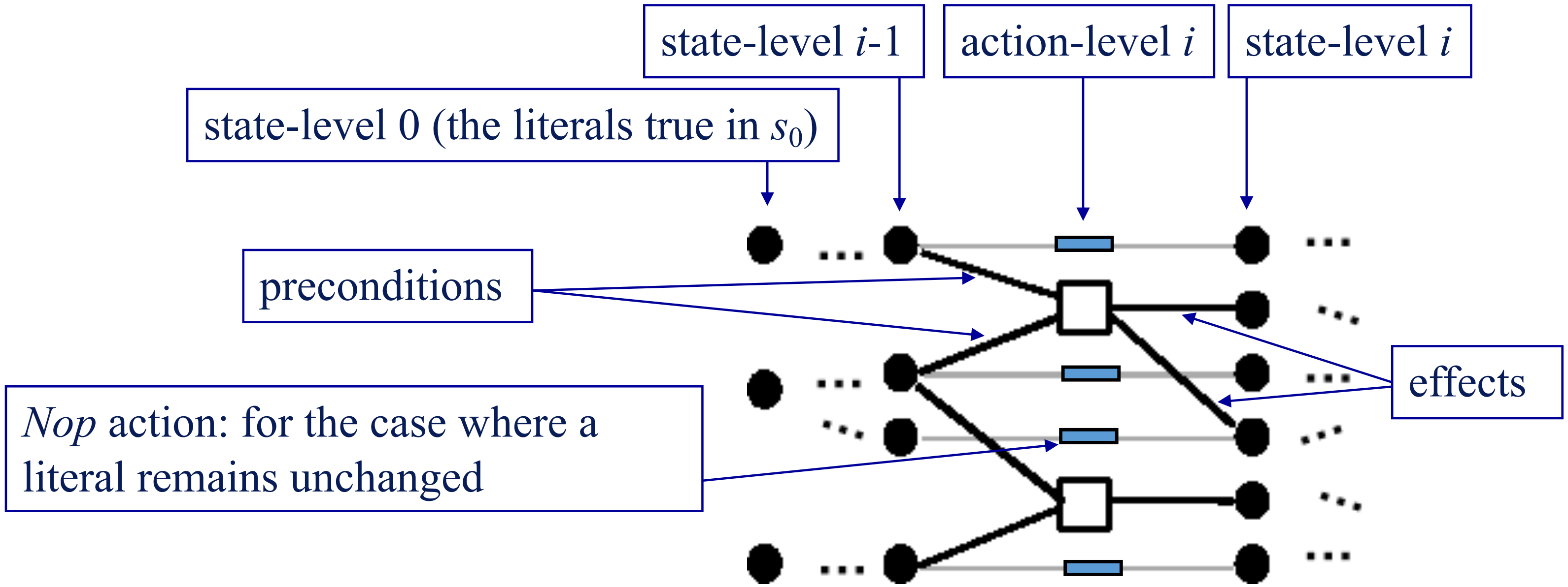
Planning techniques

- Total Order Planners (TO)
- Partial Order Planners (POP)
- Hierarchical Temporal Planners (HTN)
- **Graph-based Planners (GP)**
- SAT-based planners
- Heuristic Search Planners

Graph-based Planners (I)

- The search structure is based on a Planning Graph
- The graph is directed and layered:
  - 2 types of nodes:
    - Proposition nodes: even levels (initial state→ 0)
    - Action nodes: odd levels
  - 3 types of arcs: represent relationships between actions and propositions:
    - Added
    - Deleted
    - Nop

Graph-based Planners (II)



Graph-based Planners (III)

- GP algorithm works in two alternating phases
  - Expands (add layers) the planning graph until the last proposition layer satisfies the goal condition
  - Try to extract a valid plan (backtracking) from the planning graph

- If unsuccessful continues with the former phase, the planning graph is expanded again

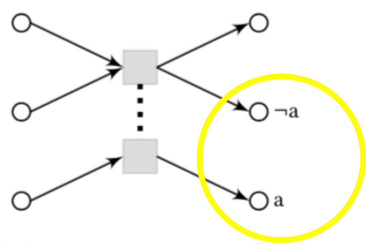
Graph-based Planners (IV)

