TÉCNICO LISBOA

Artificial Intelligence and Decision Systems

Inteligência Artificial e Sistemas de Decisão – MEEC, Maer,

Luís Manuel Marques Custódio

North tower – ISR / IST

lmmc@isr.ist.utl.pt

Problem solving by searching

Often we don't know the algorithm to solve a problem; we only have a specification of what is a solution

in this case, **we have to search for the solution**

A **problem-solving agent** is a goal-based agent that acts on the environment, leading him to go through a series of states in order to achieve the desired goal

Problem solving is often formulated as a combinatorial **search** problem – agent must find the *right* sequence of actions to achieve the goal

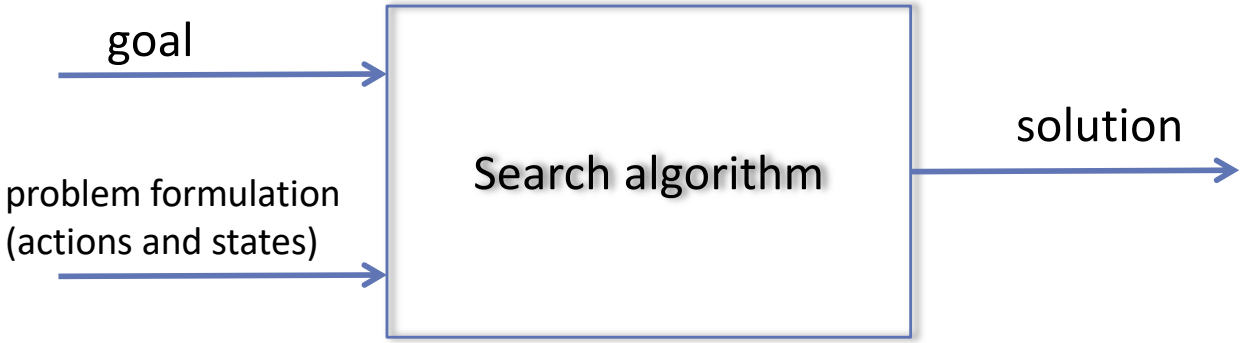
TÉCNICO LISBOA

Problem solving by searching

Steps to formulate a search problem

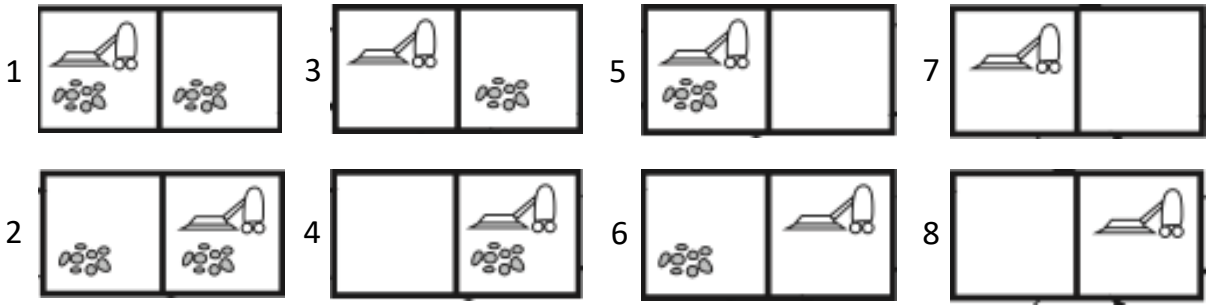
- Goal formulation
  - a goal is a (world) state or a set of states
- Problem formulation
  - Choosing what actions and states to consider, given the goal
  - an action is a transition between states
- Search
  - Process of looking for a sequence of actions that reaches the goal
  - That sequence is called a **solution**
- Execution
  - Practical implementation of the solution

Problem solving by searching



Example: vacuum cleaner world

- Goal: cleaning of the two squares
- Actions: {GoLeft, GoRight, Suck}
- States: how many?

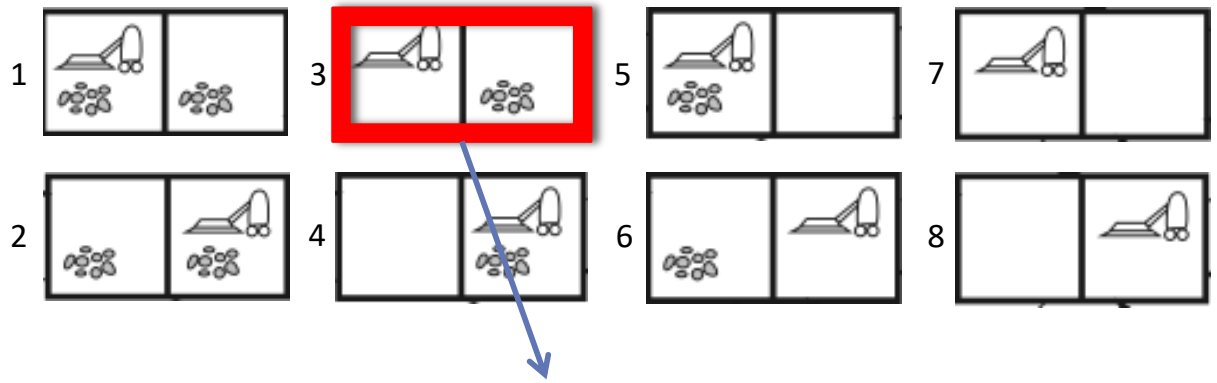


Types of search problems

- Single-state problem

  when each action generates a single state transition

  world must be fully observable and deterministic



Solution: [right, suck]

what happens if the world is not fully observable (or non deterministic)?

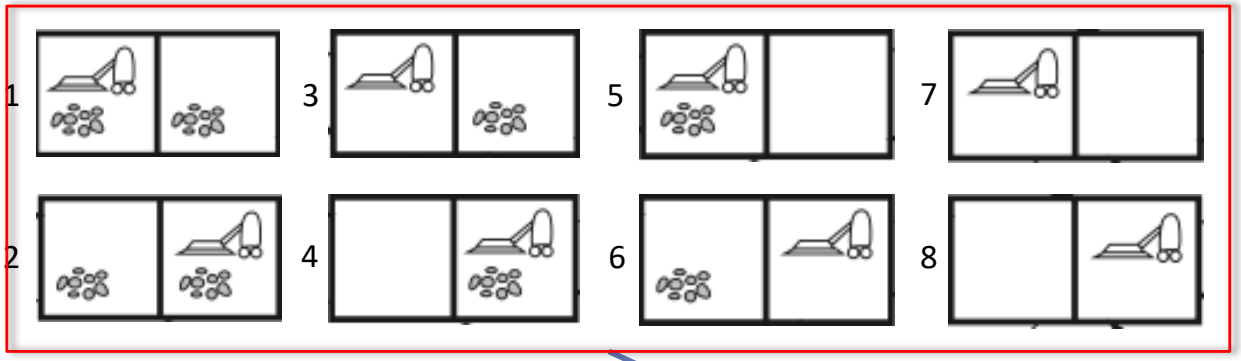
Examples:

- Lack of sensors
- Faulty cleaner

Types of search problems

- Multi-state problem

  when an action establishes a transition to a set of states



Solution: [right, suck, left, suck]

if the cleaner is ok

What if there are no sensors and the cleaner is faulty?

Example: suppose that the world is in one of these states, there is no dirt sensor, and the vacuum cleaner is faulty: it cleans if there is dirt, but it leaves dirt if it is clean

Which sequence of actions allows to reach the goal?

