Universidad de Alcalá

Planning Techniques

ISG

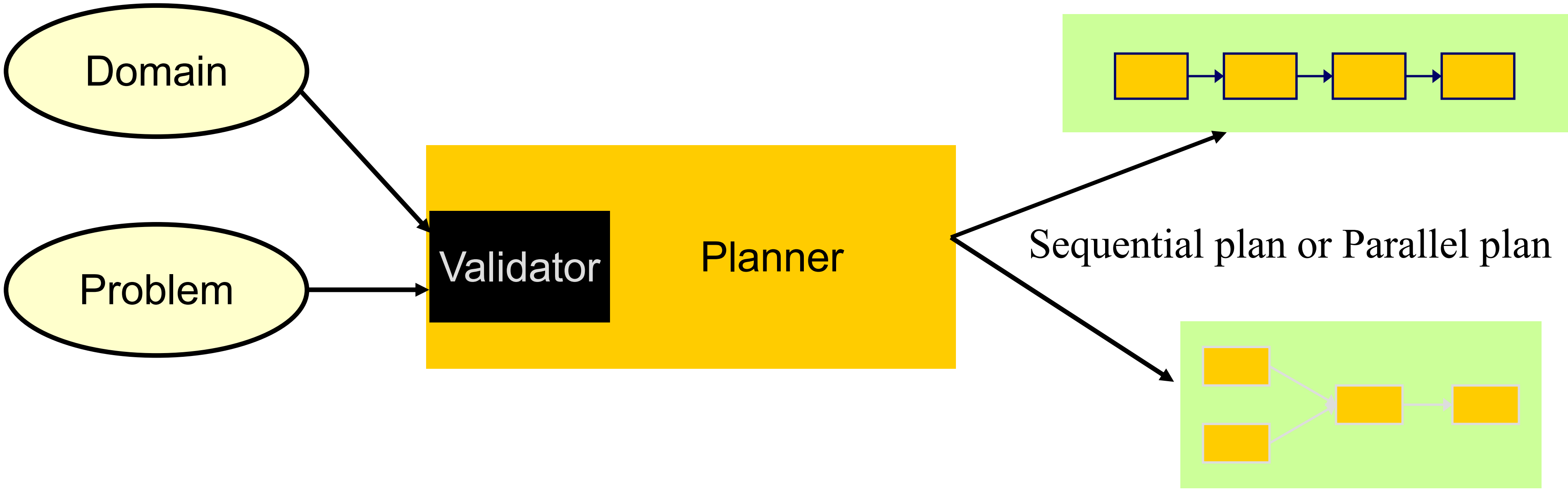
Intelligent Systems Group

Outline

- **Introduction**
- Basic search algorithms
- Classification of planning algorithms
- Planning techniques
- Conclusions

Introduction

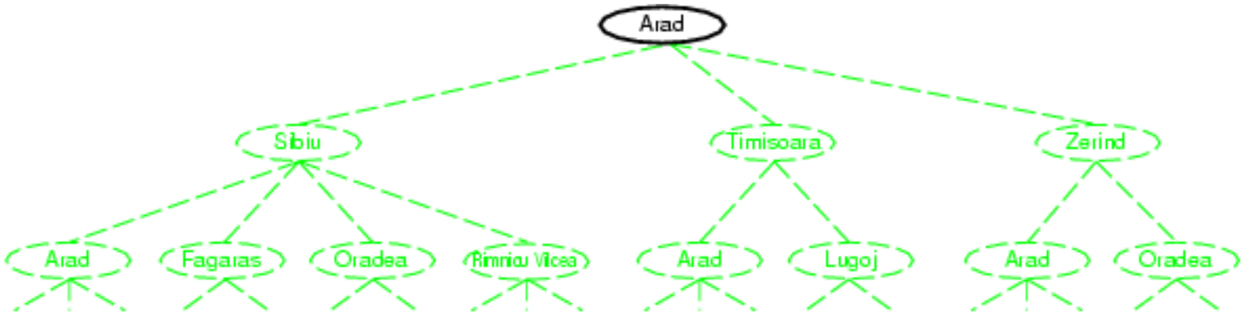
- Inputs and outputs of a planner

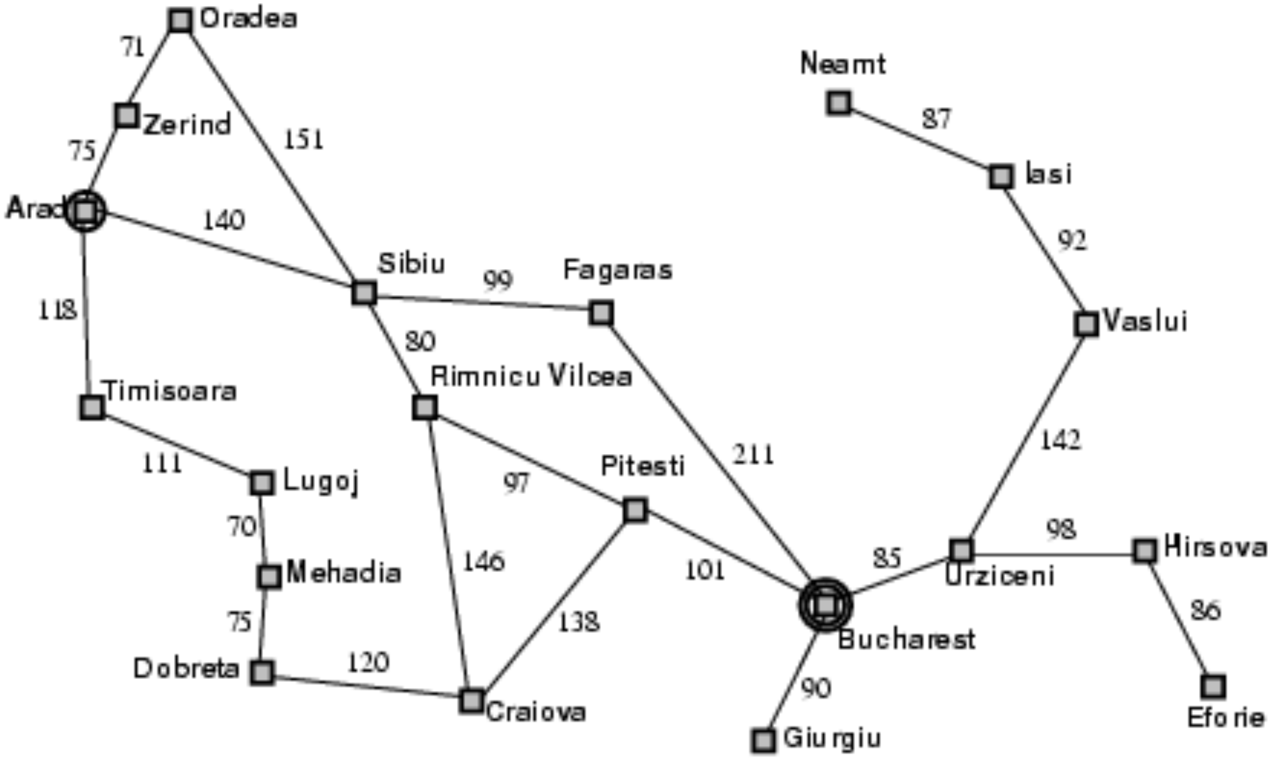


Introduction

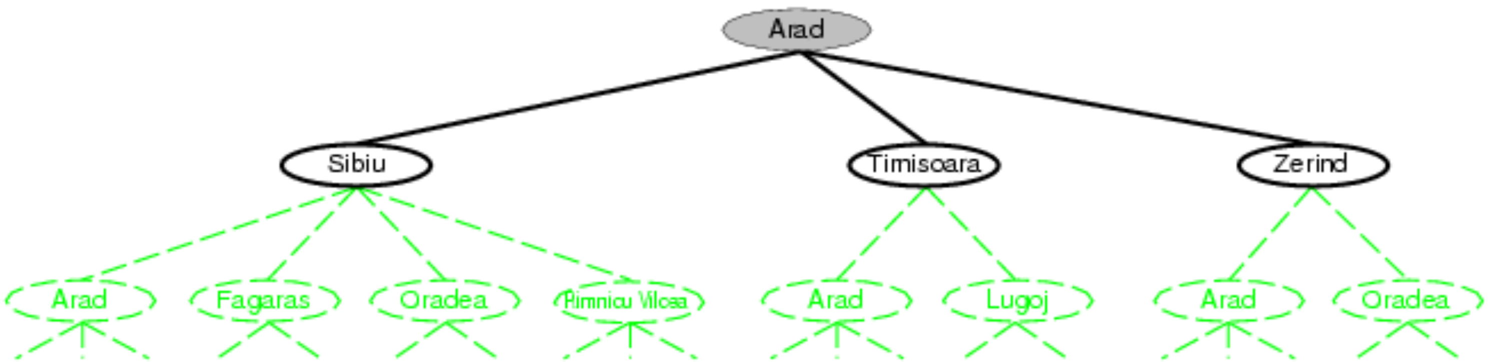
- The easiest way to build a planner is to convert it in a search problem
- What is a search problem?
  - A state space
  - A successor function
  - A start state
  - A goal test
- Representation:
  - A graph/tree
  - Node-based data structure

Introduction

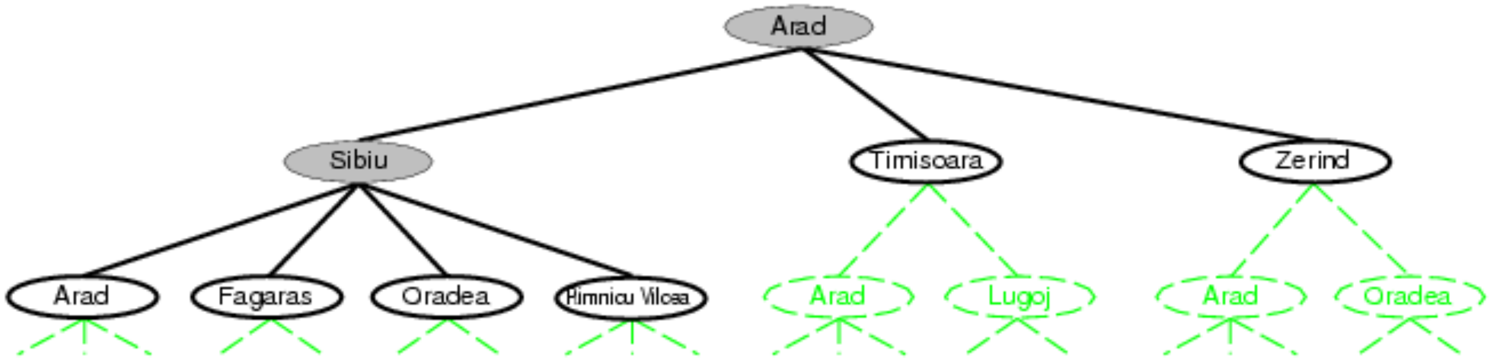




Introduction

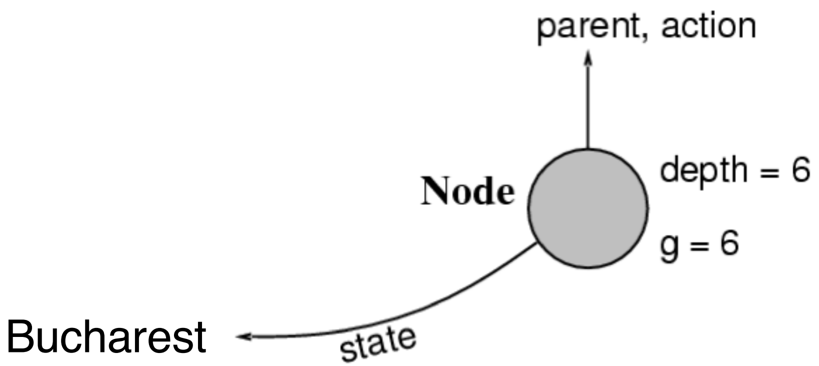


Introduction



Introduction

- A state is a (representation of) a physical configuration
- A node is a data structure constituting part of a search tree includes state, parent node, action, path cost *g(x)*, depth



- A search strategy is defined by picking the order of node expansion

Outline

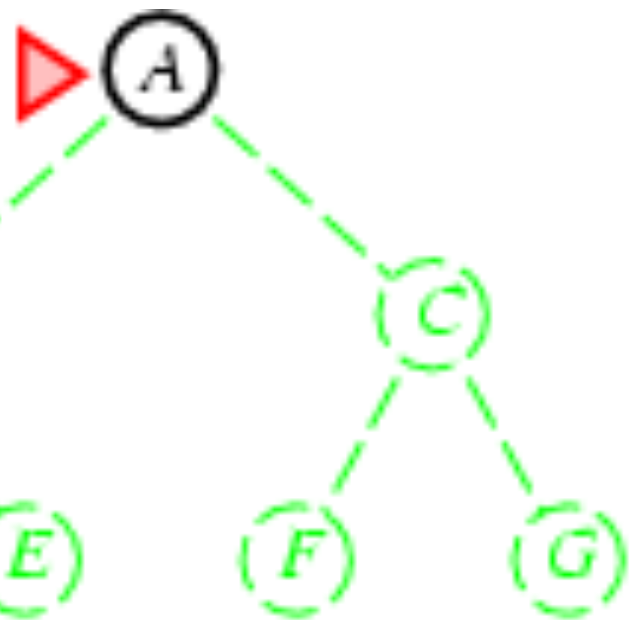
- Introduction
- **Basic search algorithms**
- Classification of planning algorithms
- Planning techniques
- Conclusions

Basic search algorithms

- **Uninformed search**
  - **Breadth-first search/ Búsqueda en anchura**
  - Depth-first search/ Búsqueda en profundidad
- Informed search
  - Greedy best first search search/Búsqueda primero el voraz
  - A*

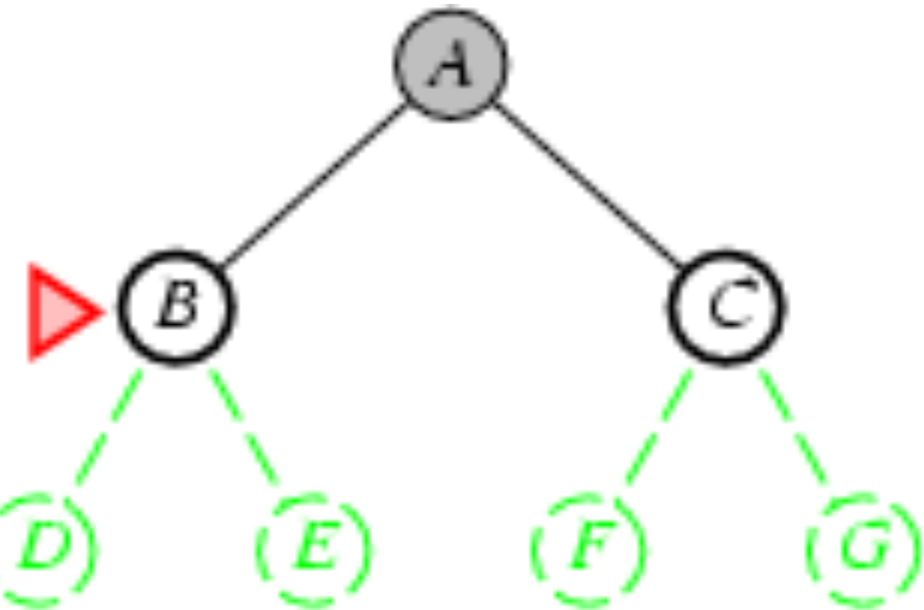
Breadth-first search

- Expand shallowest unexpanded node
- Implementation:
  - *fringe* is a FIFO queue, i.e., new successors go at end