- It is necessary to develop a reachability analysis to reduce the set of actions that are not supported in each layer
- Compatibility inferring mutual exclusion relations between incompatible actions is performed (mutex)
  - Have opposite effects
  - Incompatible preconditions
  - The effect of one action is the opposite of another
- Mutex between incompatible propositions: negated literals or all actions that can achieve them are mutex in the previous step

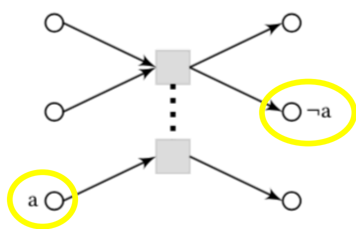
Graph-based Planners (V)

- Two actions at the same action-level are mutex if
  - *Inconsistent effects:* an effect of one negates an effect of the other
  - *Interference:* one deletes a precondition of the other
  - *Competing needs:* **they have mutually exclusive preconditions**
- Otherwise they don't interfere with each other (may appear in solution)
- Two literals at the same state-level are mutex if
  - *Inconsistent support:* one is the negation of the other,
    or all ways of achieving them are pairwise mutex

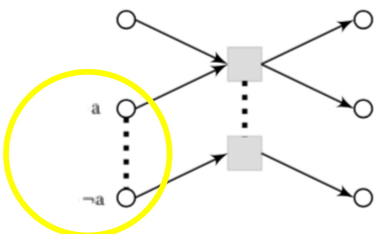
Graph-based Planners (VI)