TÉCNICO LISBOA

Types of search problems

- Contingency problems

  when the problem involves unpredictable situations whose solution can be selected only when situations occur

  Example: {Suck, GoRight, Suck only if there is dirt} [if the robot has local sensing]

- Exploration problems

  when the agent has no information about the environment or the consequences of their actions

  The agent is forced to explore the world in order to build a map of it and learn the effects of their actions

In this course, we only study the single-state problems

TÉCNICO LISBOA

Problem formulation

Elements of a search problem:

- Full state space: set of possible states of the world
- Initial state: initial condition of the world
- Operators: set of actions that change the world state
  Path: sequence of states generated by a sequence of operators
  State space of the problem: set of all states reachable from the initial space by any sequence of actions
- Goal test: condition that has to be satisfied at a goal state
  can be single state, multiple states or defined by a characteristic property of the environment
- Path cost: cost associated to a sequence of states/operators
  it is defined as the sum of the costs of the operators used to form the path

Solution: is an action sequence (a path) that leads from the initial state to a goal state

Problem formulation

Elements of a search problem:

- Evaluation of the solution quality:

  Path cost

  Search cost
  in terms of computation time and memory

Optimal solution: is a solution that has the lowest path cost among all solutions

- Problem description

  level of detail that the information about the problem must have both in describing states and actions

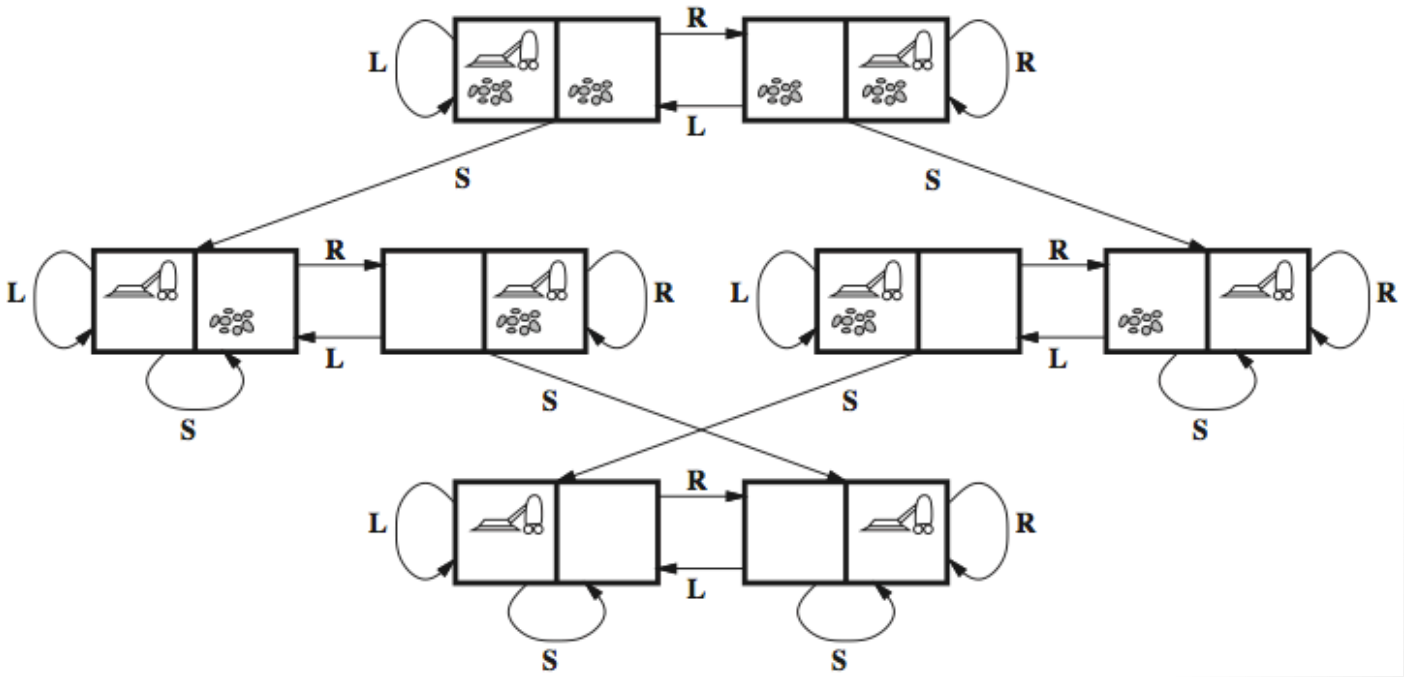
Example: vacuum cleaner
- brand
- color
- wire Length
- energy consumption

Abstraction: process of removing from a problem representation details considered irrelevant for the goal

Search problem examples

Vacuum cleaner

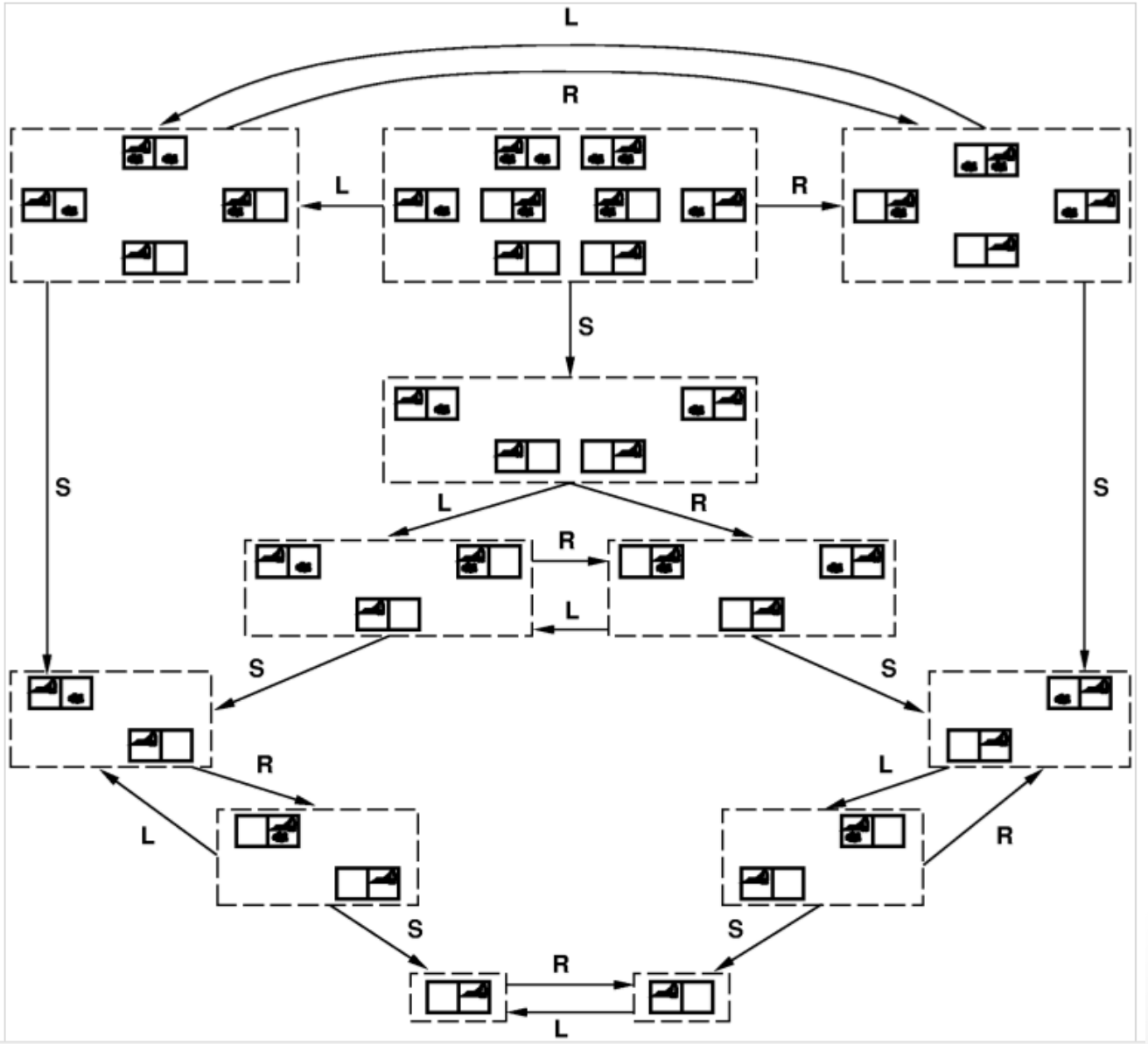
- states: $<r, d_1, d_2>$, where r is the robot position, d_1 and d_2 are T/F representing presence of dirt in each room
- operators: go left (L), go right (R), and suck (S)
- goal test: $d_1=F$ and $d_2=F$
- initial state: $<r, d_1, d_2> = <1,T,T>$
- path cost: 1 for each operator (also known as step cost)