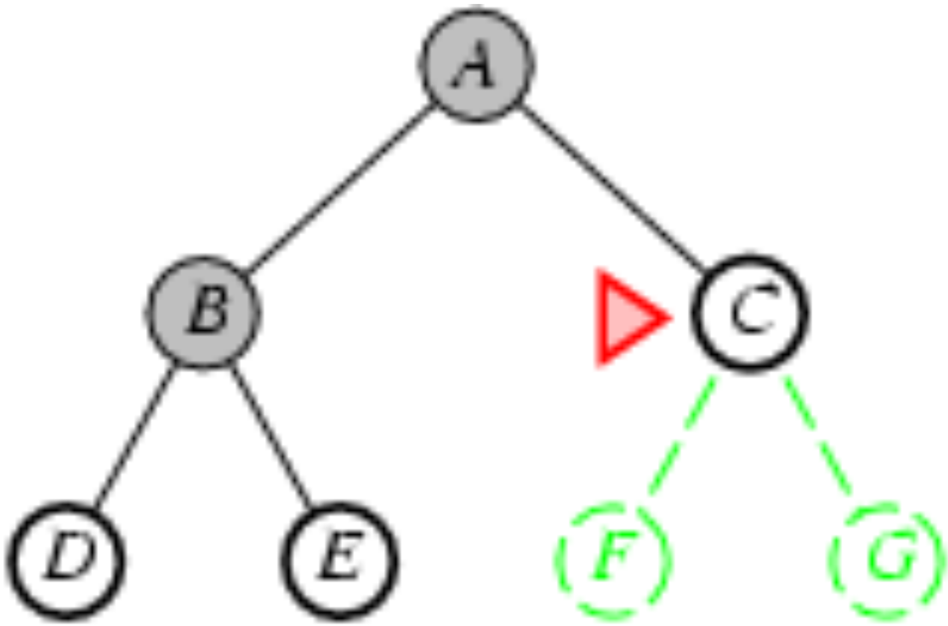
Breadth-first search

- Expand shallowest unexpanded node
- Implementation:
  - *fringe* is a FIFO queue, i.e., new successors go at end



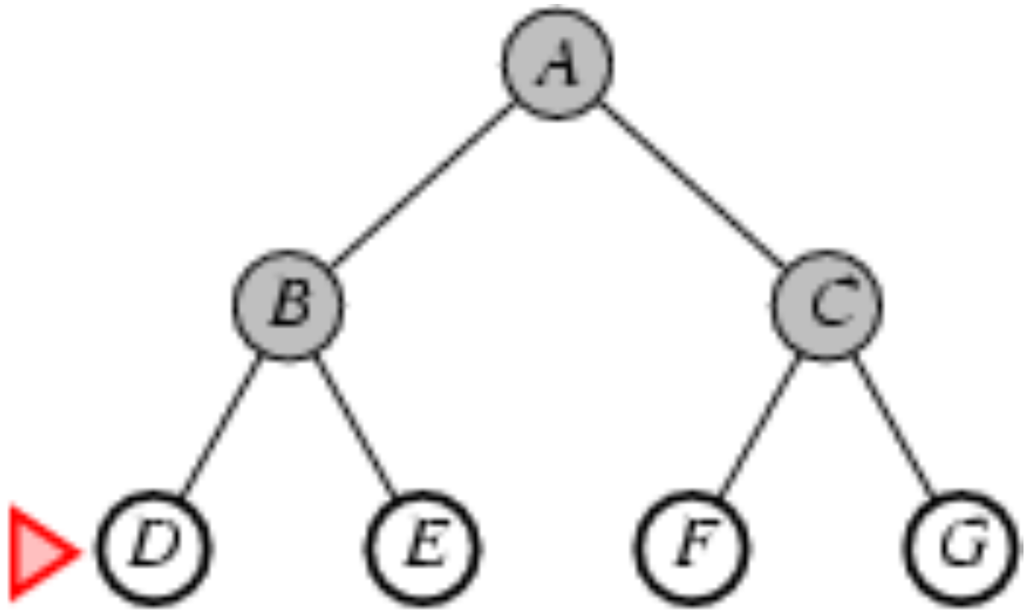
Breadth-first search

- Expand shallowest unexpanded node
- Implementation:
  - *fringe* is a FIFO queue, i.e., new successors go at end



Breadth-first search

- Expand shallowest unexpanded node
- Implementation:
  - *fringe* is a FIFO queue, i.e., new successors go at end



Basic search algorithms

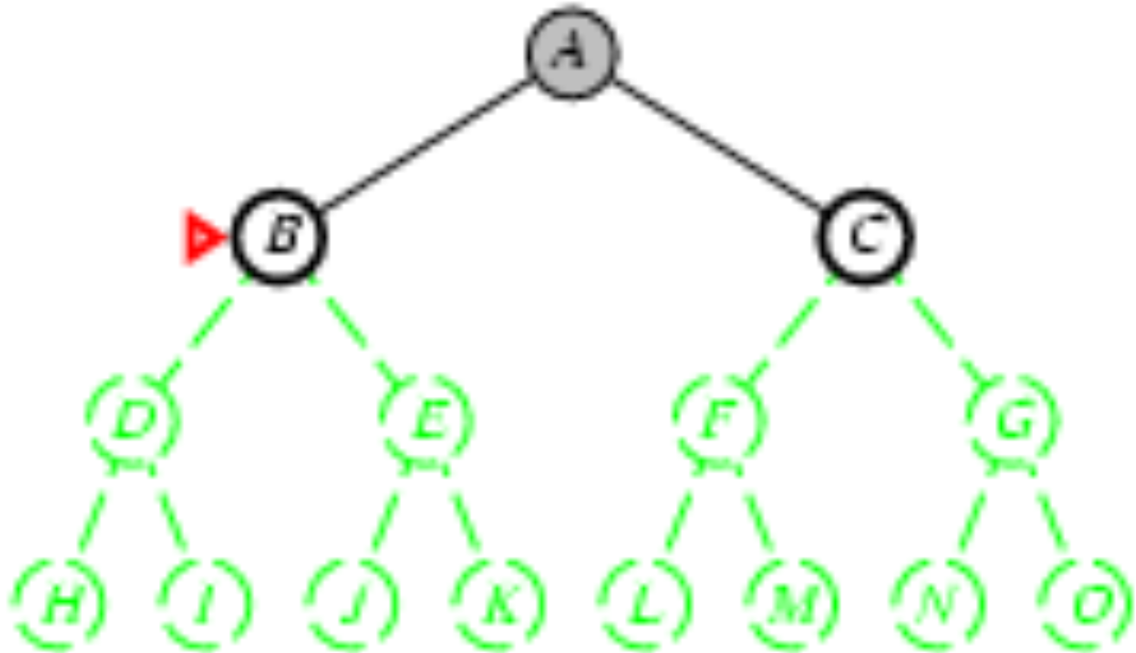
- **Uninformed search**
  - Breadth-first search/ Búsqueda en anchura
  - **Depth-first search/ Búsqueda en profundidad**
- Informed search
  - Greedy best first search search/Búsqueda primero el voraz
  - A*

Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front

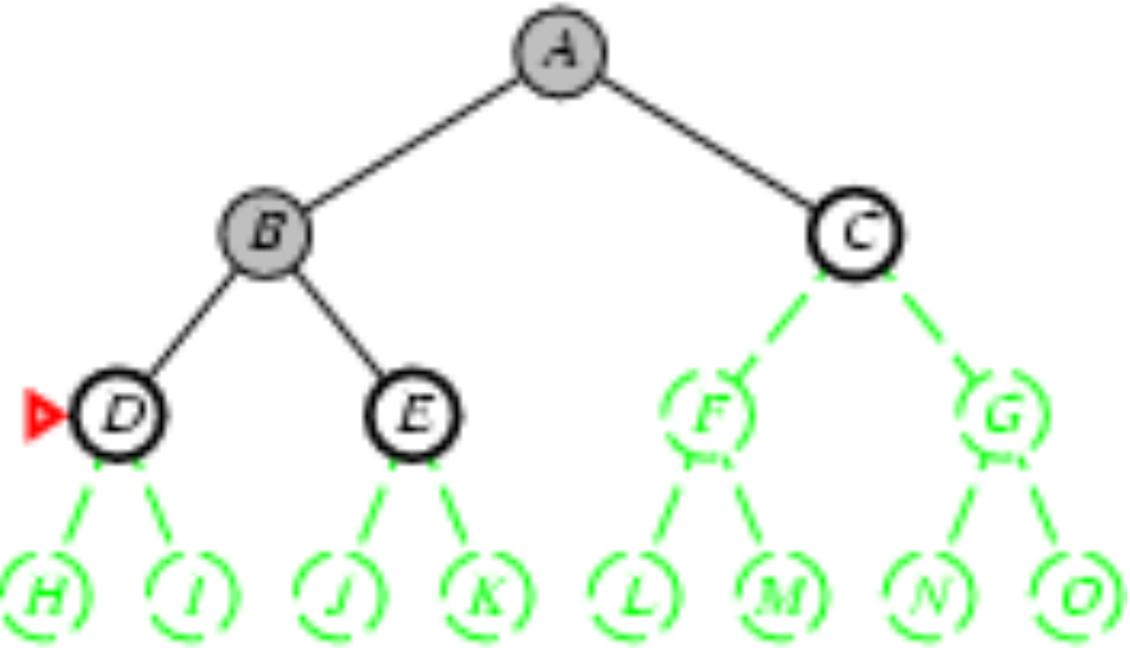
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front



Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front

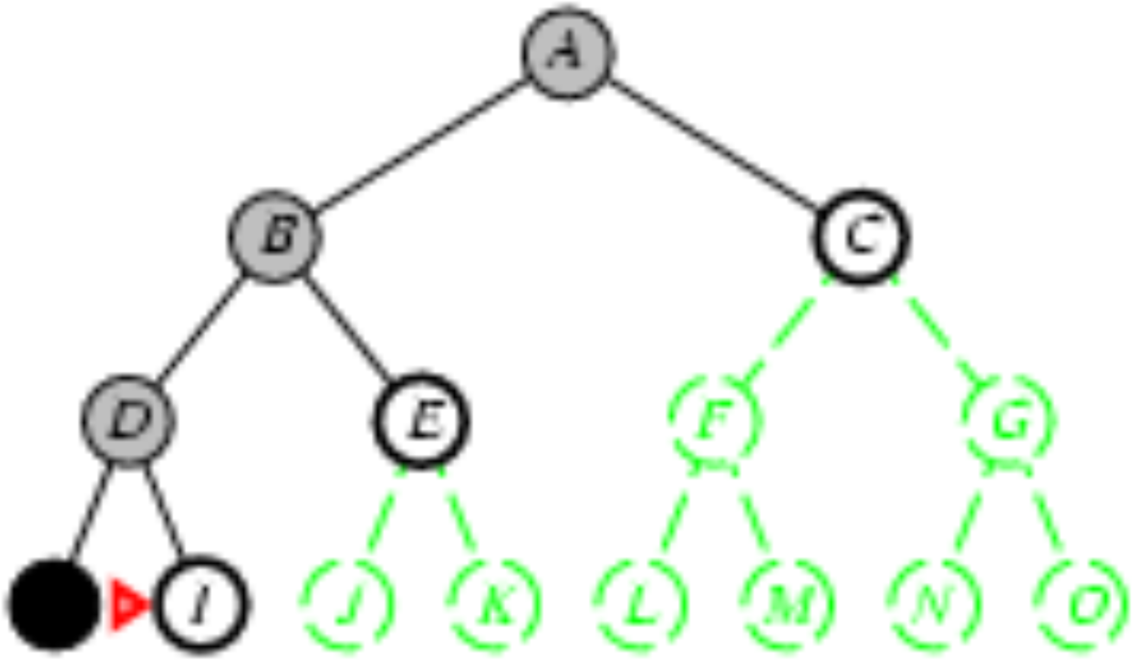


Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front

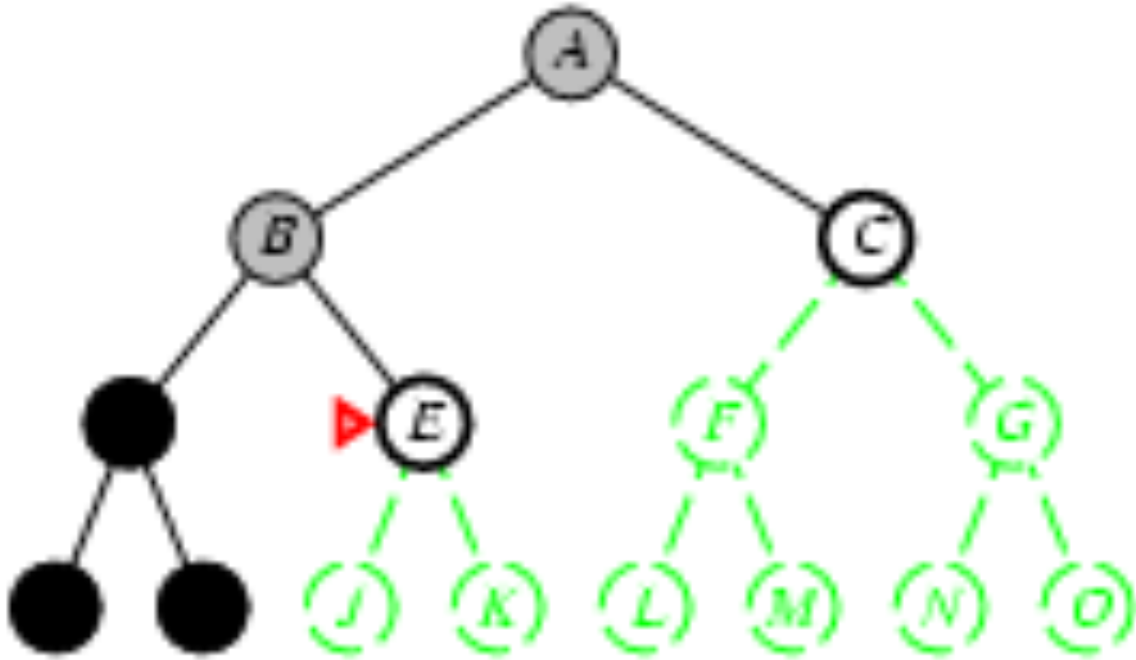
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front



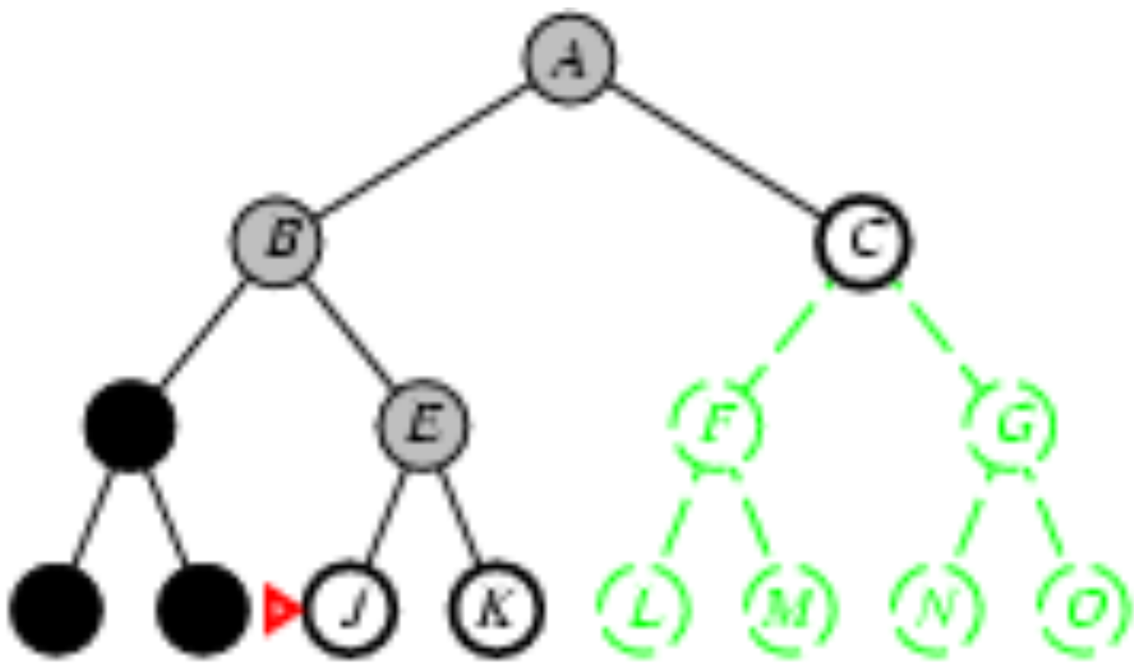
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front