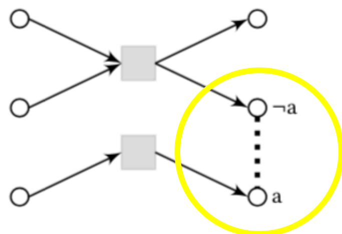
(a) Inconsistent effects



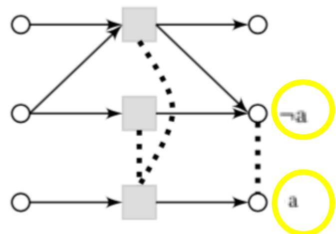
(b) Interference



(c) Competing needs

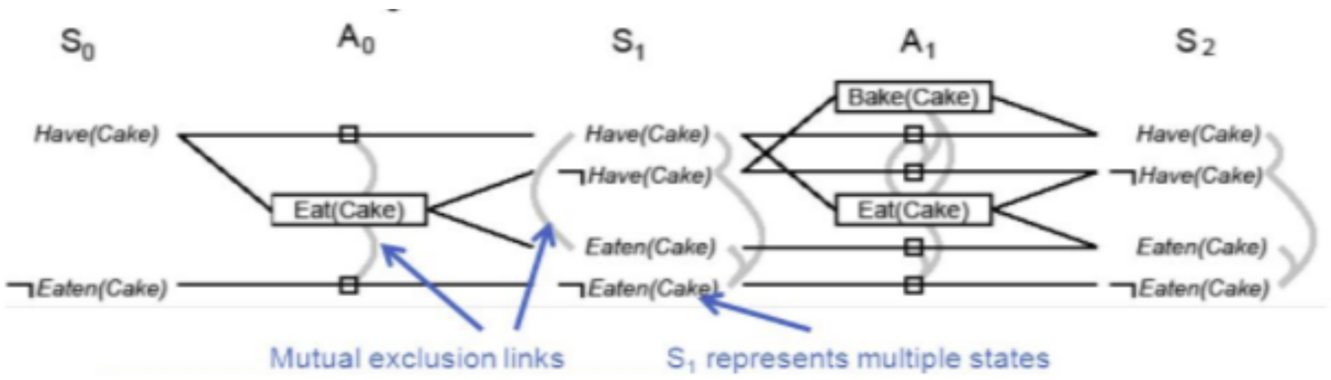


(d) Contradiction



(e) Inconsistent support

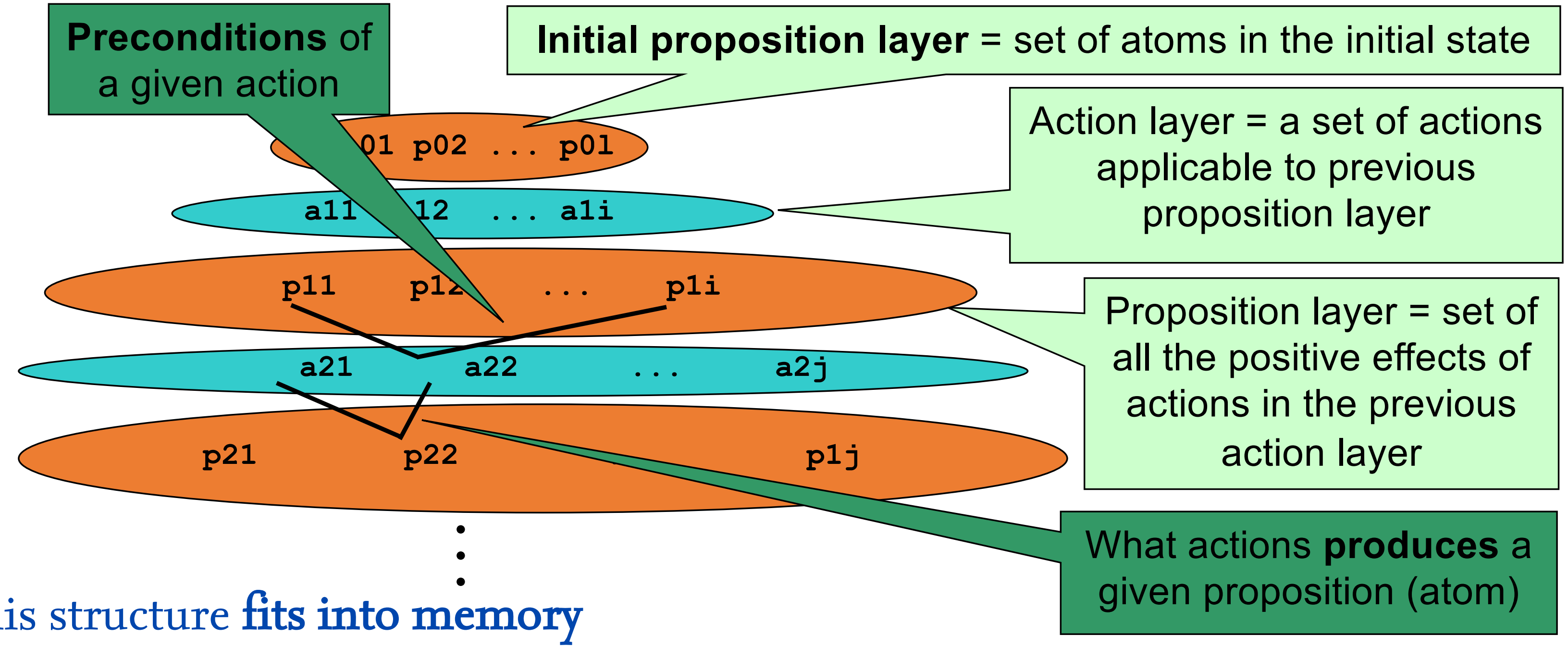
Example



(:action Eat
    :parameters (?cake)
    :precondition (and (have ?cake))
    :effect (and (eaten ?cake) (not (have ?cake)))
 (:action Bake
    :parameters (?cake)
    :precondition (not (have ?cake))
    :effect (and (have ?cake))))

Initial State
    Have (cake) & (not (Eaten(cake))
Goal
    Eaten (cake)

State reachability



- this structure **fits into memory**
  - every **proposition** knows its origin in the previous action layer
  - every **action** knows its precondition in the previous proposition layer