TÉCNICO LISBOA

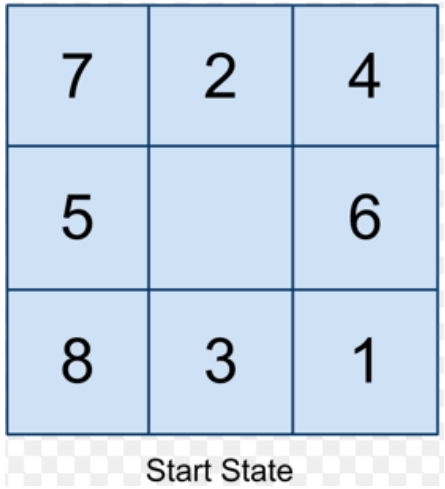
Search problem examples

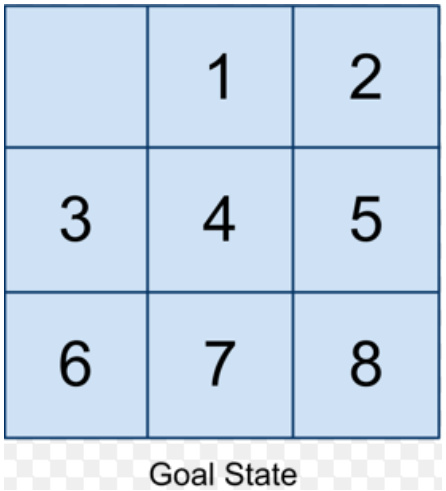
Vacuum cleaner with no localization sensors



Search problem examples

8-puzzle





States: specify in a 3x3 matrix the location of the 8 numbered tiles plus the blank one

Initial state: any puzzle configuration

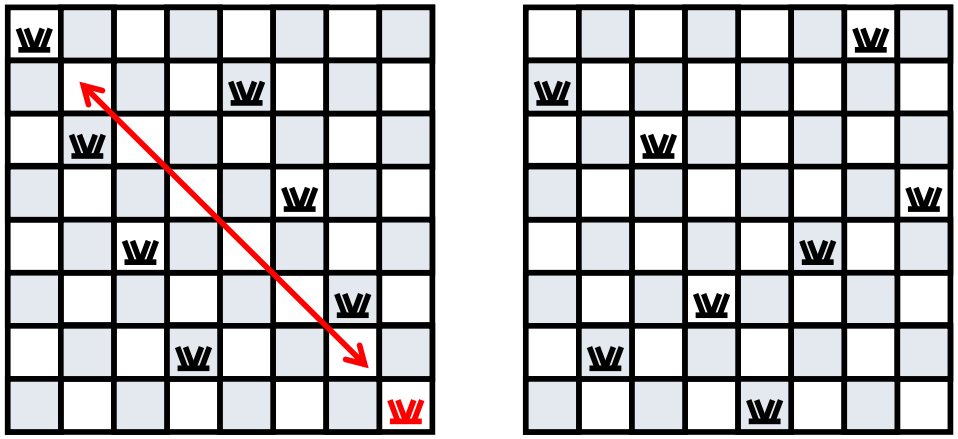
Operators: move “blank” to the left, right, up and down

Goal test: checks whether the state matches the goal configuration

Path cost: each step costs 1 (so path cost = number of steps)

Search problem examples

8-queens



almost solution

solution

States: location of the eight queens in a chess board

Initial state: empty board

Operators: add a queen to any empty square

Goal test: eight queens are on the board, none attacked

In this case, the number of possible states is 64x63x62x61x60x59x58x57

64^8 = 281 474 976 710 656

TÉCNICO LISBOA

Search problem examples

8-queens: alternative formulation

States: all possible arrangements of the queens, one per column, with no queen attacking another

Operators: add a queen to any square in the leftmost empty column such that it is not attacked by any other queen

In this case, the number of possible states is 2057 (check this out!)

Search problem examples

Cryptography

	T	R	E	S
+	D	O	I	S
+			U	M
M	E	I	I

	3	5	2	1
+	4	6	0	1
+			7	8
8	2	0	0

States: a set of letters, some of them associated with a number

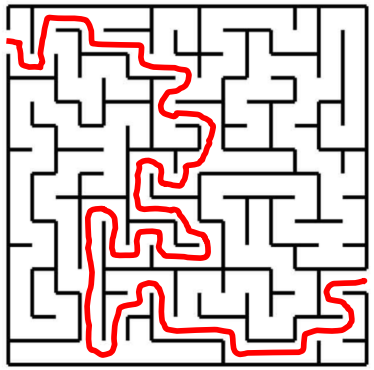
Operators: substitute all occurrences of one letter by an unused number

Goal test: all letters were substituted and the numbers satisfy the arithmetic operation

TÉCNICO LISBOA

Search problem examples

Mazes



States: maze configuration plus current location

Initial state: any maze location

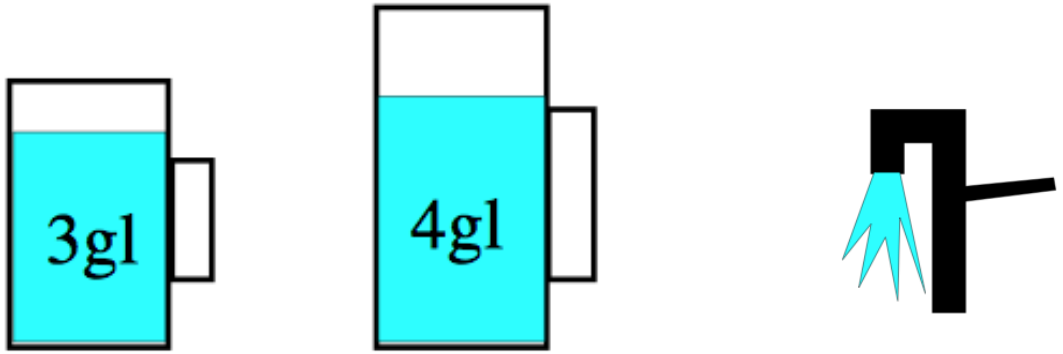
Operators: move to an adjacent and linked position

Goal test: current location = maze exit

Path cost: number of steps

Search problem examples

Water jugs



States: amount of water in each jug (e.g., <1,4>)

Initial state: <0,0>

Operators: fill J4; fill J3;
empty J4; empty J3;
pour water from J4 into J3 until J3 is full or J4 is empty;
pour water from J3 into J4 until J4 is full or J3 is empty;

Goal test: <0,2>

Path cost: amount of water used

TÉCNICO LISBOA

Search problem examples

Hanoi towers

States: location of the disks in the three poles

Initial state: all disks correctly arrange in the left pole

Operators: move a free disk to another pole, which is empty or all disks in it are bigger

Goal test: all disks correctly arrange in the right pole

Path cost: number of disk moves

Search problem examples

Missionaries and cannibals

States: location of the missionaries, cannibals and the boat

Initial state: all on one side of the river

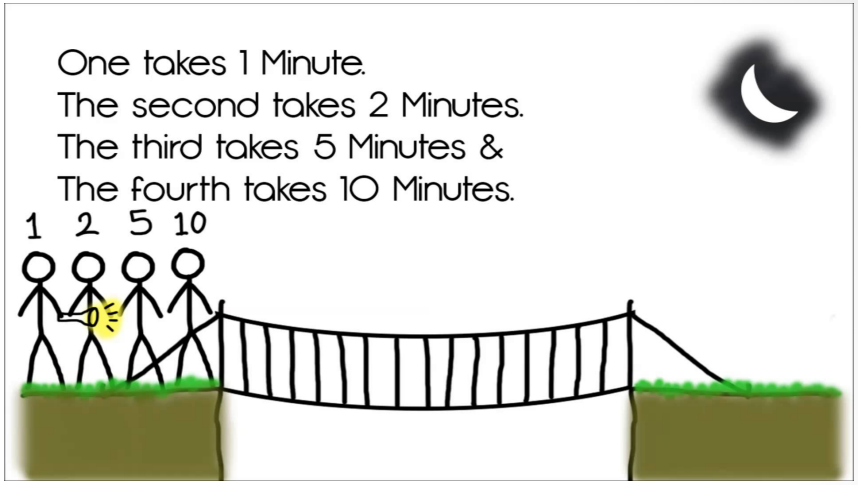
Operators: move one or two persons to the other side

Goal test: all on the other side of the river

Path cost: number of boat moves

Search problem examples

4 Men and the Bridge



States: location of the men and the flashlight

Initial state: all on one side

Operators: move one or two men, with the flashlight, to the other side

Goal test: all on the other side

Path cost: number of minutes needed

Search problem examples

Travelling salesman problem

Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?