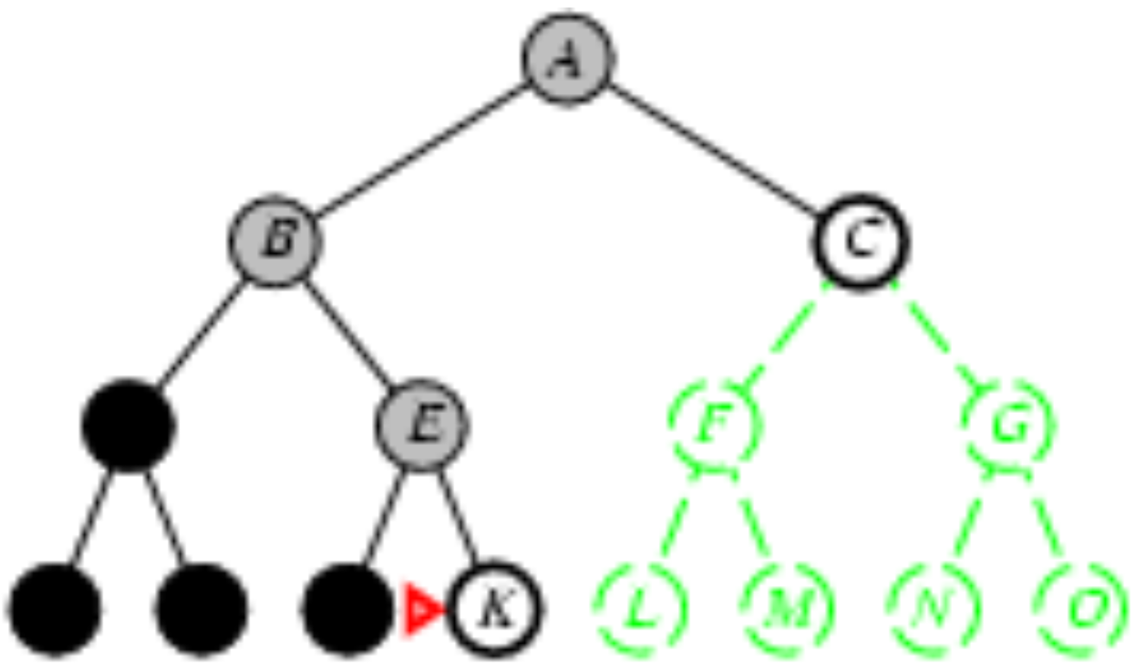
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front



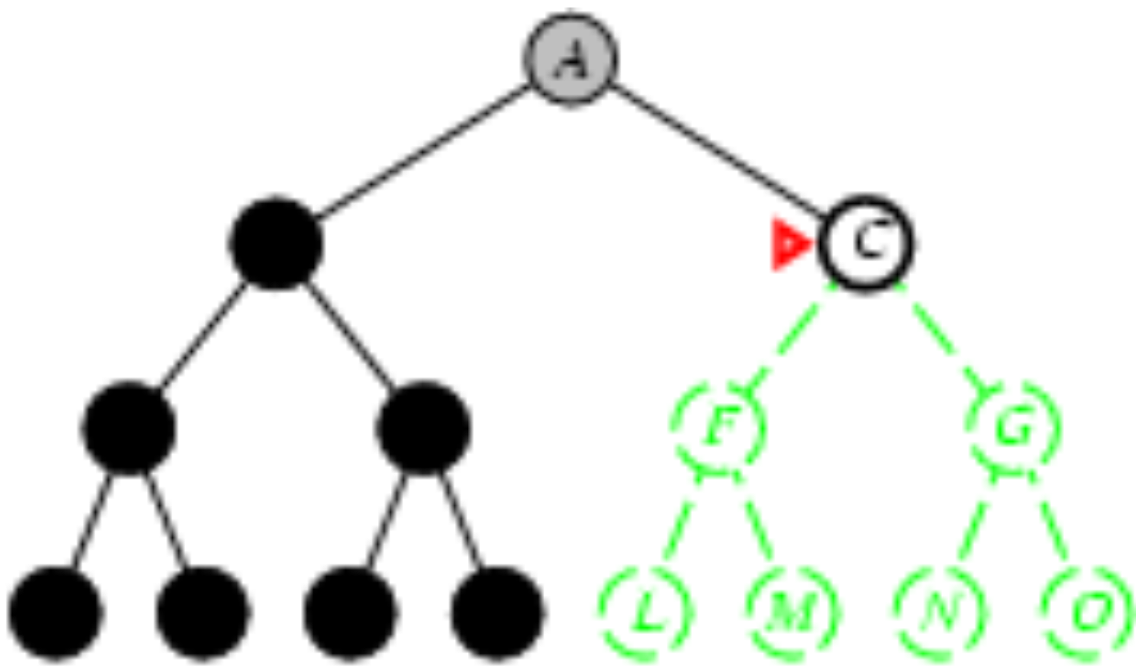
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front



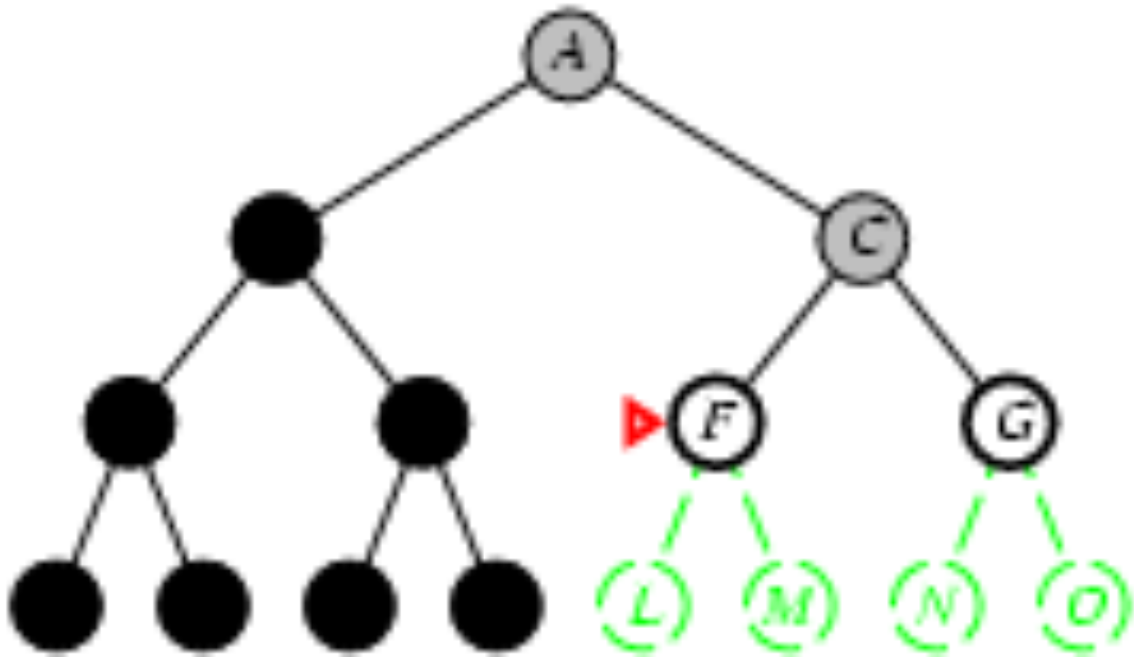
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front



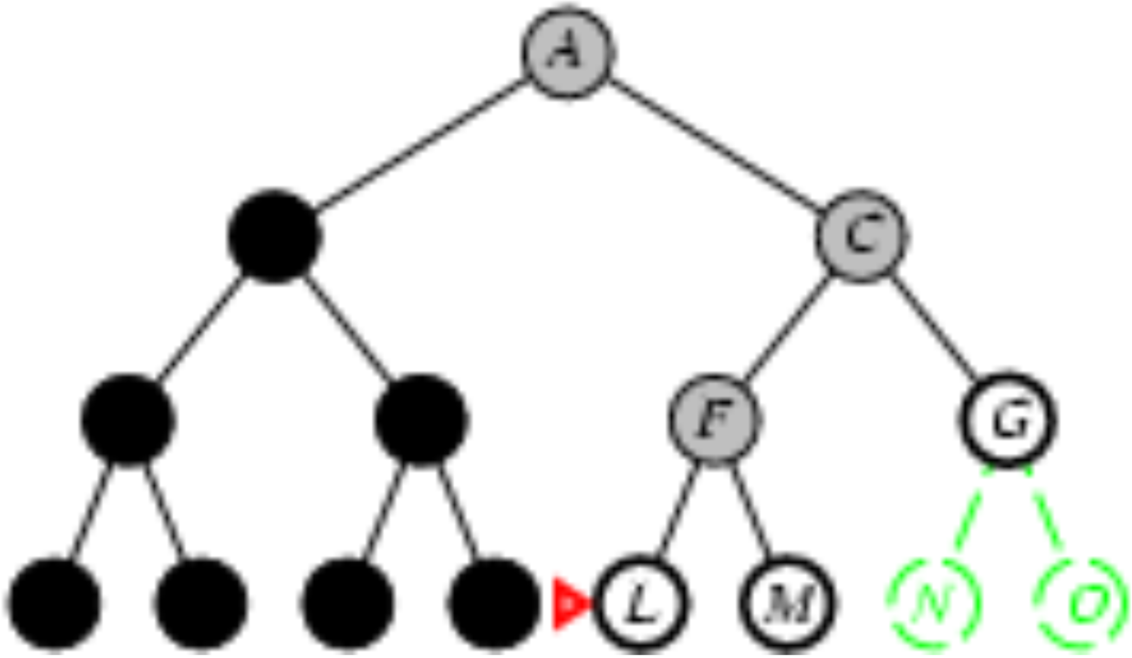
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front



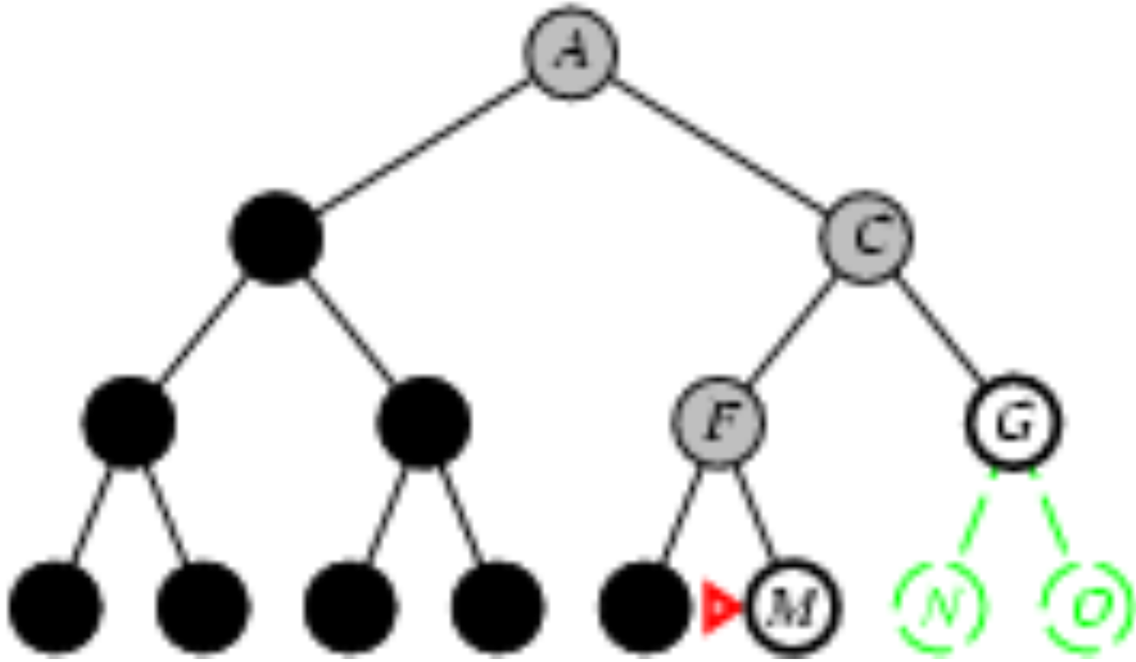
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front



Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front



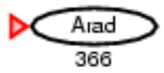
Basic search algorithms

- **Uninformed search**
  - Breadth-first search/ Búsqueda en anchura
  - Depth-first search/ Búsqueda en profundidad
- **Informed search**
  - **Greedy best-first search search**/Búsqueda primero el voraz
  - A*

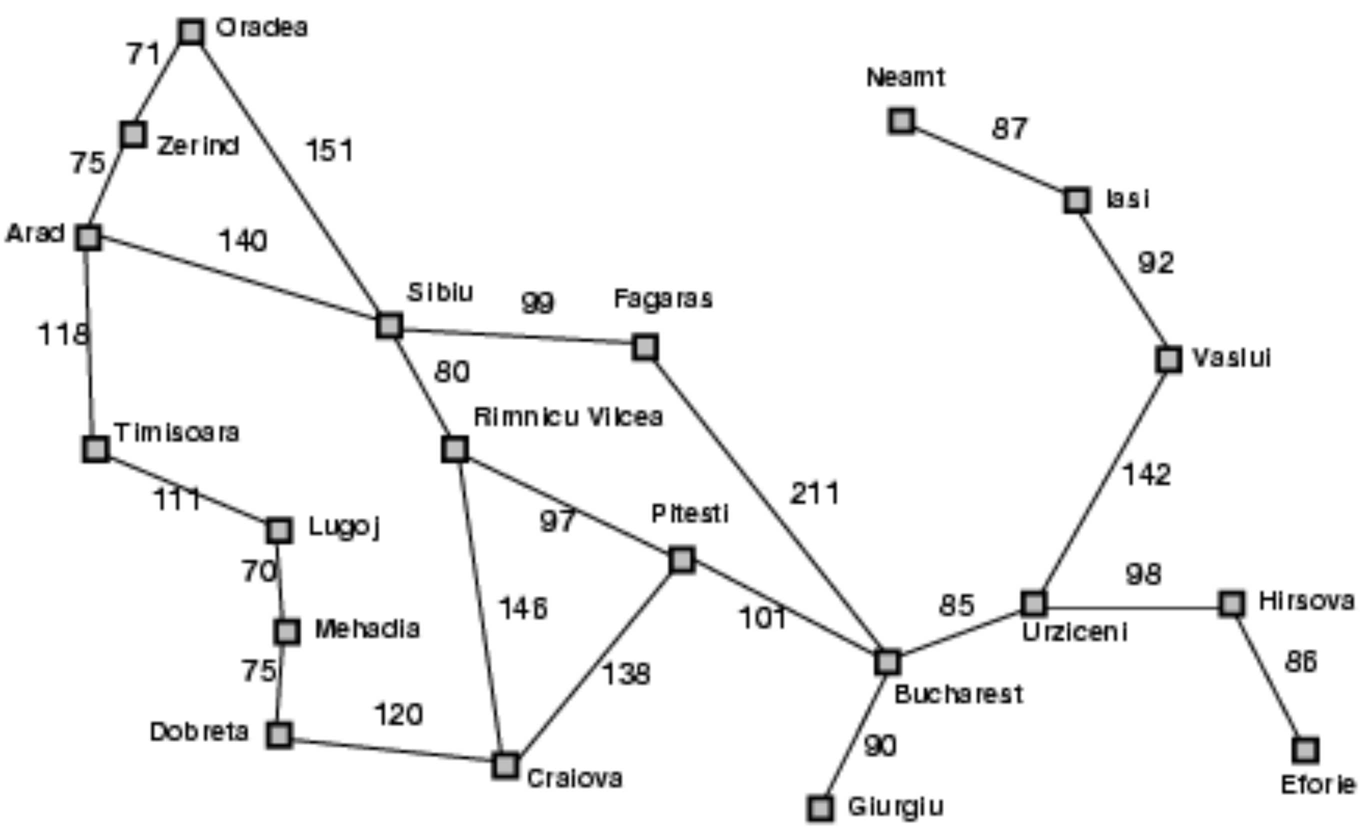
Greedy best-first search

- Evaluation function $f(n) = h(n)$ (heuristic) = estimate of cost from n to *goal*
- Greedy best-first search expands the node that appears to be closest to the goal
- Implementation: as a priority queue to keep the fringe in ascending order of f-values
- e.g., $h_{SLD}(n)$ = straight-line distance from n to Bucharest