Planning techniques

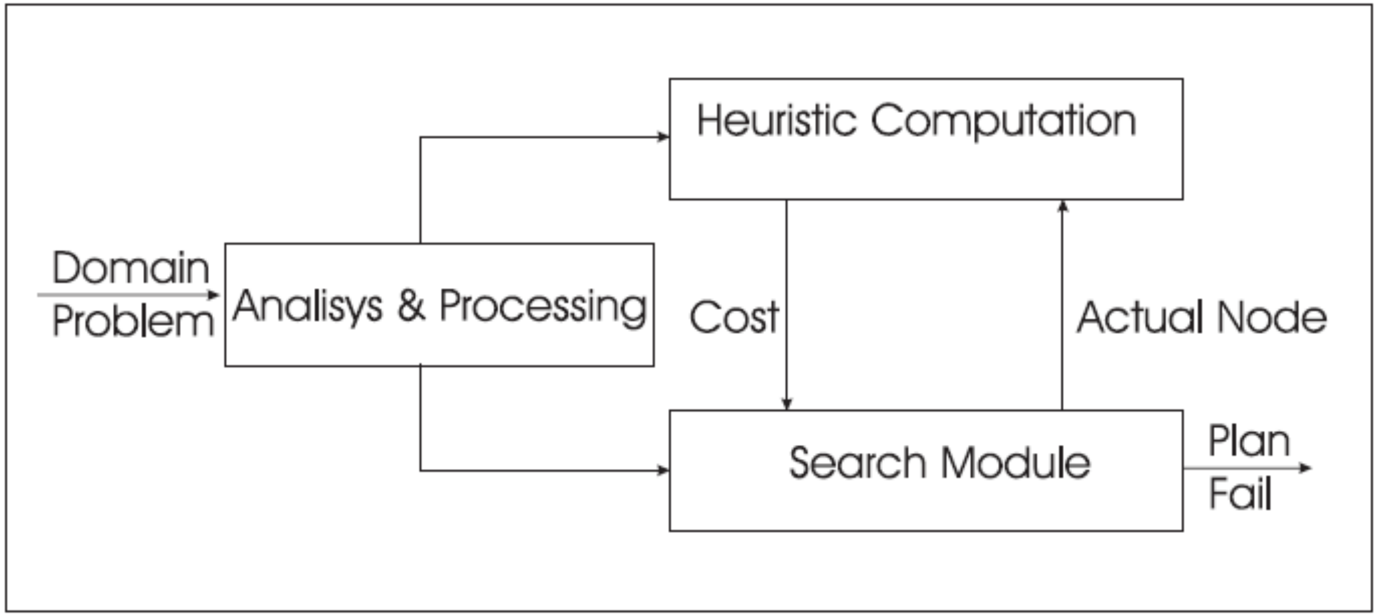
- Total Order Planners (TO)
- Partial Order Planners (POP)
- Hierarchical Temporal Planners (HTN)
- Graph-based Planners (GP)
- SAT-based planners
- **Heuristic Search Planners**

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^{add})
  - FF: delete-relaxation

FF (I)

- General architecture

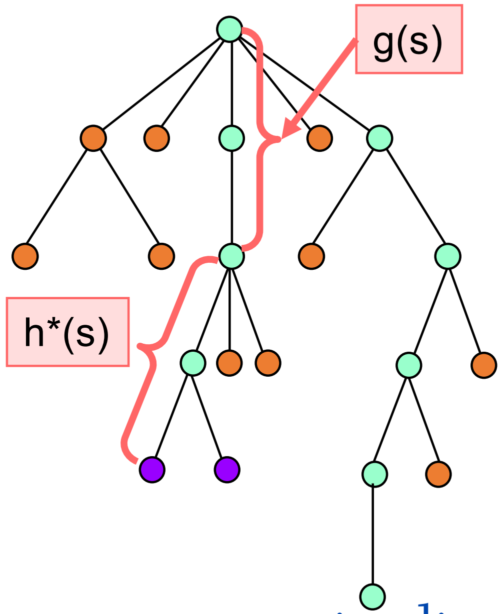


FF (II)

- **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: Heuristic

- Build GP until all goals are reached
  - First fact layer = S0 and first action layer contains all actions that are applicable in S0
  - The union of all **add** effects of these actions with the facts that are already there forms the second fact layer
  - To this layer, again all actions are applied, and so on, until a fact layer is reached that contains all goals
- When goals are reached, extract a relaxed plan:
  - Start at the top graph layer m, working on all goals
  - At each layer i, if a goal is present in layer i – 1, then insert it into the goals to be achieved at i– 1
  - Else, select action in layer i– 1 that adds the goal, & insert the action's preconditions into the goals at i– 1
  - Once all goals at i are worked on, continue with the goals at i– 1
  - Stop when the first layer is reached
  - The process results in a relaxed plan ⟨O0, ..., Om–1⟩, where each Oi is the set of actions selected at time step i, with prevalence of NoOp over normal actions
  - Estimate solution length by counting the actions in that plan, NoOp counts as **0**

FF: h

Move (x,y) **Pre:** in(x) opened **Add:** in(y) **Del:** in(x) **Open** **Pre:** closed **Add:** opened **Del:** closed	**Close** **Pre:** opened **Add:** closed **Del:** opened **Polish** **Pre:** opened **Add:** polished **Del:**