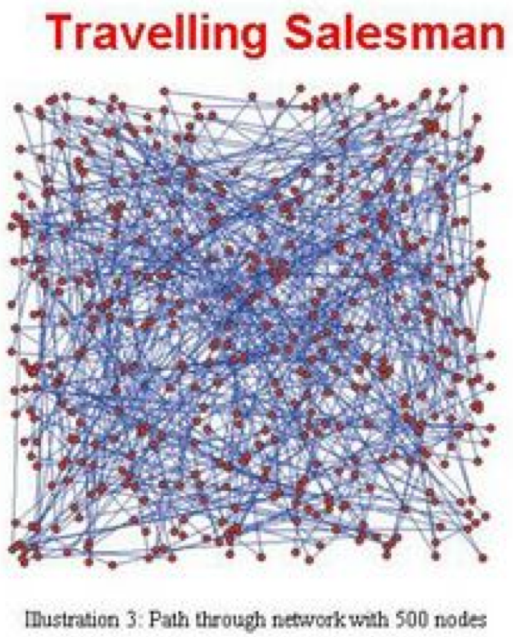
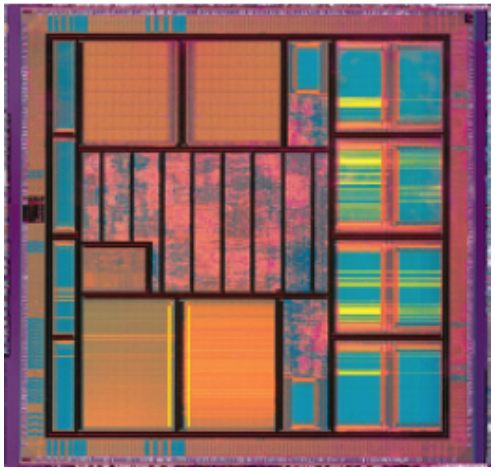


Illustration 3: Path through network with 500 nodes

TÉCNICO LISBOA

Search problem examples

VLSI layout

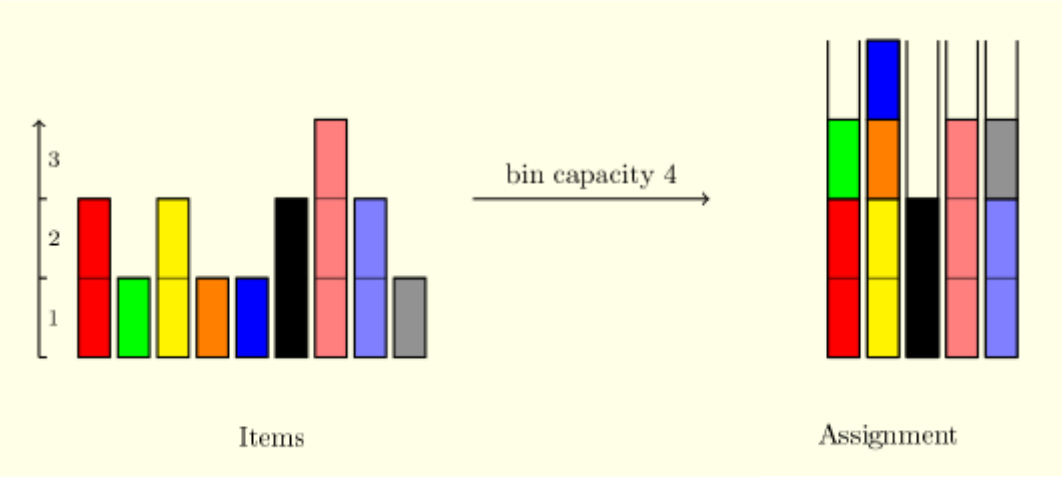


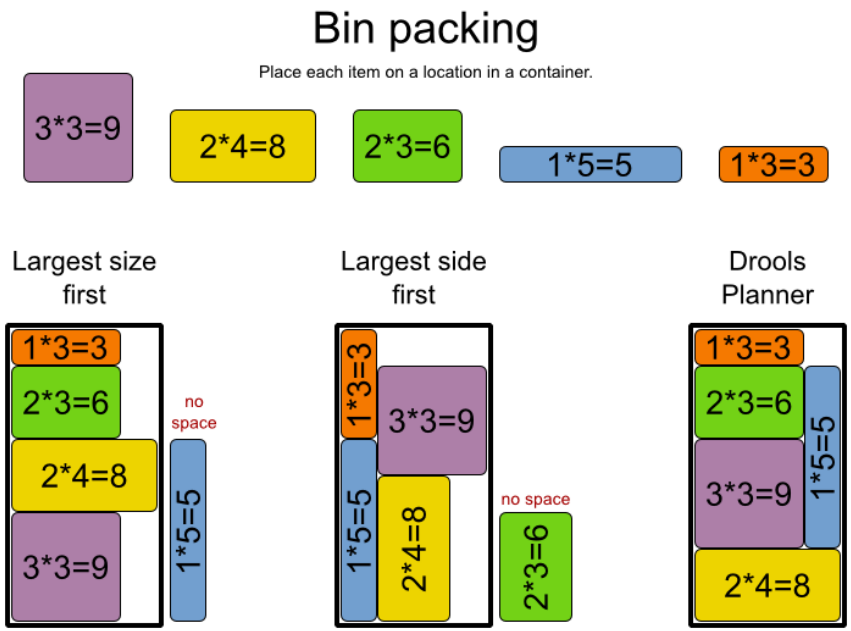
How to position millions of components and connections on a chip to minimize area, minimize circuit delays, minimize stray capacitances, and maximize manufacturing yield?