Greedy best-first search example

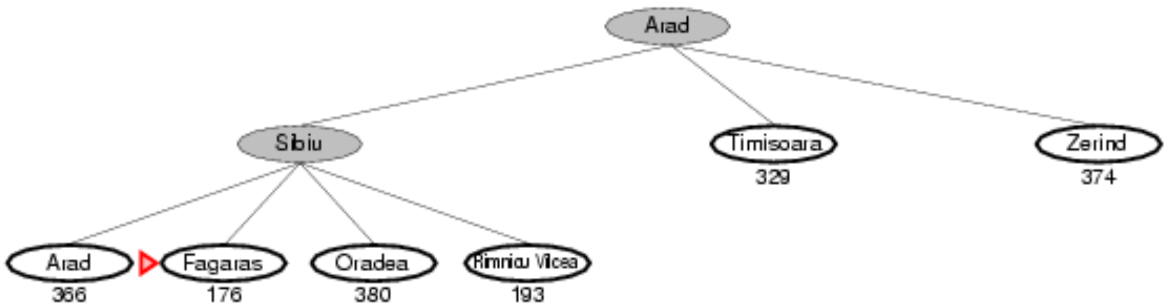


Greedy best-first search example

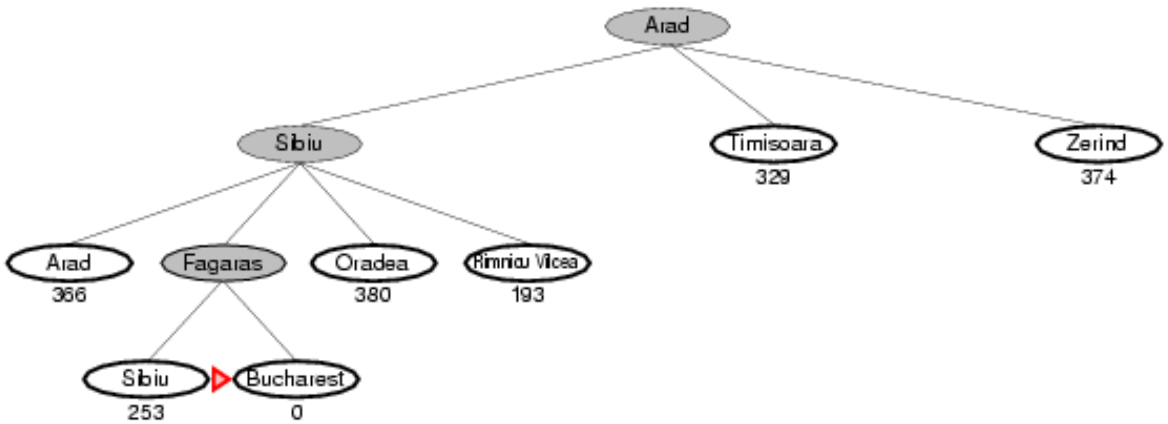


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy best-first search example



Greedy best-first search example



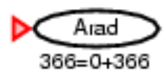
Basic search algorithms

- **Uninformed search**
  - Breadth-first search/ Búsqueda en anchura
  - Depth-first search/ Búsqueda en profundidad
- **Informed search**
  - **Greedy best-first search search**/Búsqueda primero el voraz
  - A*

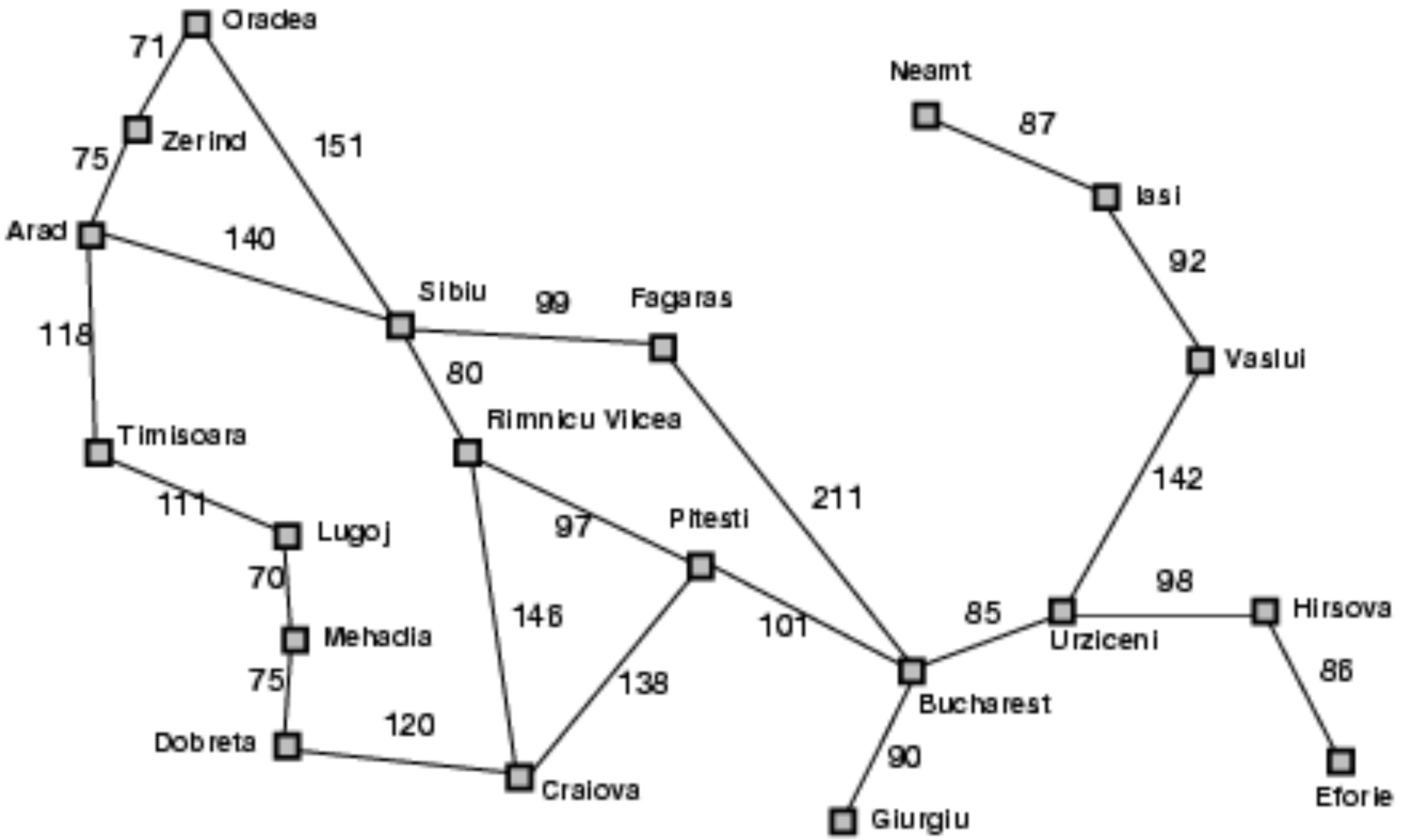
A^* search

- Idea: avoid expanding paths that are already expensive
- Evaluation function $f(n) = g(n) + h(n)$
  - $g(n)$ = cost so far to reach n
  - $h(n)$ = estimated cost from n to goal
  - $f(n)$ = estimated total cost of path through n to goal
- A * is optimal if h(n) is an admissible heuristic such that h(n) never overestimates the cost to reach the goal

A* search example



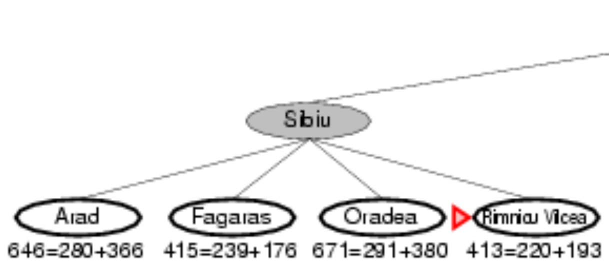
A* search example

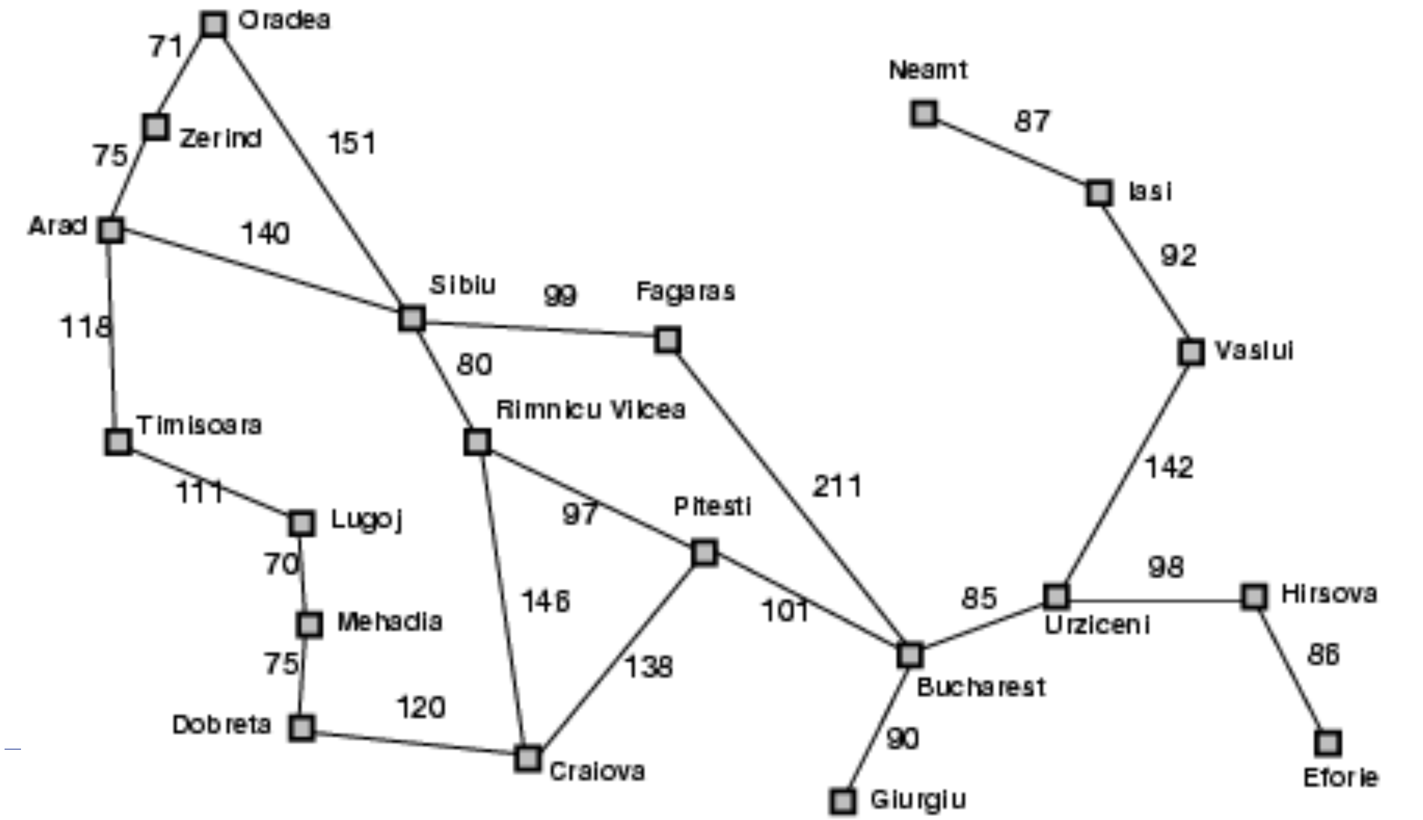


Straight–line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* search example