In(A)
Closed

→

In(B)
Closed
Polished

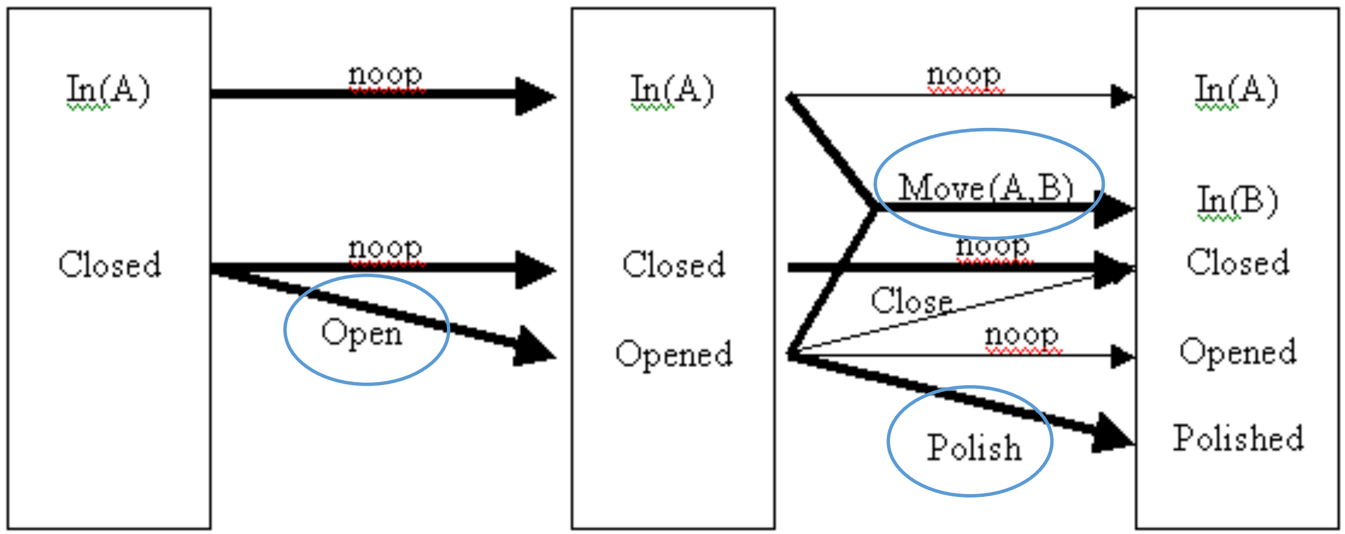
FF: h

Node 1

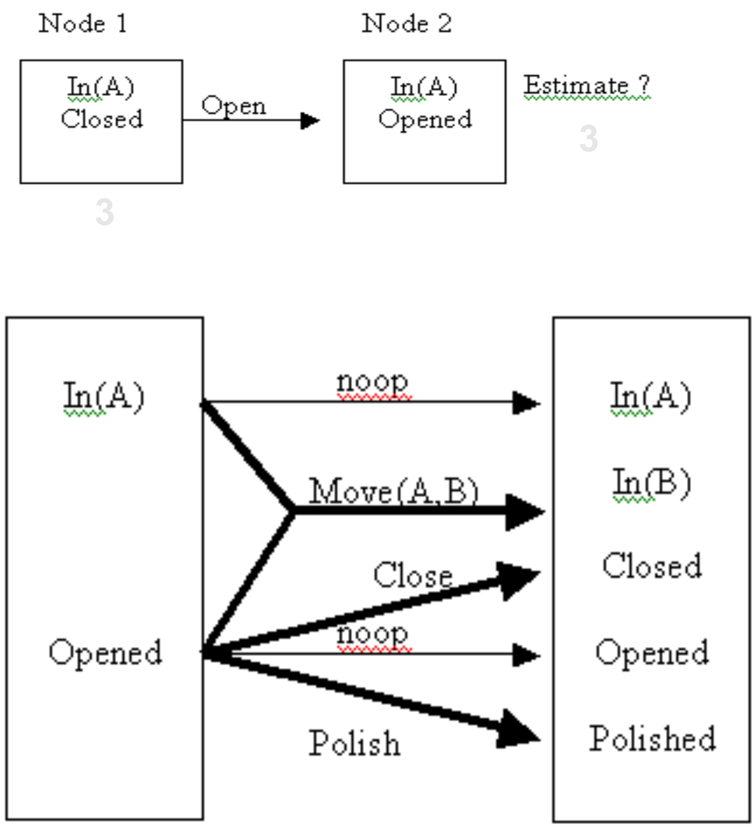
In(A)
Closed

Estimate?