Also, the packing and cutting problems…

Search problem examples

Bin packing problem

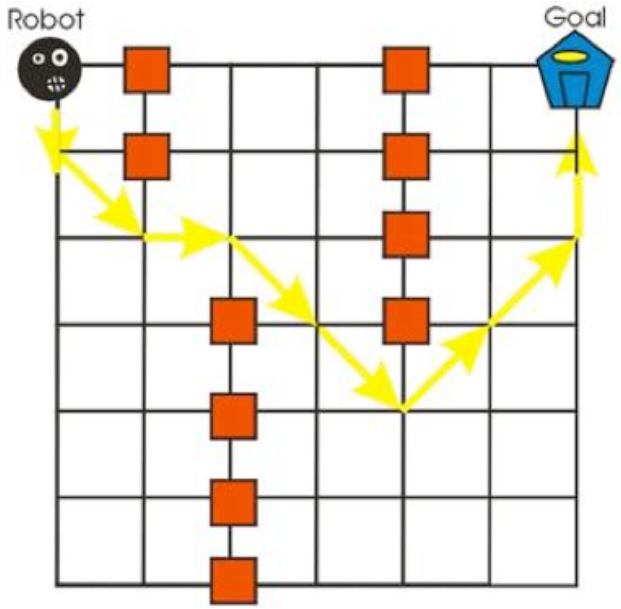


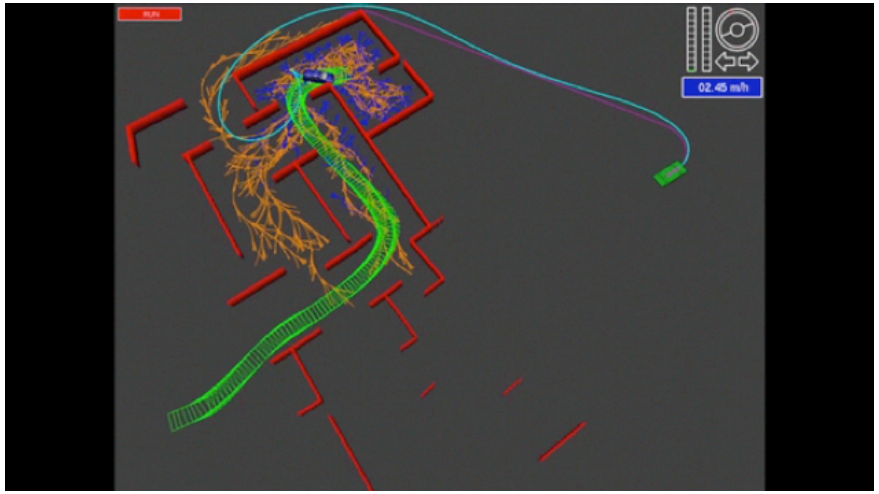


TÉCNICO LISBOA

Search problem examples

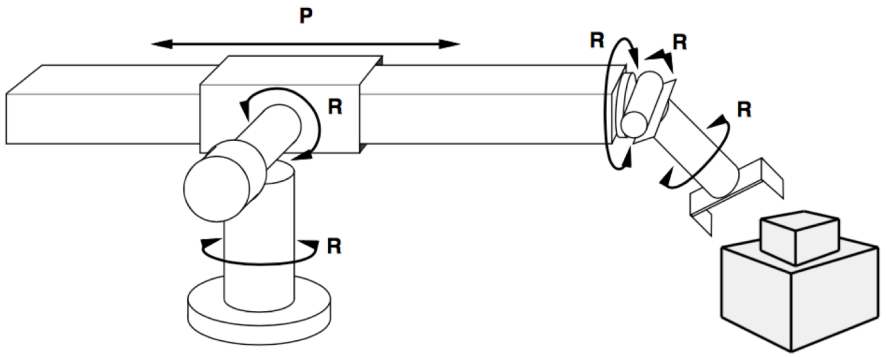
Robot navigation





Search problem examples

Robotic assembly



States: real-valued coordinates of robot joint angles and parts of the object to be assembled

Initial state: initial position of the robot and parts

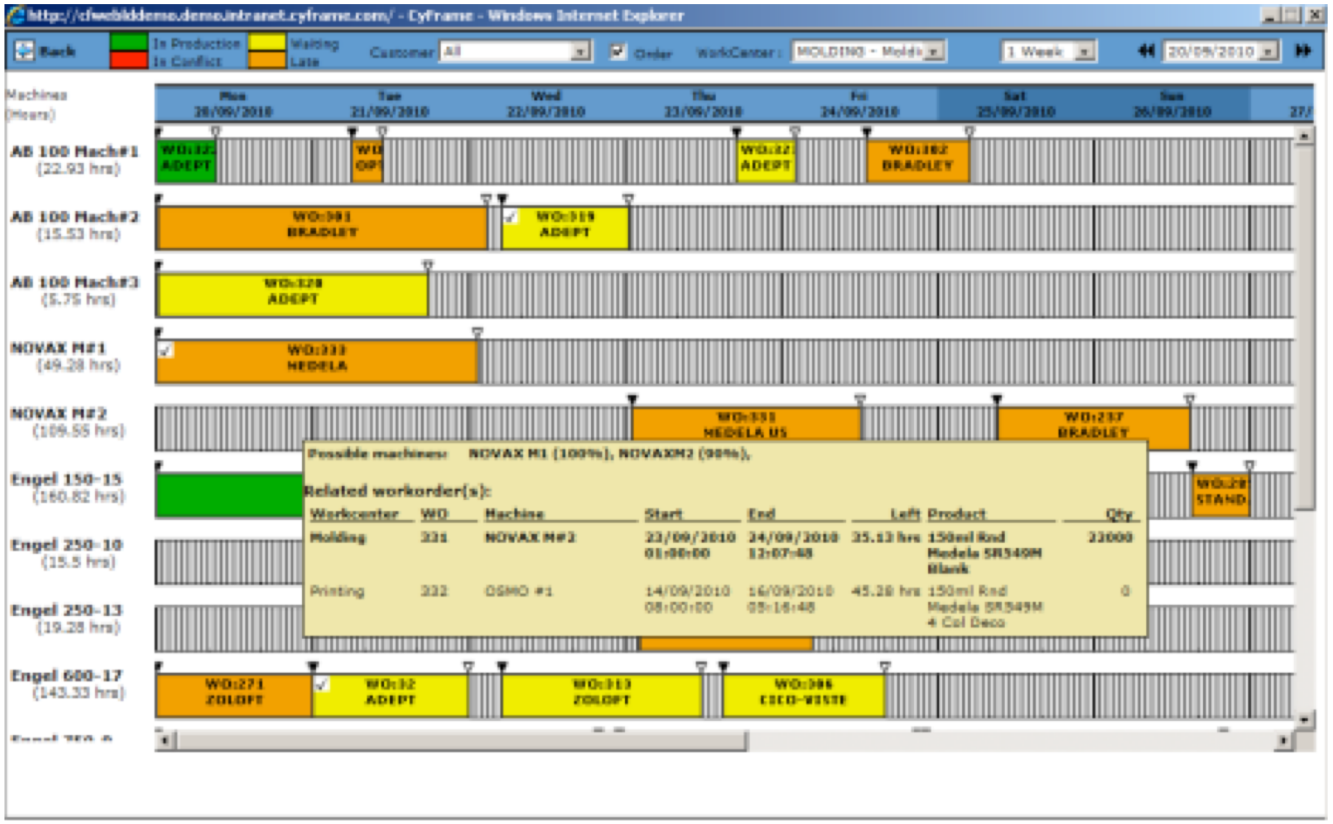
Operators: continuous motions of robot joints

Goal test: assembly complete?

Path cost: time to execute

Search problem examples

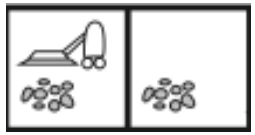
Manufacturing scheduling



Given n products to be manufactured, each has a setup time, processing time and a due date. To be completed, each product has to be processed at several machines. How to sequence these products on the machines in order to optimize a certain performance criterion?

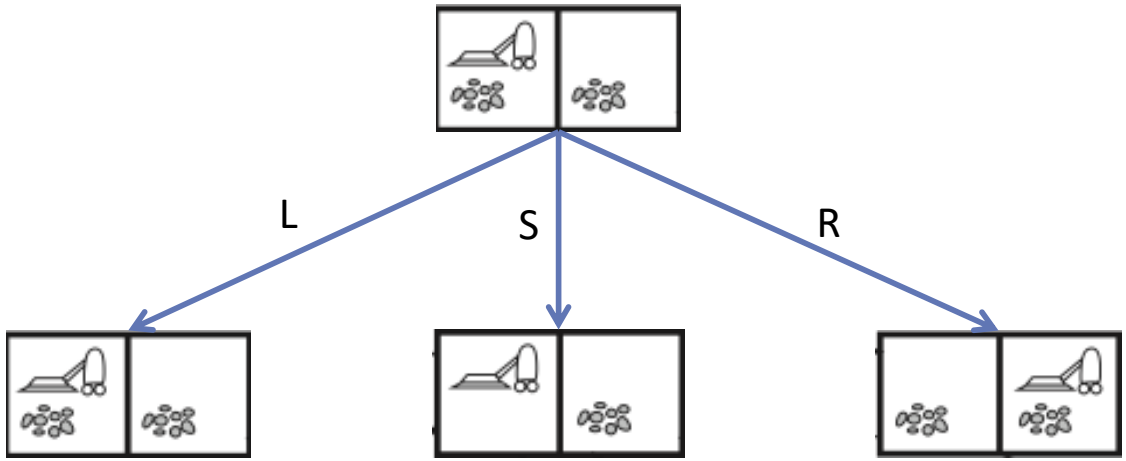
Search mechanism

Example: vacuum cleaner

Initial state 

Steps:

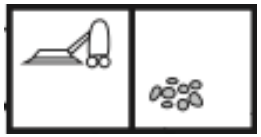
- Verify if it is a goal state no
- Use operators to generate successor states of the current state



Search mechanism

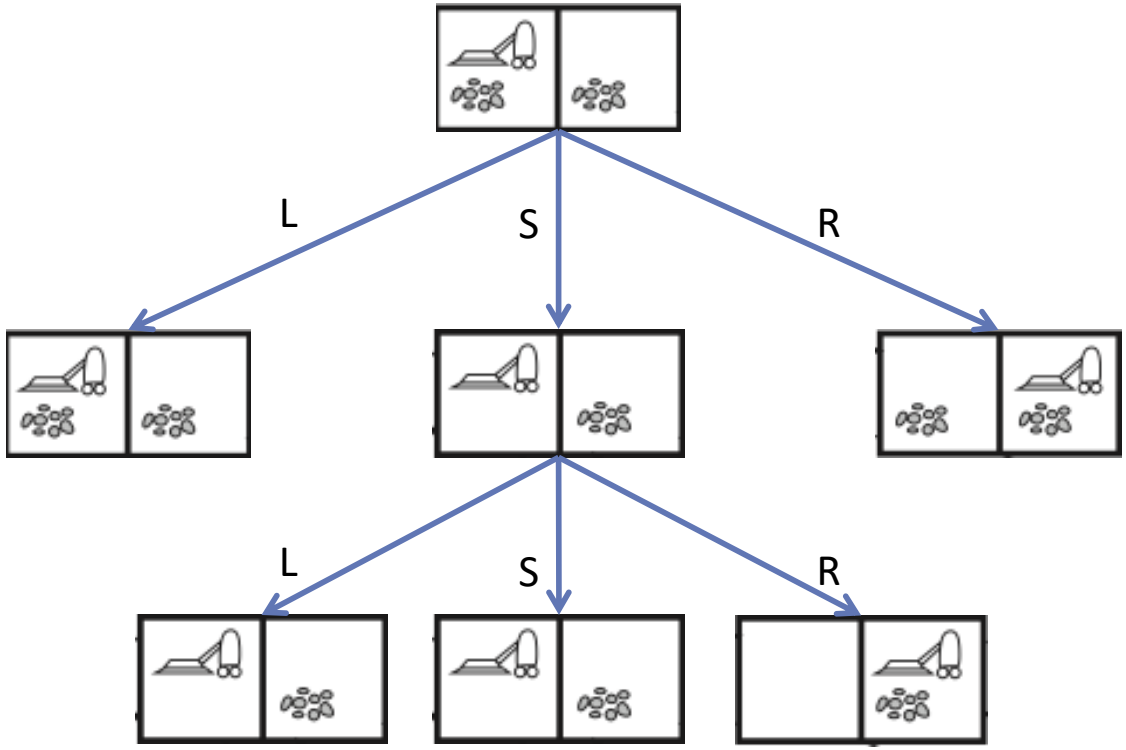
Example: vacuum cleaner

- Choose next state to explore



- Verify if it is a goal state no

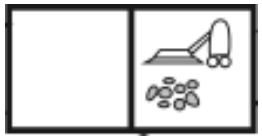
- Use operators to generate successor states of the current state



Search mechanism

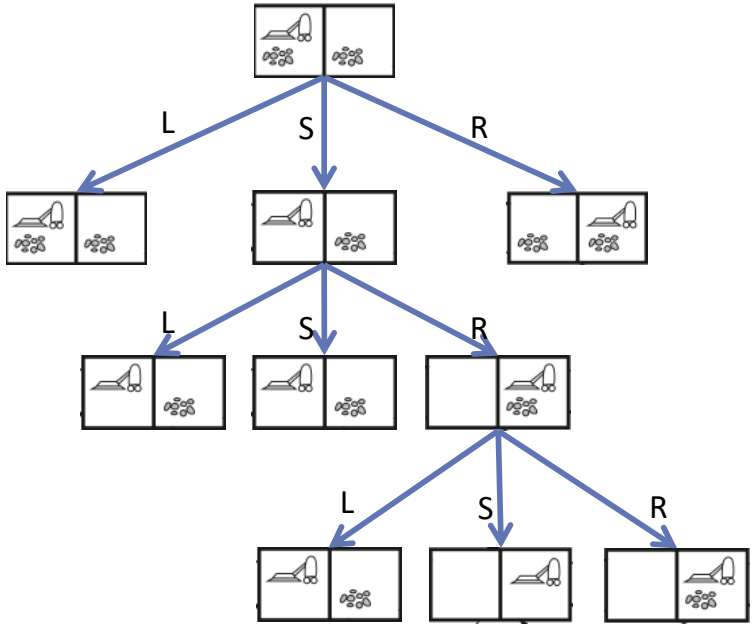
Example: vacuum cleaner

- Choose next state to explore



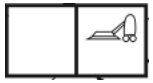
- Verify if it is a goal state no

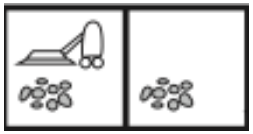
- Use operators to generate successor states of the current state



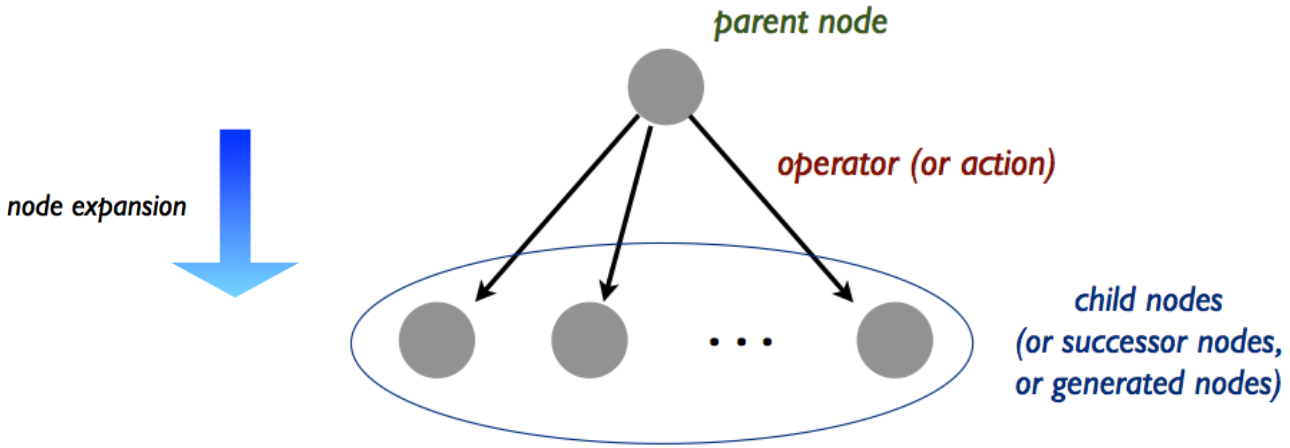
Search mechanism

Example: vacuum cleaner

- Choose next state to explore 
- Verify if it is a goal state yes

Solution = {S, R, S} for the initial state: 

Search terminology



Successor function: given a node, returns the set of child nodes

Open list (or frontier or fringe): set of nodes not yet expanded

Closed (explored) list: set of nodes already expanded

Leaf node: a node without successors

State space can be:

- Tree-based – no repeated nodes in search
- Graph-based – directed cycle graph

General tree search algorithm

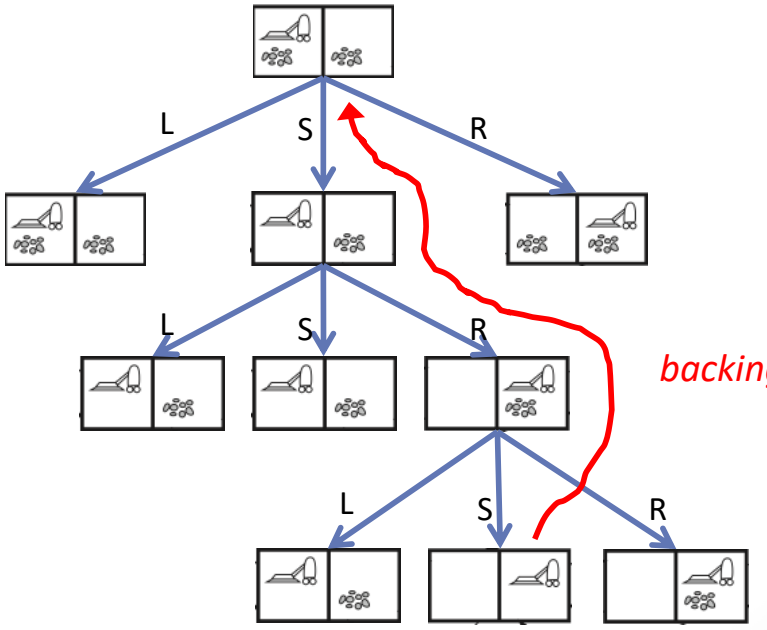
Start with open_list = { initial_node }

Algorithm:

iterate

1. select a node from open_list

2. check whether it is a goal node (i.e., satisfies goal test)
   if affirmative, return *solution by backing up to the root*



backing up to the root: S,R,S

General tree search algorithm

Start with open_list = { initial_node }

Algorithm:

iterate

1. select a node from open_list

2. check whether it is a goal node (i.e., satisfies goal test)
   if affirmative, return solution by backing up to the root

3. otherwise,
   remove it from open_list,
   expand it using the successor function, and
   insert successor nodes into open_list

Choosing different selection criteria (search **strategy)** spans a variety of search methods

TÉCNICO LISBOA

General tree search algorithm

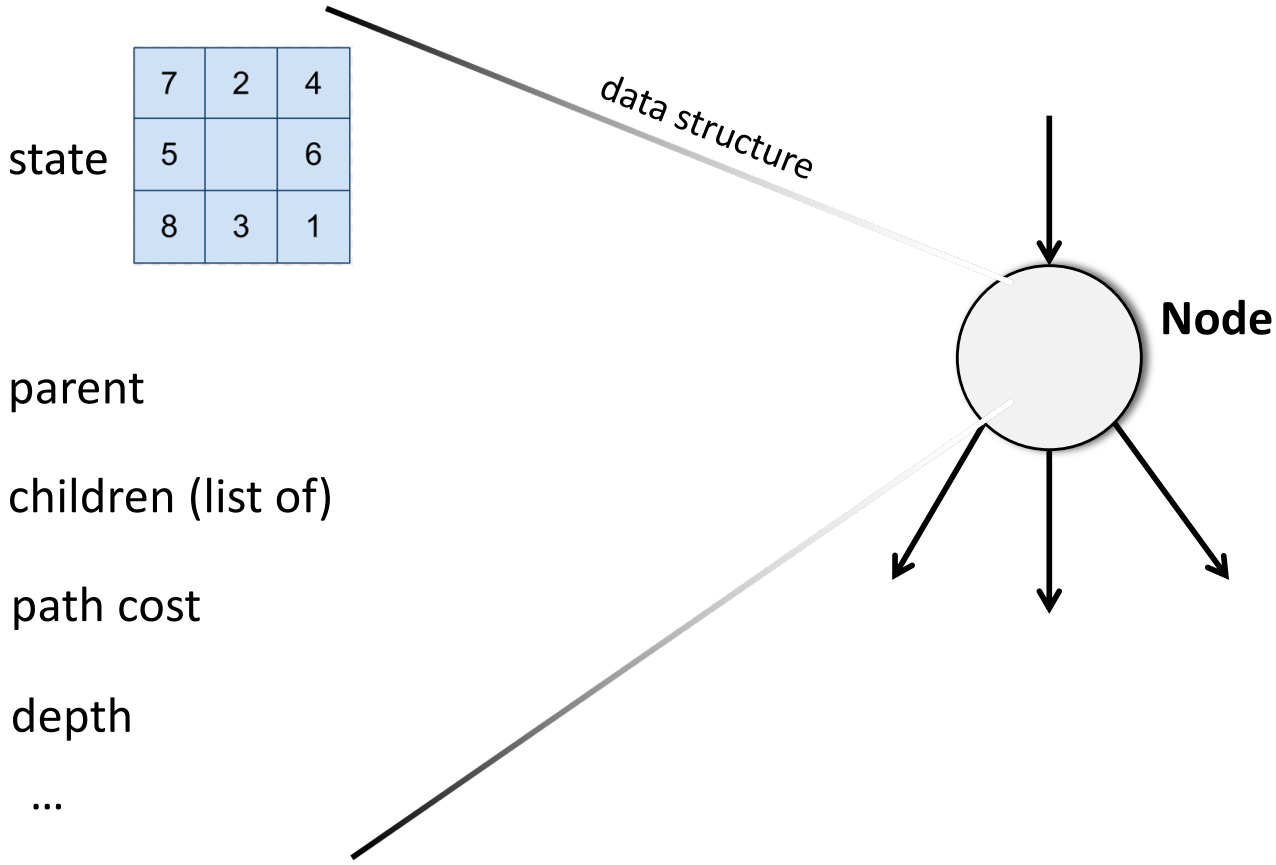
function Tree-search (*problem*, *strategy*) **returns** a solution or failure
    insert the root node into the open list
        (the root node contains the initial state of problem)
    loop do
        if there are no candidate nodes for expansion **then return** failure
        choose a node for expansion according to strategy
        if the node contains a goal state **then**
            return the corresponding solution
        else
            expand the node and add the resulting nodes to the open list
     end

tree search – repeated nodes are inserted in the open list

States vs. nodes

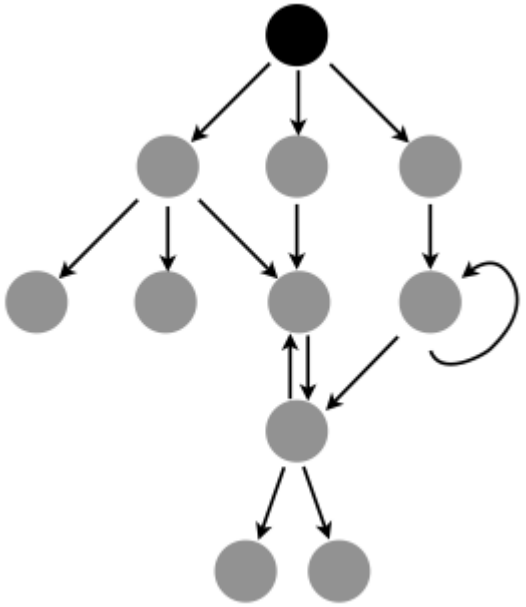
state – representation of a physical configuration of the problem (environment)

node – data structure constituting part of a search tree (or graph)



General graph search algorithm

In case of graph search, all explored nodes are stored in the **closed (explored) list** and all newly generated repeated nodes that match previously generated nodes can be discarded instead of being added to the frontier



```
function GRAPH-SEARCH(problem) returns a solution, or failure
  initialize the frontier using the initial state of problem
  initialize the explored set to be empty
  loop do
      if the frontier is empty then return failure
      choose a leaf node and remove it from the frontier
      if the node contains a goal state then return the corresponding solution
      add the node to the explored set
      expand the chosen node, adding the resulting nodes to the frontier
         only if not in the frontier or explored set
```

graph search – repeated nodes are not inserted in the open list (frontier)

TÉCNICO LISBOA

More terminology

Completeness: guarantee that a solution is found if there is one

Optimality: the solution found minimizes path cost over all possible solutions

Time complexity: how long does it take to find a solution (usually measured in terms of the number of nodes generated)