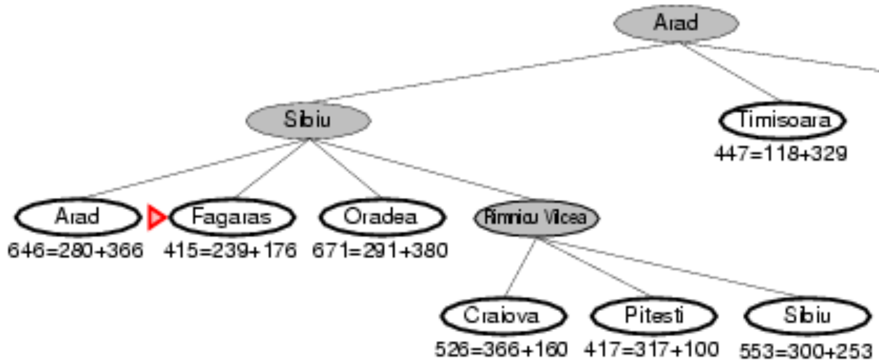


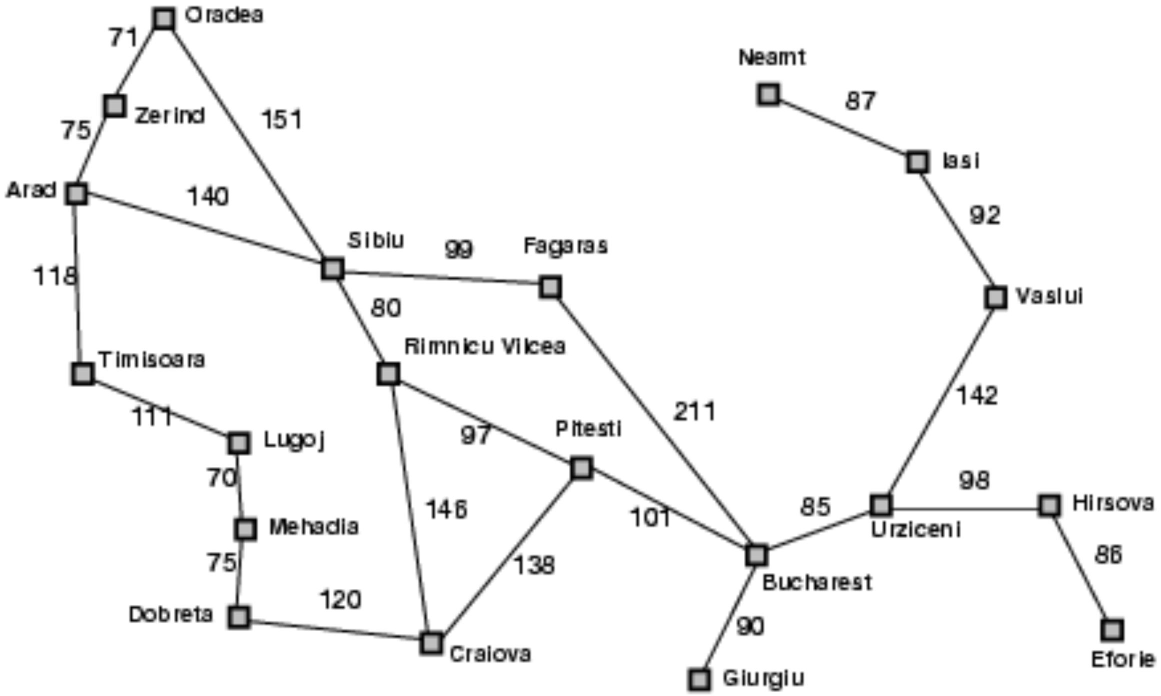


Straight–line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

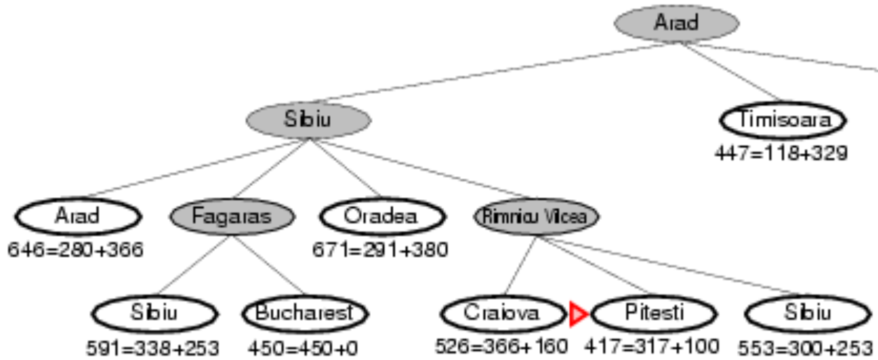
A* search example

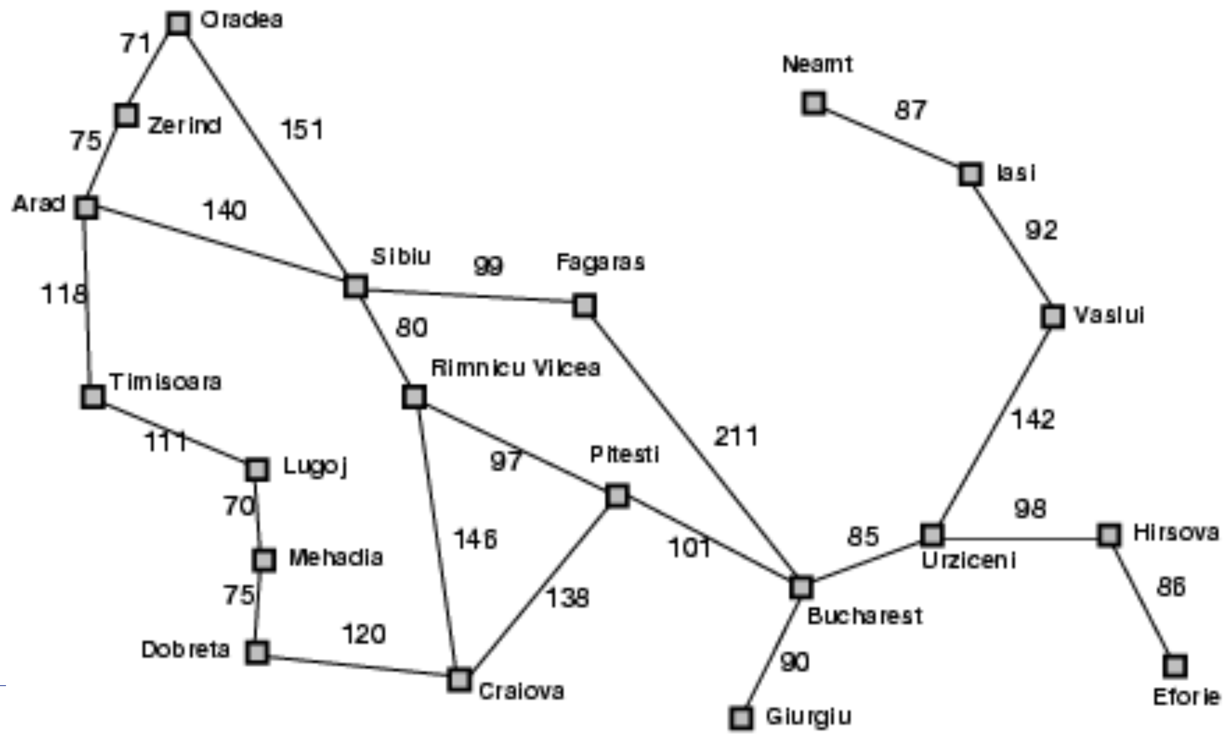




Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

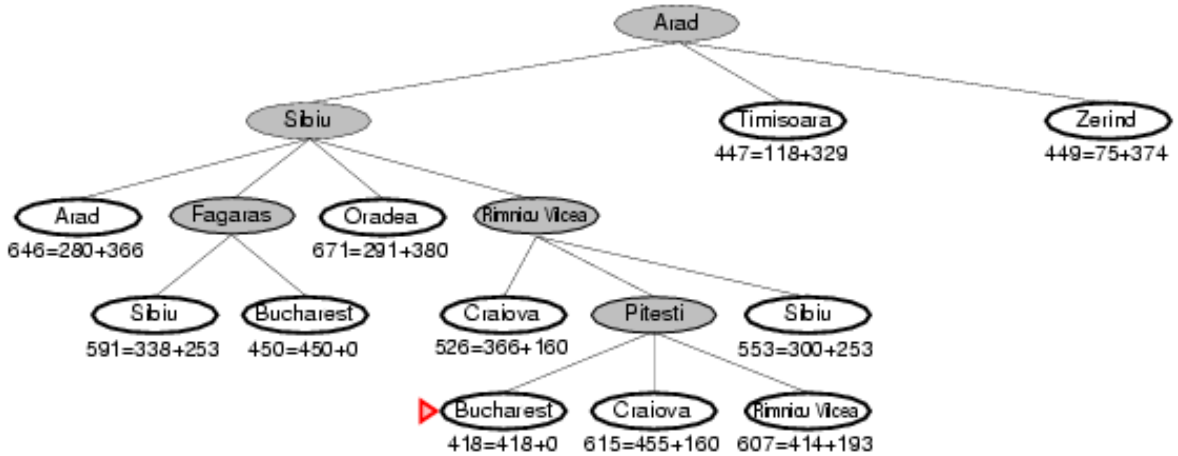
A* search example





Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A^* search example



Outline

- Introduction
- Basic search algorithms
- **Classification of planning algorithms**
- Planning techniques
- Conclusions

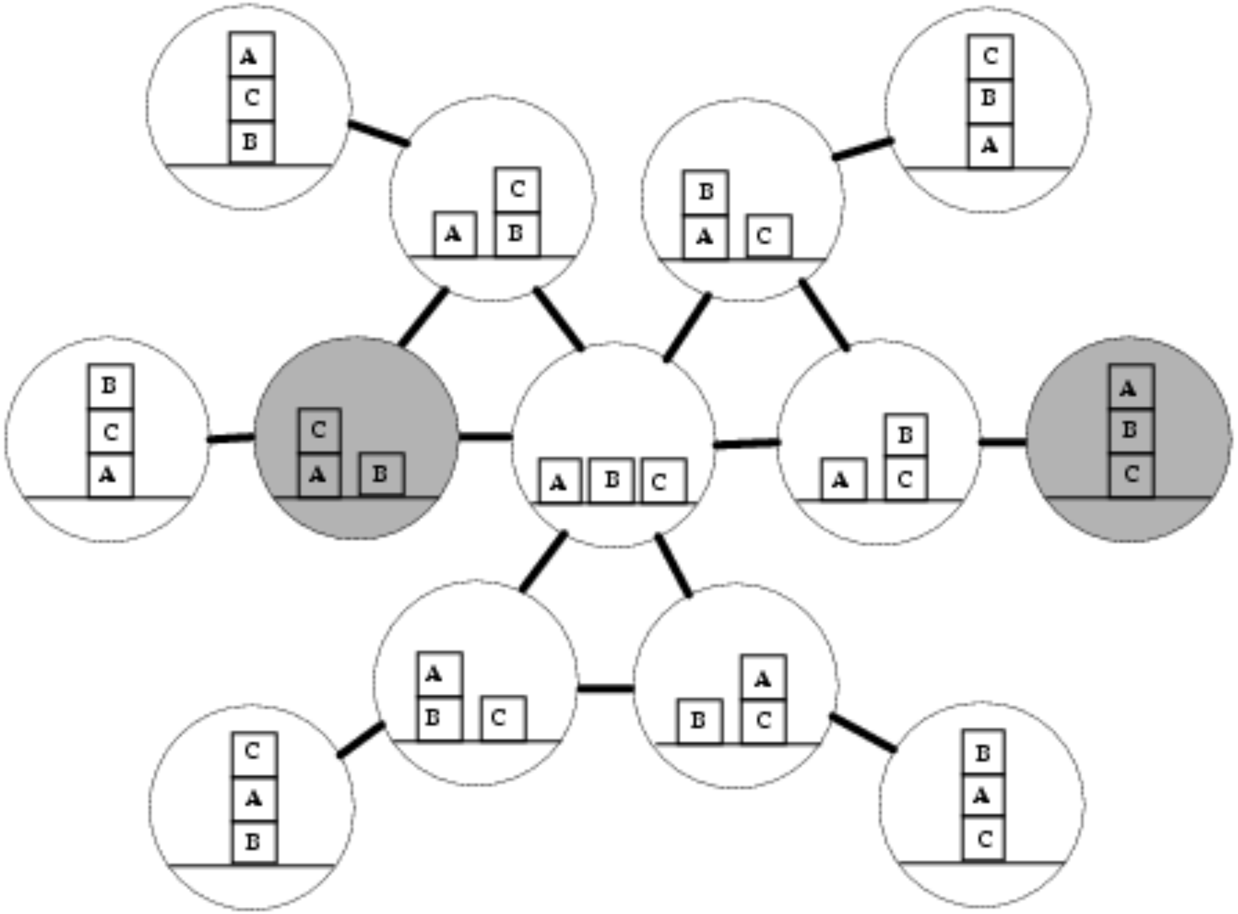
Classification of algorithms

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

State Space Search (SSS)

- 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 so far

State Space Search (SSS)



State Space Search (SSS)

- 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*

State Space Search (SSS)

- The algorithm is:
  - Robust? (the plan that returns, works)
  - Complete? (if there is a solution plan, can find it)
- What algorithm is faster?
  - They have the same complexity, both will do approx. ”n" decisions before finding the solution
  - The number of decisions at each branch point suppose to be "b"
  - By experience, the branching factor (b) is smaller in one. What?

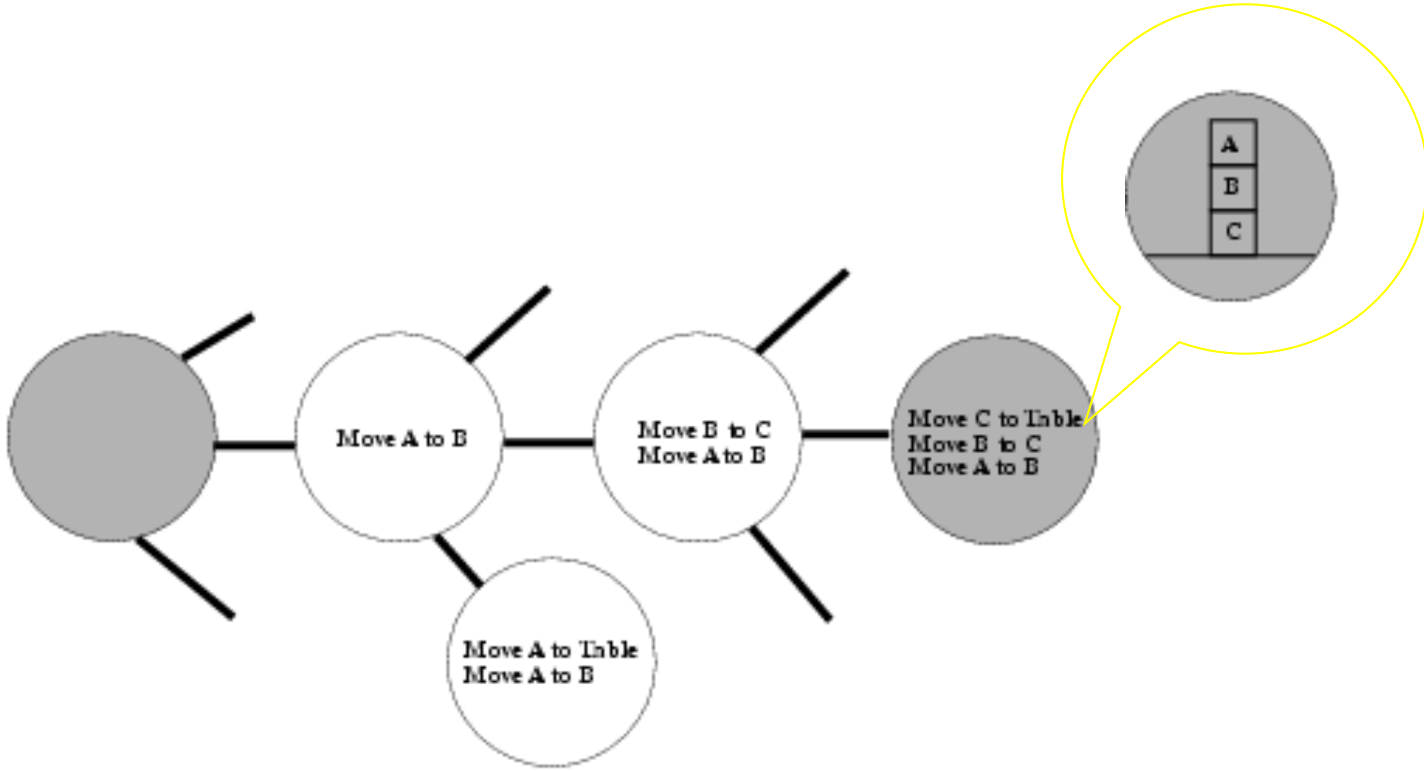
$O(b^n)$

Regression

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)
- 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
- Basic search algorithms
- Classification of planning 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 (length, cost)

Planning techniques

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

TO Planners

- The solution is an **ordered** sequence of actions
- Distinguishing TO must be separated from the distinction of SSS and PSS based planners
  - SSS: each search state corresponds to a state in the world
  - PSS: each search state corresponds to a plan

- Generally, TO is associated to SSS: Prodigy or VVPLAN, although there are PSS: INTERPLAN or WARPLAN