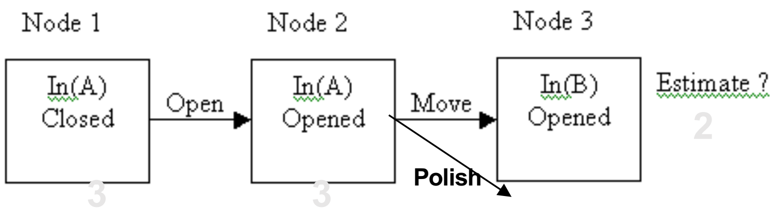
3



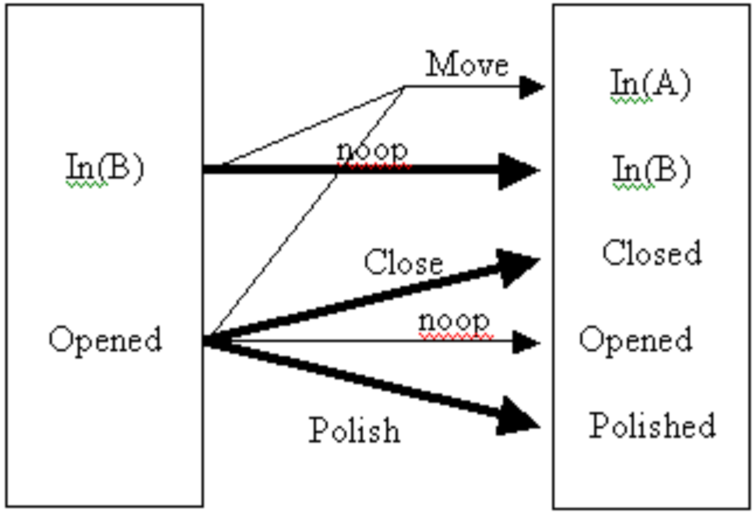
FF: h


Node 1
Node 2
In(A)
Closed
Open
In(A)
Opened
Estimate ?
3
3
In(A)
Opened
noop
Move(A,B)
Close
noop
Polish
In(A)
In(B)
Closed
Opened
Polished

FF: h

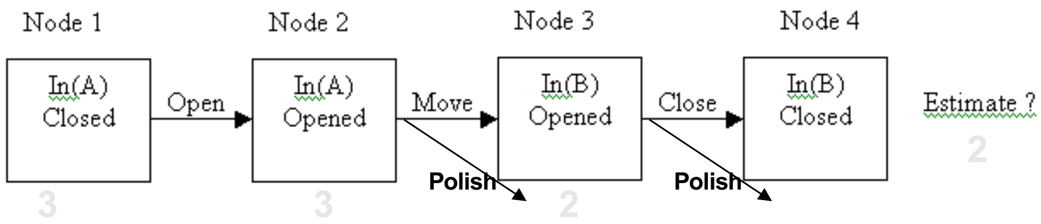


Node 1
Node 2
Node 3
In(A)
Closed
Open
In(A)
Opened
Move
Polish
In(B)
Opened
Estimate ?
2
3
3

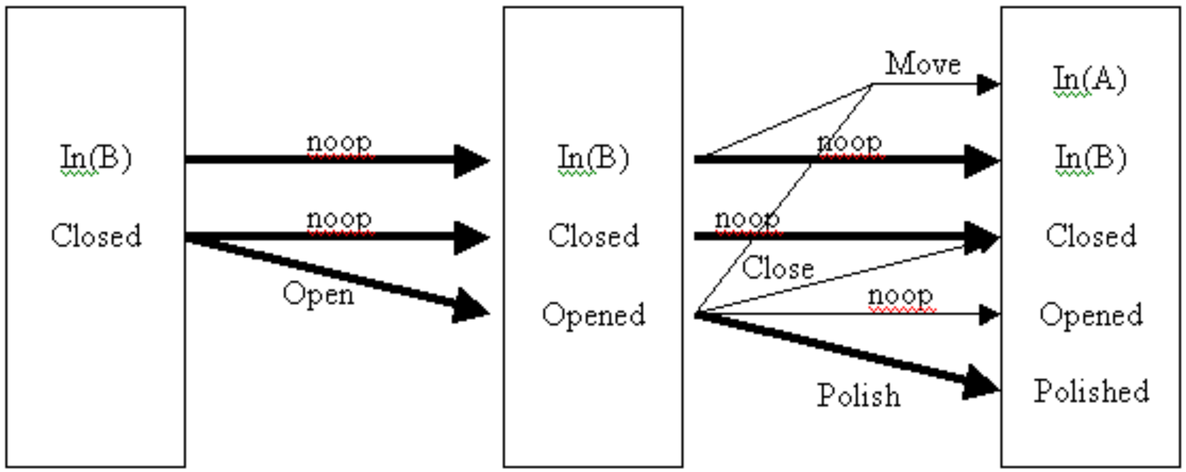


In(B)
Opened
Move
noop
Close
noop
Polish
In(A)
In(B)
Closed
Opened
Polished

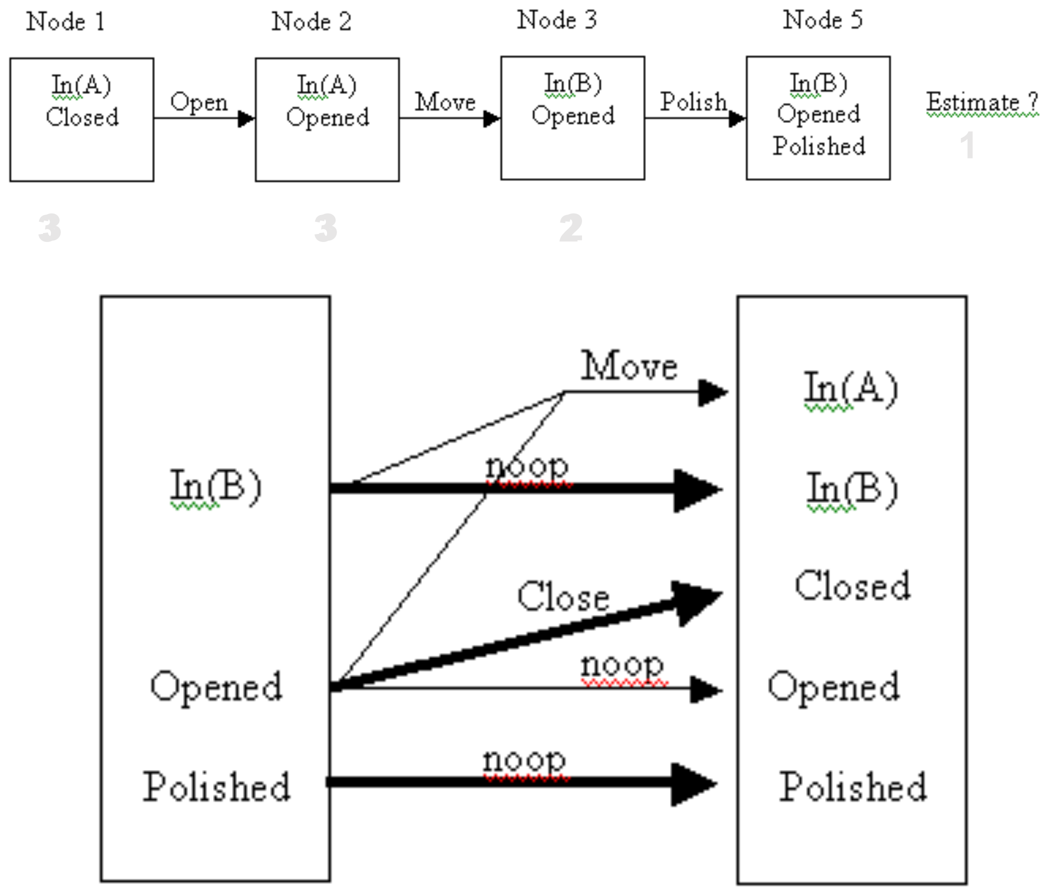
FF: h



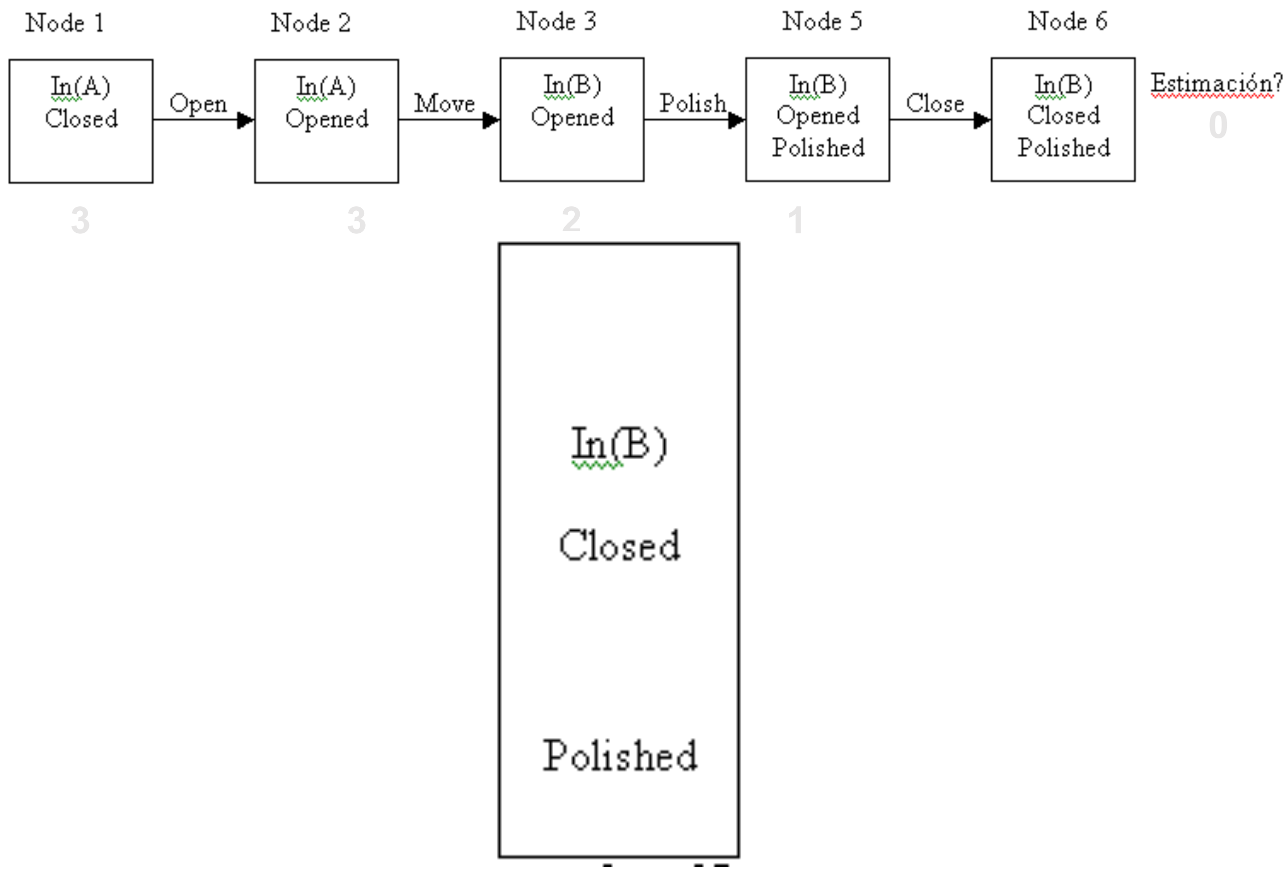
It does not improve the previous result, try Polish



FF: h


Node 1
In(A)
Closed
Open
Node 2
In(A)
Opened
Move
Node 3
In(B)
Opened
Polish
Node 5
In(B)
Opened
Polished
Estimate ?
1
3
3
2
In(B)
Opened
Polished
Move
noop
Close
noop
noop
In(A)
In(B)
Closed
Opened
Polished

FF: h


Node 1
Node 2
Node 3
Node 5
Node 6
In(A)
Closed
Open
In(A)
Opened
Move
In(B)
Opened
Polish
In(B)
Opened
Polished
Close
In(B)
Closed
Polished
Estimación?
0
3
3
2
1
In(B)
Closed
Polished

Outline

- Introduction
- Classification of algorithms
- Planning techniques
- **Conclusions**

Conclusions

- Planning is an interesting area because it combines logic and search
- We have studied the main planners and the search algorithms that use
- There are actually no better strategies than others
- Competition between approaches and the intersection and combination of techniques has resulted in gains in efficiencies in syst. Planning
- HSP: 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)