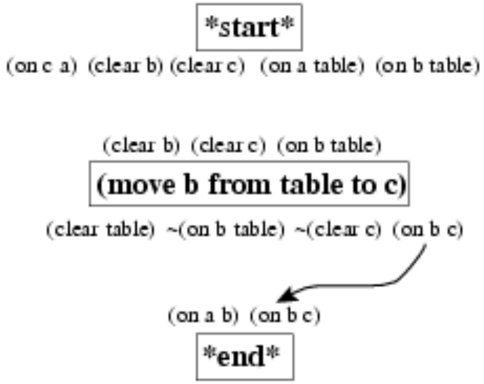
Planning techniques

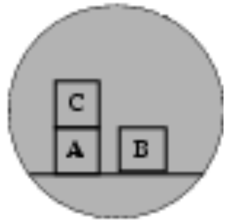
- Total Order Planners (TO)
- **Partial Order Planners (POP)**
- Hierarchical Task Network (HTN)
- Graph-based Planners (GP)
- SAT-based planners
- Heuristic Search Planners

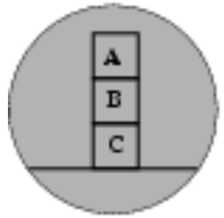
Partial Order Planners (POP)

- Perform PSS
- The planning algorithm implements the *least commitment* technique
  - Only essential planning decisions are saved because it is not necessary to commit
  - The causal link structure is responsible for storing them
    - *3 fields: producer, consumer and the proposition*
  - As can be actions that threaten it, we can apply: $A_p \xrightarrow{Q} A_c$
    - Demotion: add the restriction before the step that threats it
    - Promotion: add the restriction after the step that threats it
    - Separation: add the restriction to the variable binding
    - Confrontation: add the negation to the conditional effects
- Examples: UCPOP, Cassandra, ZENO, VHPOP

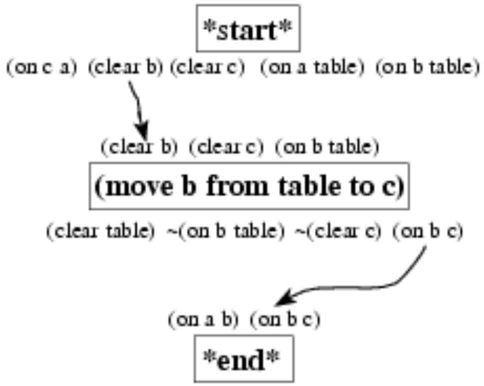
UCPOP





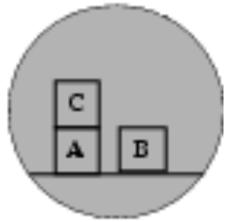


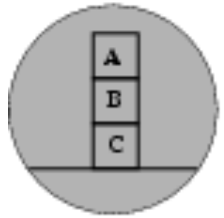
After adding a causal link to support `(on B C)`, the plan is as shown and **agenda** contains {`(clear B) (clear C) (on B Table) (on A B)`} as open propositions.

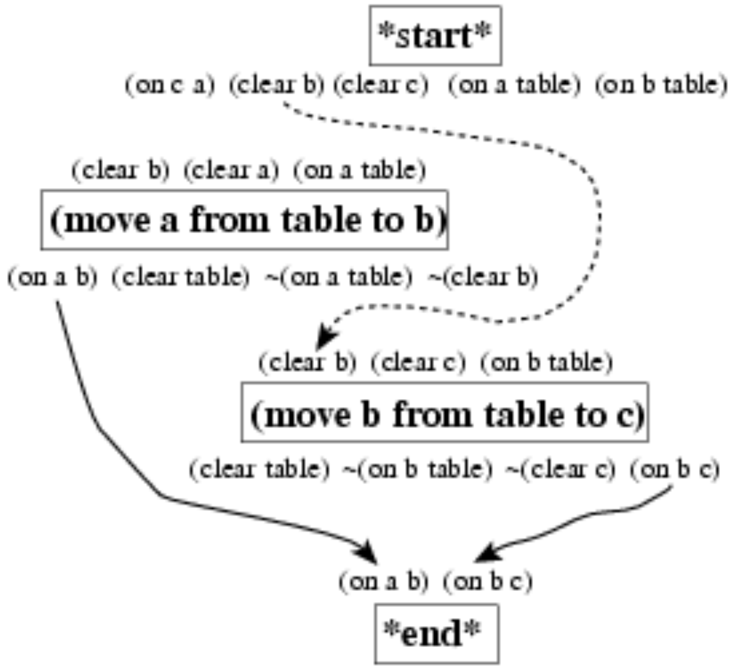


After adding a causal link to support `(clear B)`, the plan has two causal links and **agenda** is set to {`(clear C) (on B Table) (on A B)`}.

UCPOP

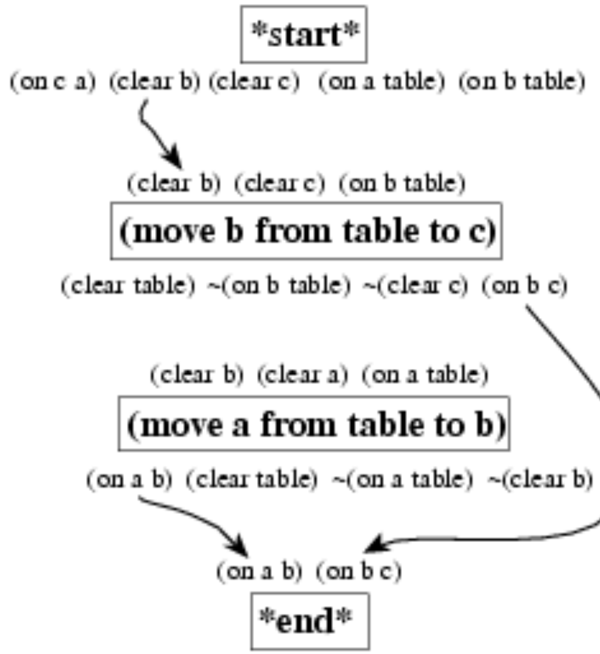


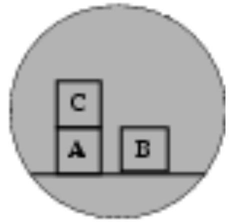


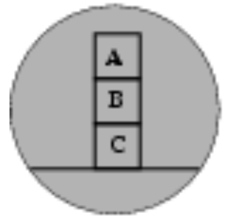


Since the `move-A` action could possibly precede the `move-B` action, it threatens the link labeled `(clear B)` as indicated by the dashed line.

UCPOP

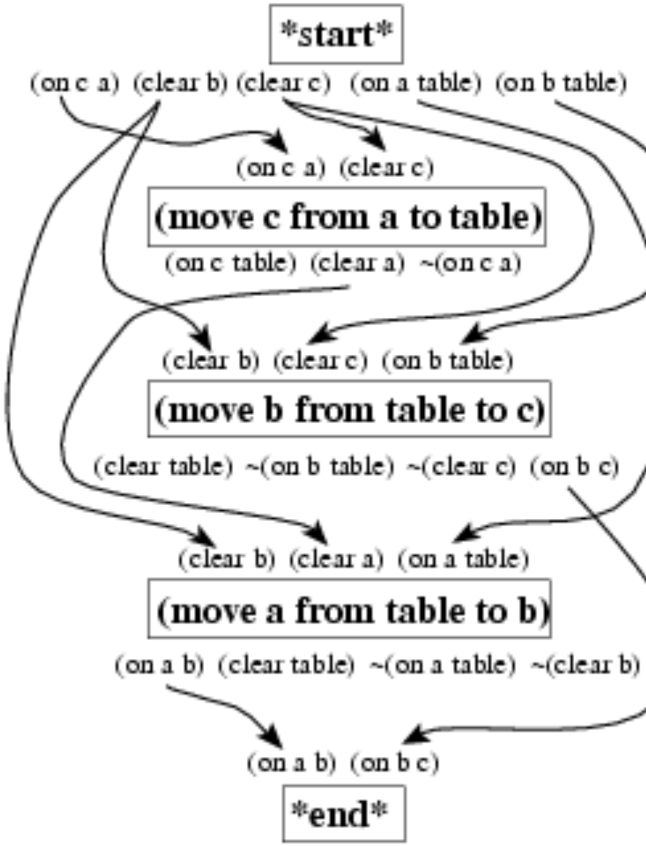


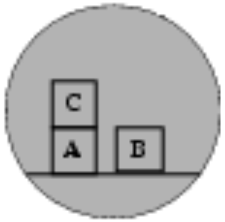




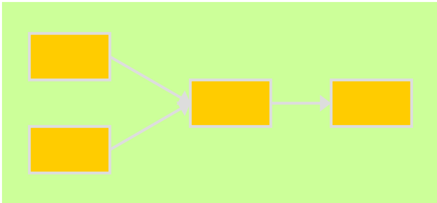
After promoting the threatening action, the plan's actions are totally ordered.

UCPOP

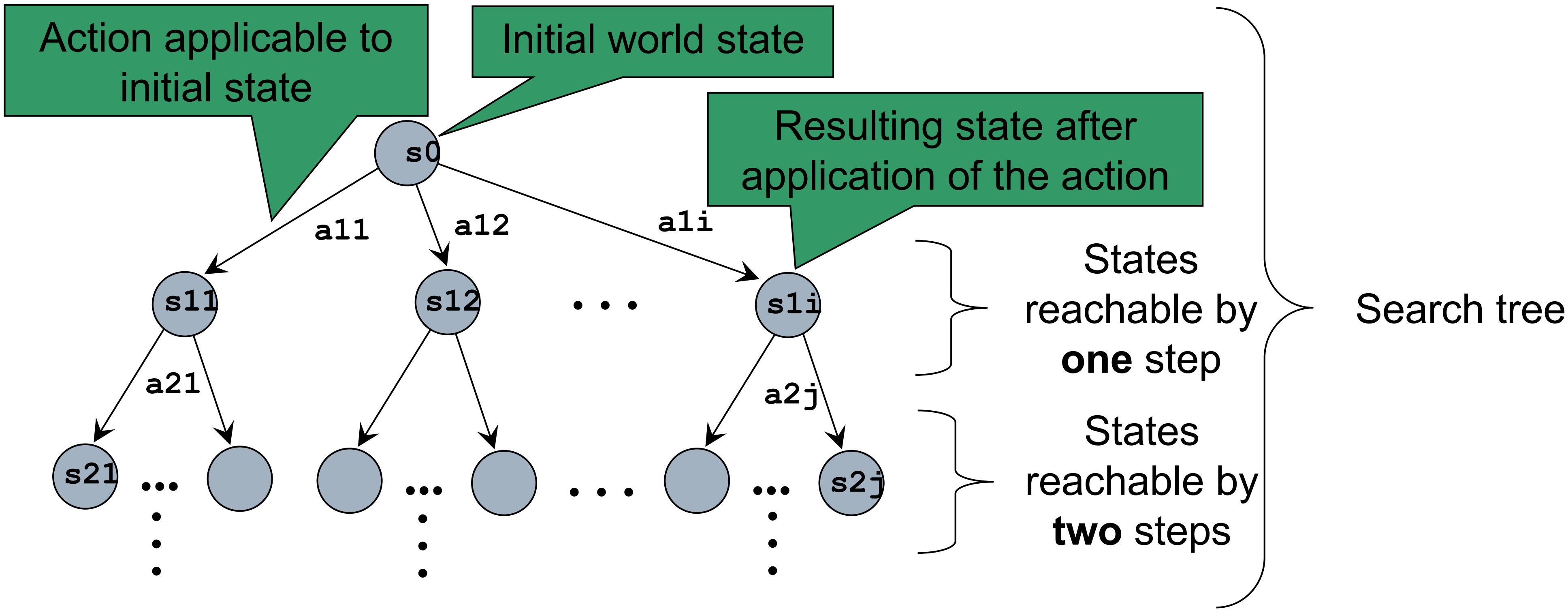

start
(on c a) (clear b) (clear c) (on a table) (on b table)
(on c a) (clear c)
(move c from a to table)
(on c table) (clear a) ~(on c a)
(clear b) (clear c) (on b table)
(move b from table to c)
(clear table) ~(on b table) ~(clear c) (on b c)
(clear b) (clear a) (on a table)
(move a from table to b)
(on a b) (clear table) ~(on a table) ~(clear b)
(on a b) (on b c)
end


C
A
B


A
B
C



State reachability: planning tree



- **expand** the tree until a goal state satisfying goal is reached

Unmanageable size ⇒ cannot be stored in memory

Planning techniques

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

HTN

- Problems and operators are arranged in a network or {} of task (called actions) that correspond to transition states
- High-level tasks are reduced into low levels tasks
- A method maps a task in a network of partially ordered tasks with few restrictions
- The algorithm iteratively expands tasks and resolves conflicts until a plan consisting of primitives and free tasks conflict is found

HTN

- Differences between HTN and STRIPS - style planners lies in what they do and how they plan for:
  - STRIPS: the objective is to find a {} of ordered actions from the IS to goals. It finds appropriate operators that have the desired effects and making their preconditions, the subgoals
  - HTN: it plans looking for task network that may include other things besides goals. Plan by decomposing tasks and conflict resolution. Methods should contain all possible ways to get tasks, which is much more tedious

HTN

- Examples:
  - NONLIN
  - O-PLAN
  - DEVISER
  - SIPE
  - NMRA
  - SHOP: plans the tasks in the order they are executed. Domain Independent
  - TALPlanner (Temporal Action Logic planner) domain dependent

Planning techniques

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

Graph-based Planners

- The search structure is based on a planning Graph→ Graphplan
- The graph is directed and layered. contains:
  - 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 and nop

Graph-based Planners

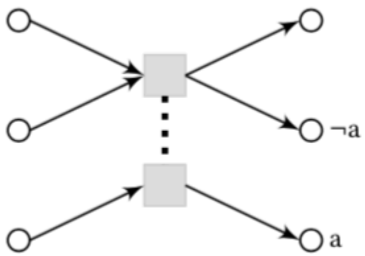
- Graphplan algorithm works in two alternating phases
  - expands (add layers) the planning graph until the last proposition layer satisfies the goal condition
  - tries to extract a valid plan (backtracking) from the planning graph

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

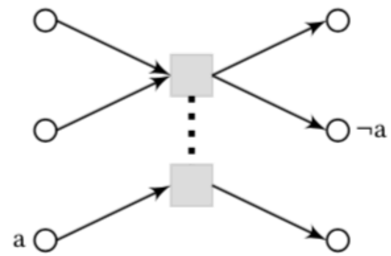
Graph-based Planners

- 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
  - Between incompatible propositions: negated literals or all actions that can achieve them are mutex in the previous step

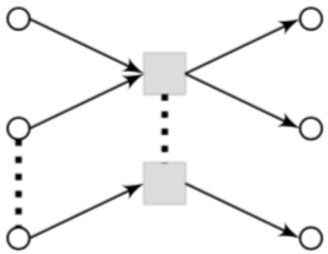
Graph-based Planners



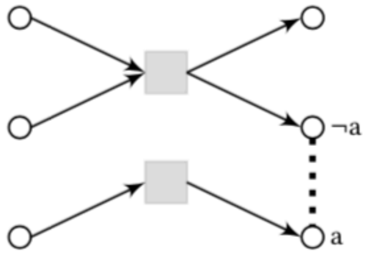
(a) Inconsistent effects



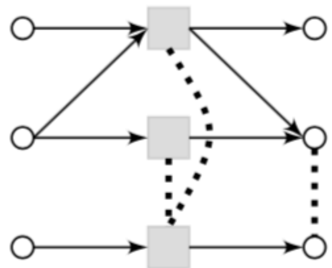
(b) Effects clobbering preconditions



(c) Competing needs

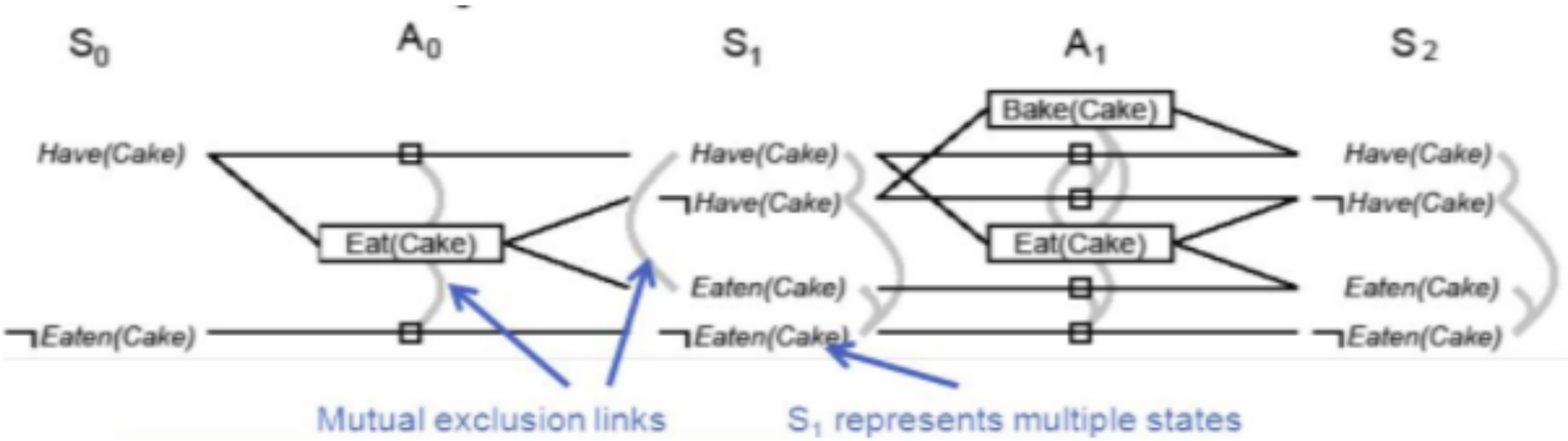


(d) Contradiction



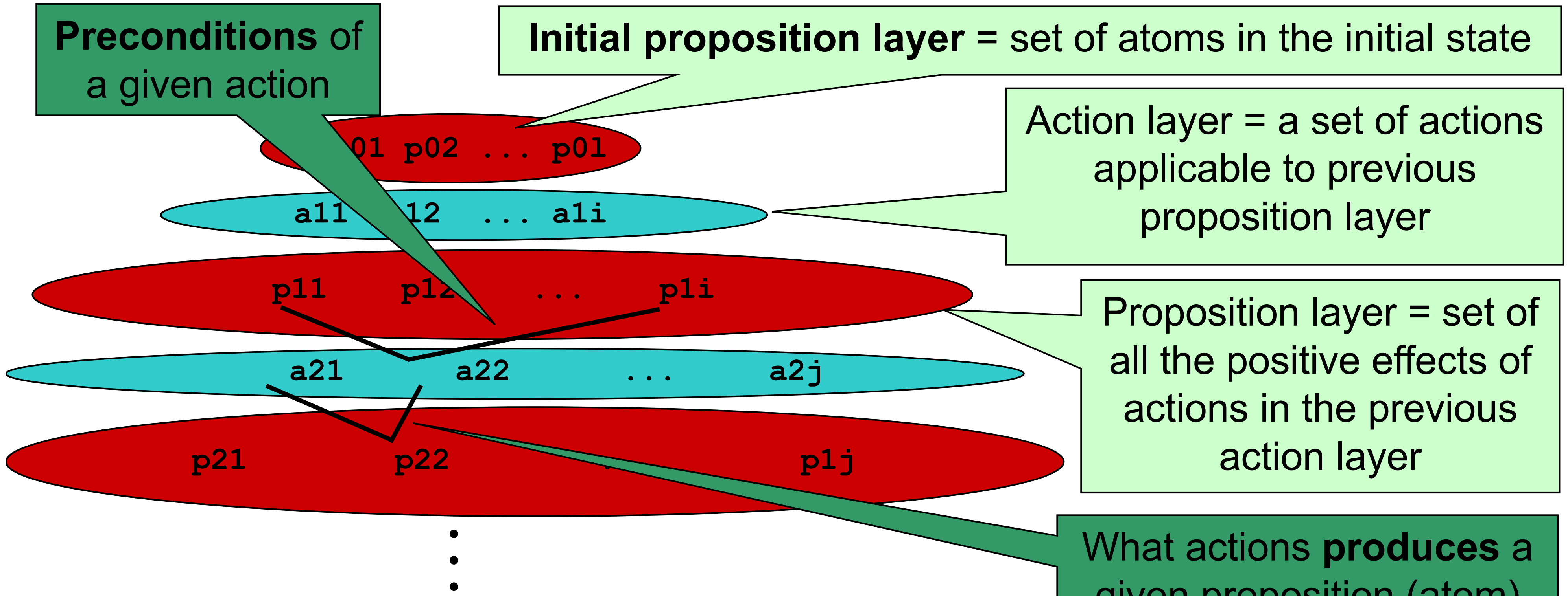
(e) Inconsistent support

Graphplan




```
(: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))))
```

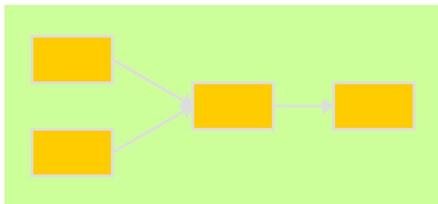
State reachability: planning graph



- 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

Graph-based Planners

- Example: SGP, TGP, IPP, SAPA, STAN, LPG
- Output



Planning techniques

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

SAT

- It is based on the evaluation of the satisfiability of a logical sequence (SAT-based)
- The philosophy of the algorithm is:
  - The planning problem is translated to CNF
  - Guess the length which aims to achieve the goal
  - A set of propositional clauses is generated to check the satisfiability
  - Apply algorithms studied in the logic chapter (DPLL, WALSAT, LMTS-style)
- Examples:
  - SATPLAN, Blackbox, etc

Planning techniques

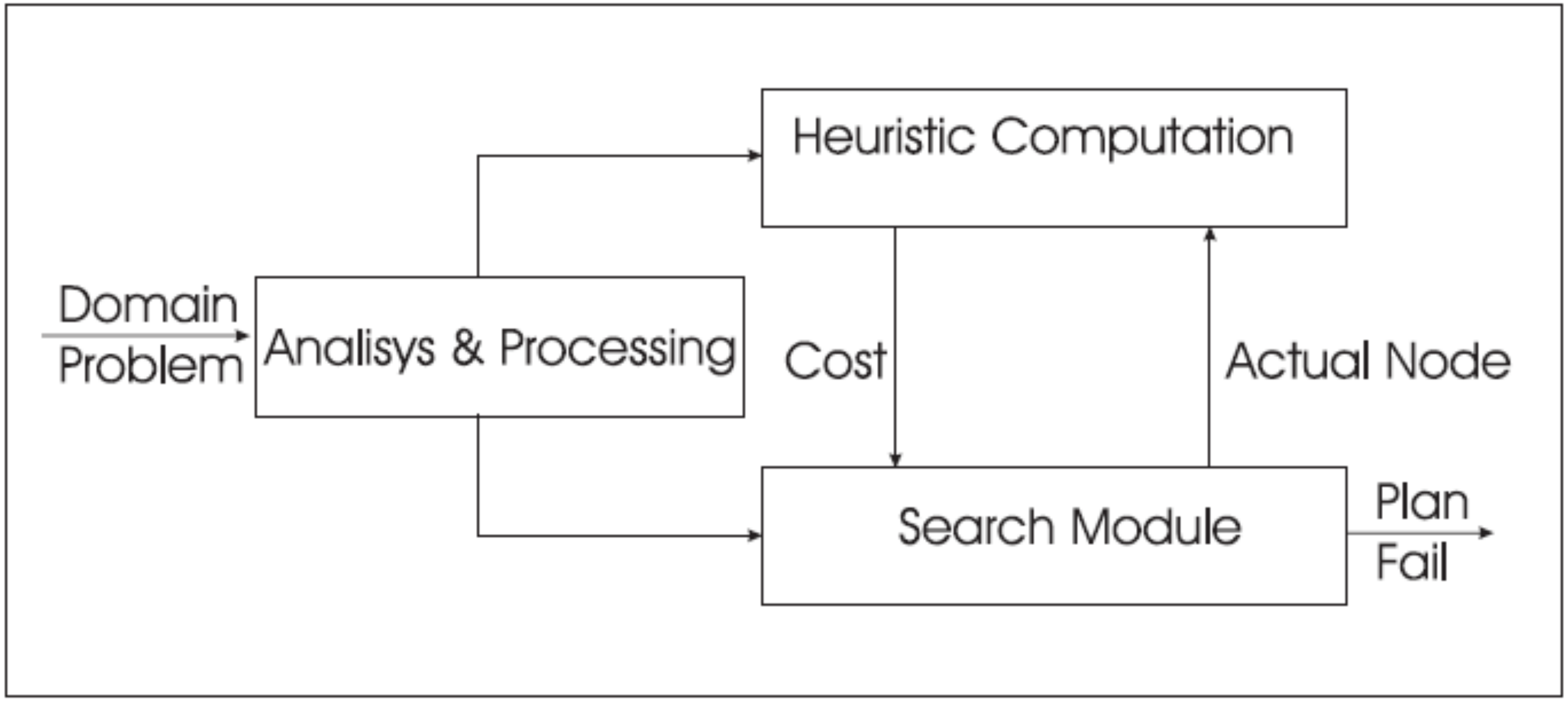
- Total Order Planners (TO)
- Partial Order Planners (POP)
- Hierarchical Temporal Planners (HTN)
- Graph-based Planners (GP)
- SAT-based planners
- **Heuristic Search Planners**

Heuristic Search Planners (HSP)

- 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
- Heuristic: not consider the delete effects
  - FF: based on a relaxed GP

HSP

- General architecture



HSP

- **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

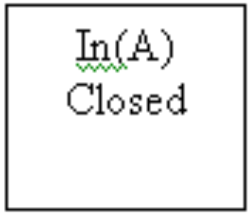
FF: h

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**	$\longrightarrow$	**In(B)** **Closed** **Polished**

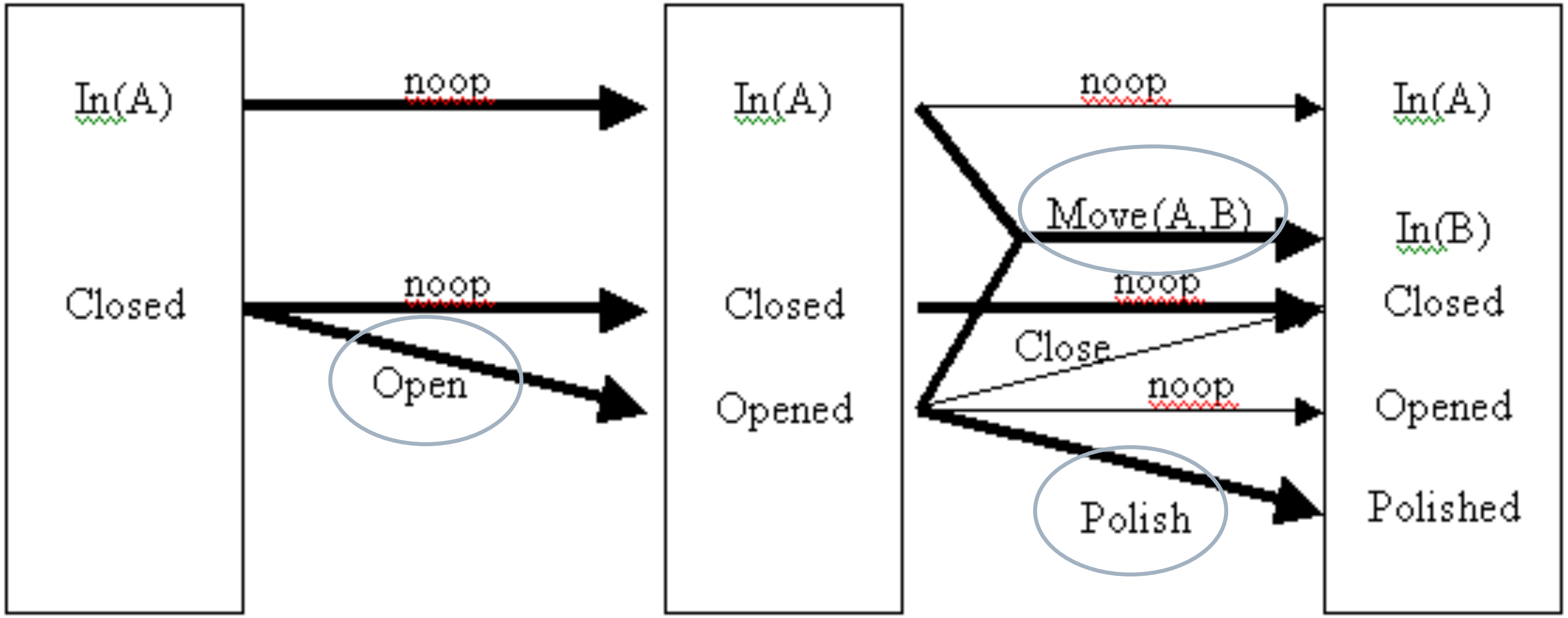
FF: h

Node 1

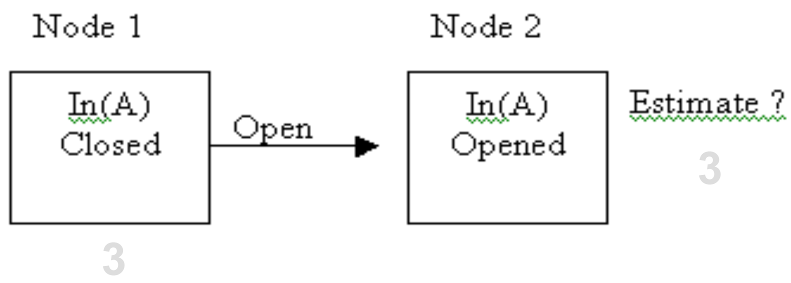


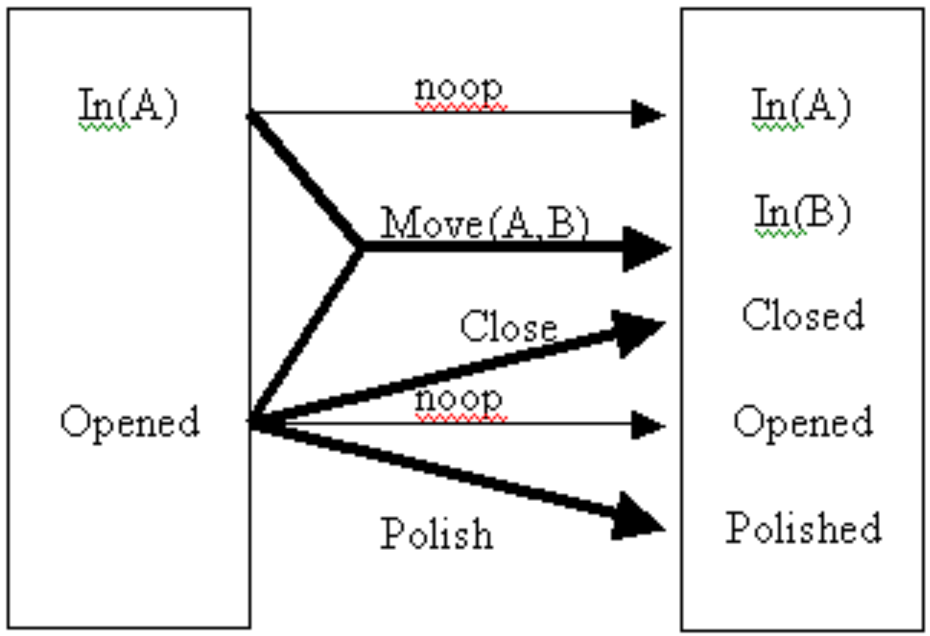
Estimate?

3

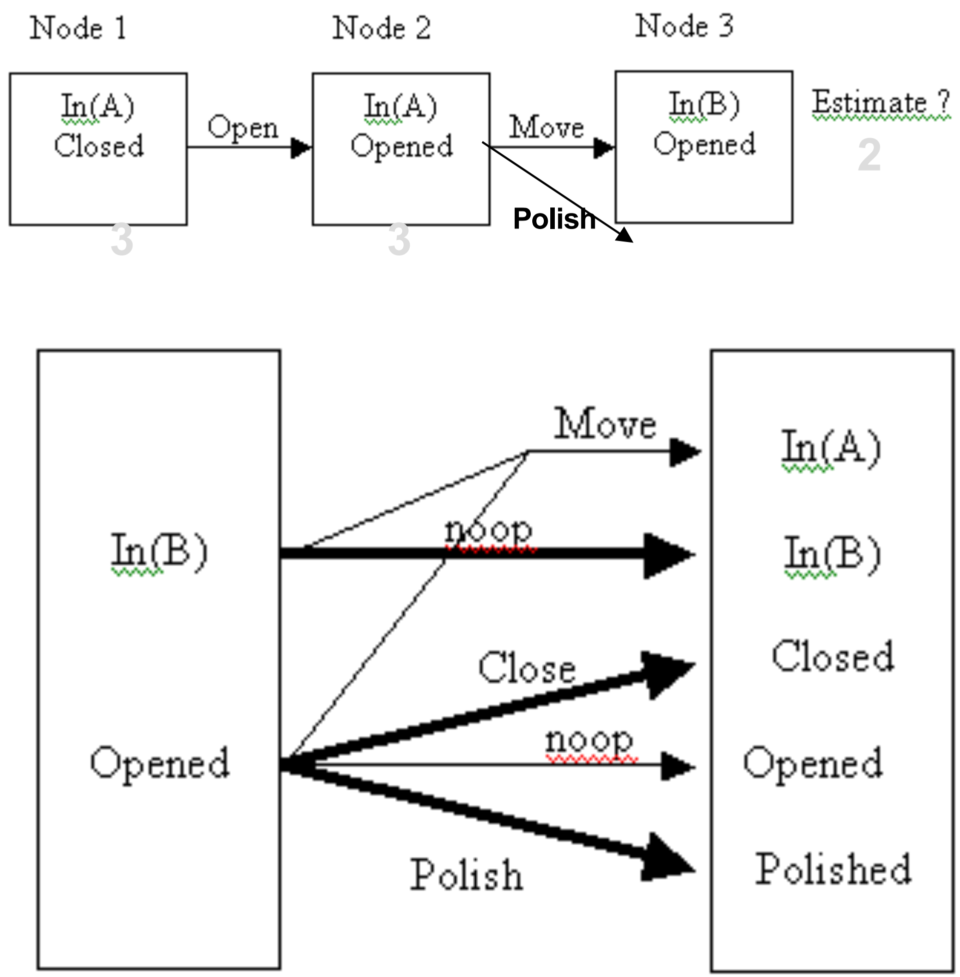


FF: h


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

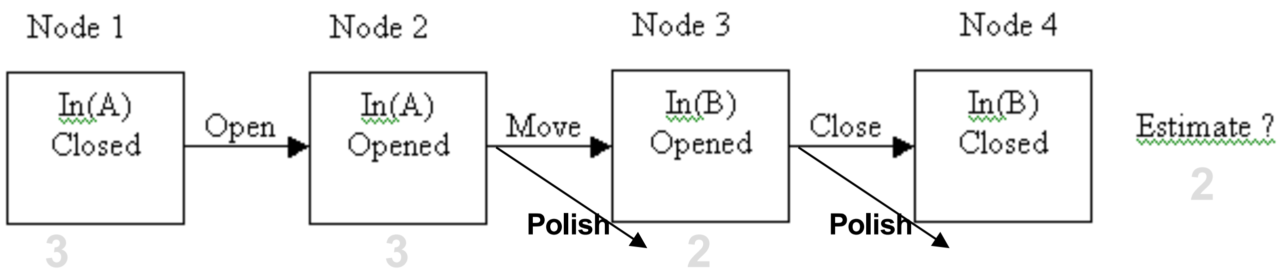

In(A)
Opened
noop
Move(A,B)
Close
noop
Polish
In(A)
In(B)
Closed
Opened
Polished

FF: h

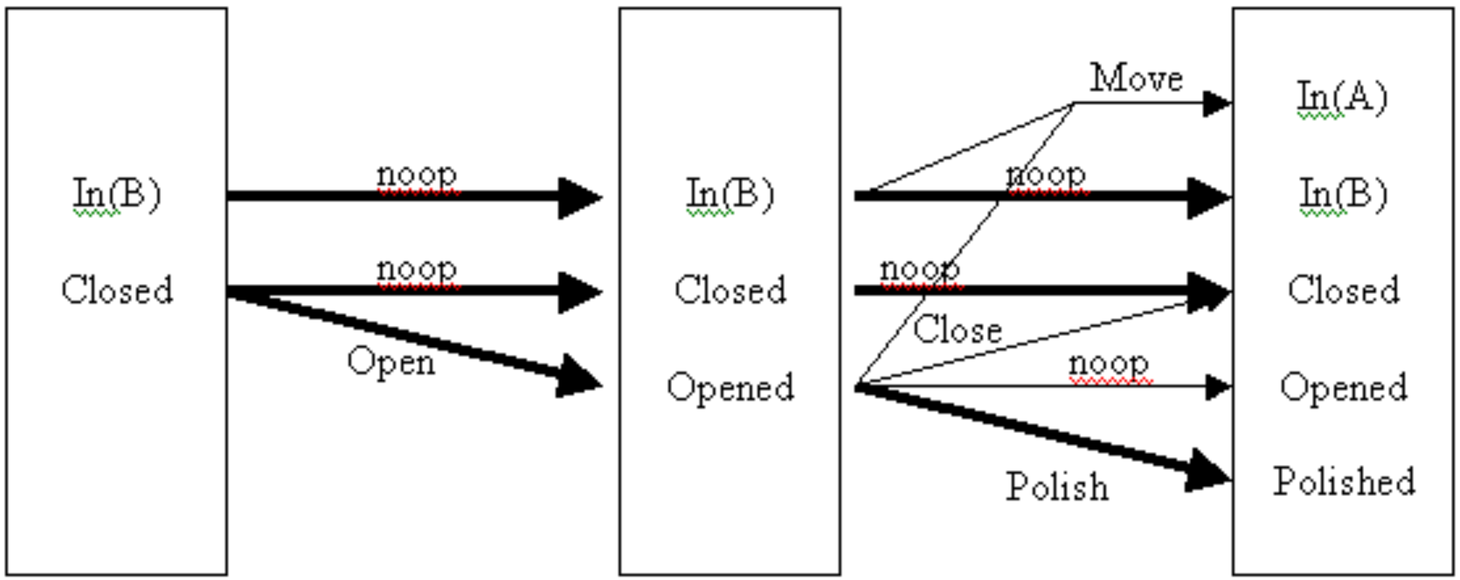


Node 1
Node 2
Node 3
In(A)
Closed
Open
In(A)
Opened
Move
In(B)
Opened
Estimate ?
2
3
3
Polish
In(B)
Opened
Move
noop
Close
noop
Polish
In(A)
In(B)
Closed
Opened
Polished

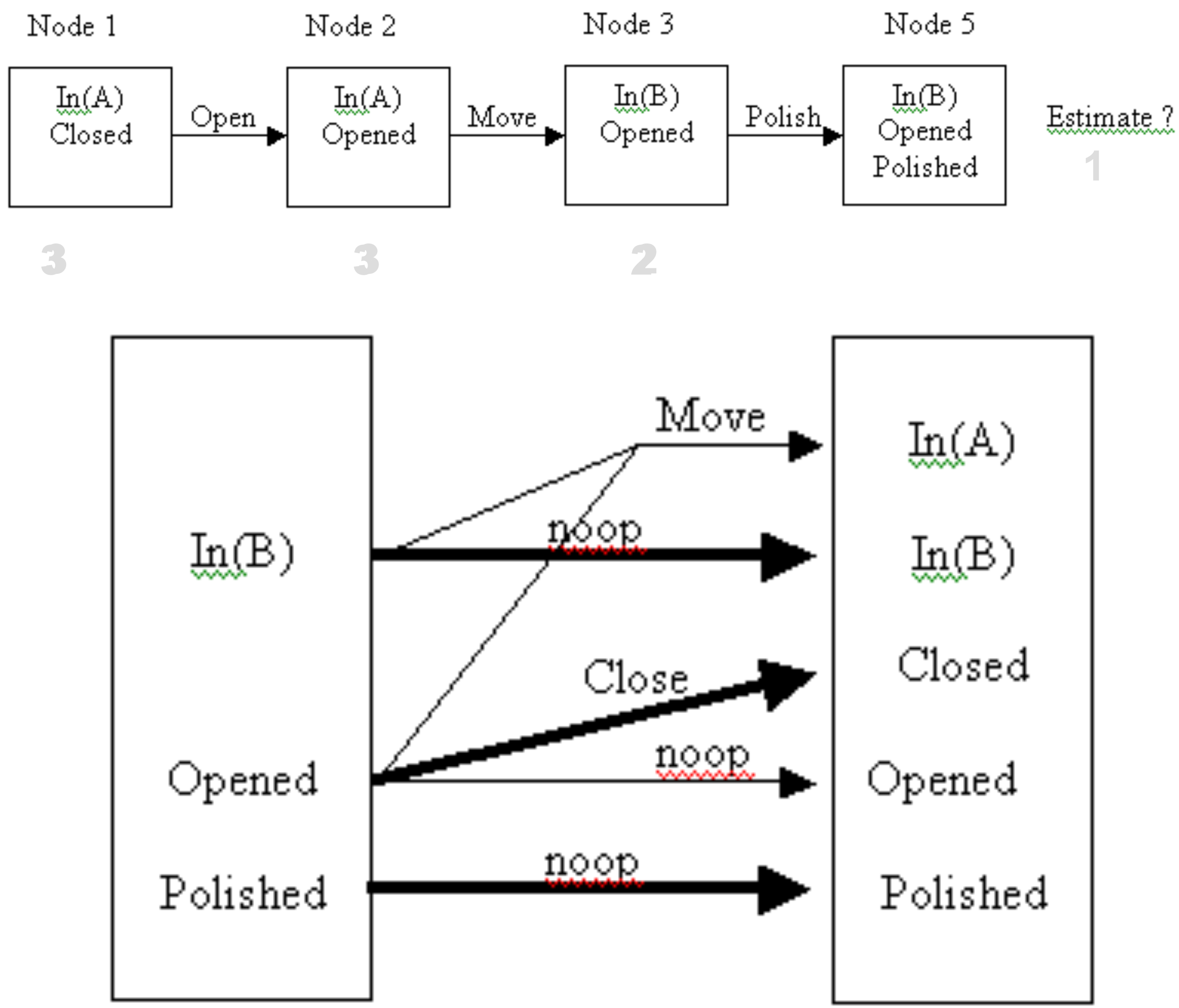
FF: h



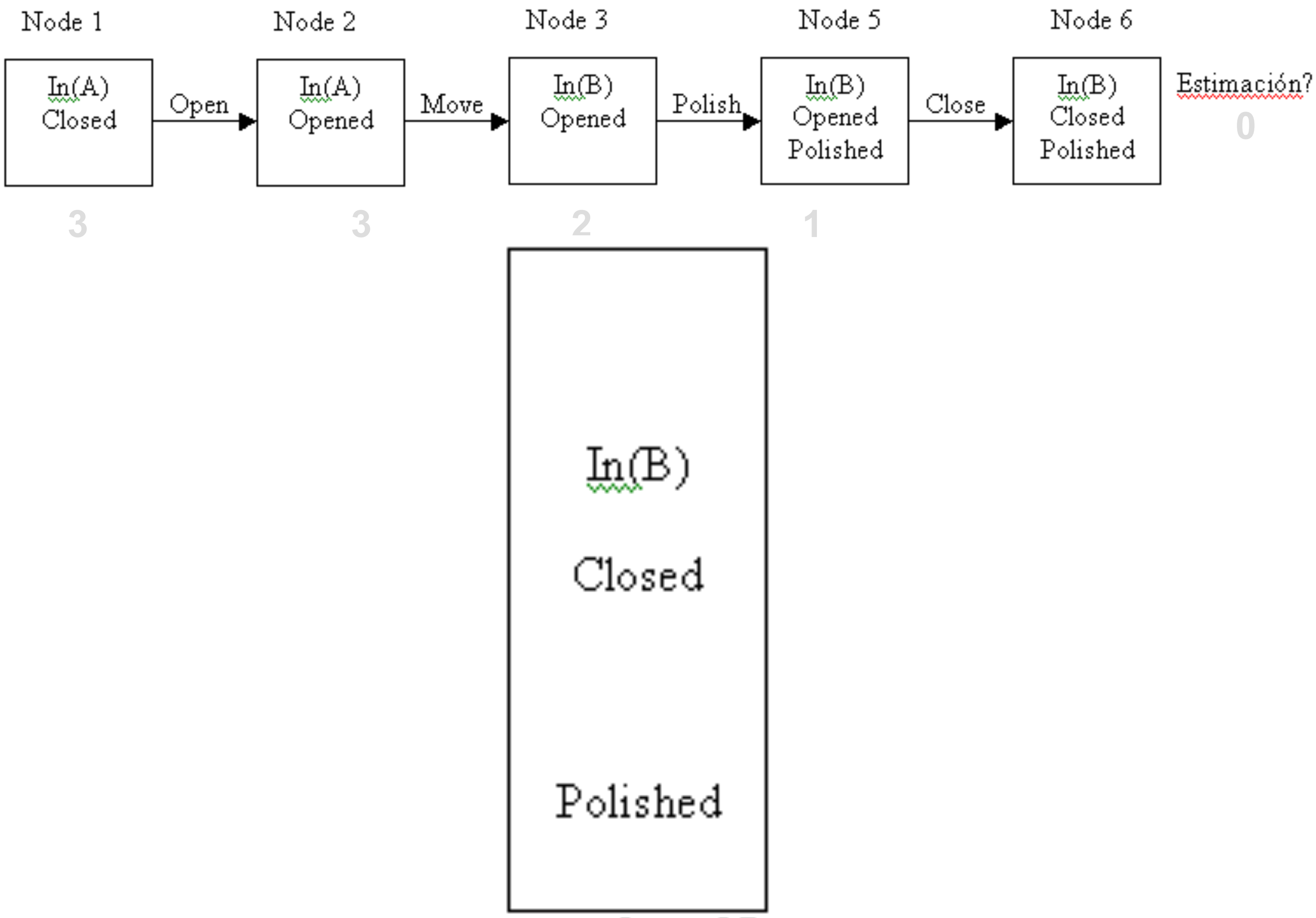
It does not improve the previous result, try Polish



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

Outline

- Introduction
- Basic search algorithms
- Classification of planning 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

Universidad de Alcalá

Planning Techniques

ISG

Intelligent Systems Group