Space complexity: how much memory is needed to find a solution (usually measured in terms of the maximum number of nodes stored in memory)

Branching factor (b): maximum number of successors of any node

Depth (d) of the shallowest goal node

m = maximum length of any path in state space (may be infinite)

g(n): the cost of going from the root node to node n (path cost function)

TÉCNICO LISBOA

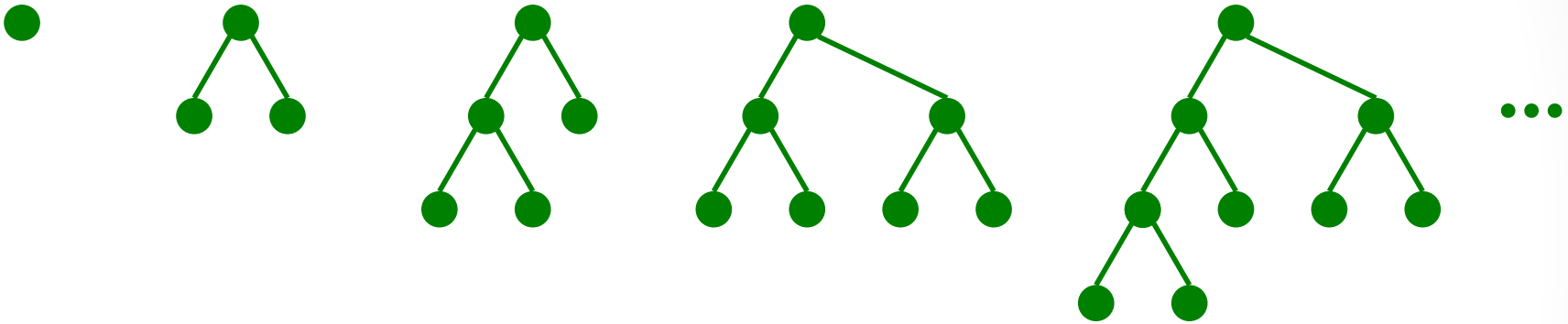
Classes of search strategies

Uninformed (or blind) search: does not use additional (domain-dependent) information about states beyond that provided in the problem definition

Informed (or heuristic) search: uses problem-specific knowledge about the domain to "guide" the search towards more promising paths

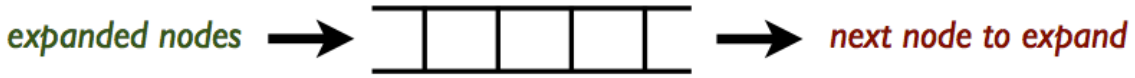
Uniformed search strategies

Breadth-first search – all nodes are expanded at a given depth (level) before any nodes at the next level



Strategy: select earliest expanded node first

can be easily implemented using a FIFO queue of open nodes



Uniformed search strategies

Breadth-first search – all nodes are expanded at a given depth before any nodes at the next level

Properties:

- Complete
- Optimal, if path cost is a non-decreasing function of depth

Let **d** be the depth of the solution and **b** the branching factor, then

if goal test is applied once each node is generated, in the worst case the total number of nodes generated is $1 + b + b^2 + b^3 + \cdots + b^d$

- time complexity: $O(b^d)$
- space complexity: $O(b^d)$

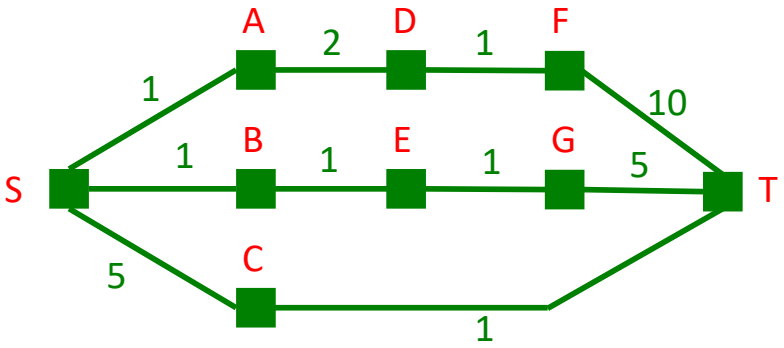
Depth	Nodes	Time	Memory
2	110	.11 milliseconds	107 kilobytes
4	11,110	11 milliseconds	10.6 megabytes
6	10^6	1.1 seconds	1 gigabyte
8	10^8	2 minutes	103 gigabytes
10	10^{10}	3 hours	10 terabytes
12	10^{12}	13 days	1 petabyte
14	10^{14}	3.5 years	99 petabytes
16	10^{16}	350 years	10 exabytes

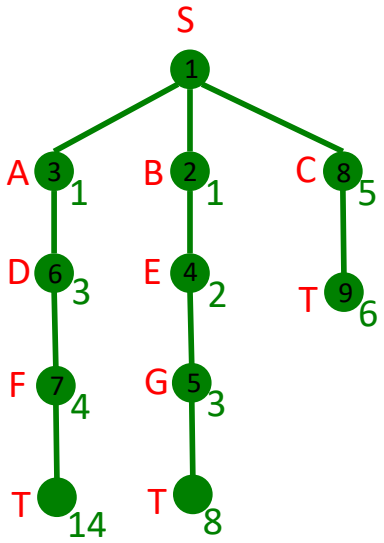
Figure 3.13 Time and memory requirements for breadth-first search. The numbers shown assume branching factor $b = 10$; 1 milion nodes/second; 1000 bytes/node.

what if goal test is applied when node is selected for expansion? $O(b^{d+1})$

Uniformed search strategies

Uniform-cost search – select node with lower path cost





Green numbers: path cost from node S

Black numbers: expansion order

Optimal solution

Strategy: select node with lower path cost

can be easily implemented using a priority queue of open nodes

Uniformed search strategies

General graph search algorithm – adapted to Uniform-cost search

```
function GENERAL GRAPH SEARCH (problem) returns a solution, or failure

  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  frontier ← a priority queue ordered by PATH-COST, with node as the only element
  explored ← an empty set
  loop do
      if EMPTY?(frontier) then return failure
      node ← POP(frontier)  /* chooses the lowest-cost node in frontier */
      if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
      add node.STATE to explored
      for each action in problem.ACTIONS(node.STATE) do
          child ← CHILD-NODE(problem, node, action)
          if child.STATE is not in explored or frontier then
              frontier ← INSERT(child, frontier)
          else if child.STATE is in frontier with higher PATH-COST then
              replace that frontier node with child
```

new

Uniformed search strategies

Uniform-cost search – select node with lower path cost

Properties:

- Complete, if step costs are strictly positive
- Optimal, if step costs are strictly positive, because

$$\forall_n \quad g(sucessor(n)) > g(n)$$

- time and space complexity: proportional to number of nodes with path cost less than of optimal solution

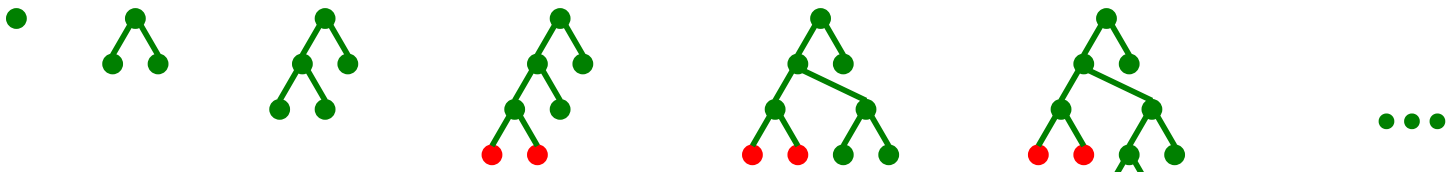
Mathematically, is

$$O\left(b^{1+\left\lfloor C^*/\varepsilon \right\rfloor}\right)$$

where C* is the cost of the optimal solution and ε is a lower bound to the step cost

Uniformed search strategies

Depth-first search – select latest expanded node first



Red nodes are leaf nodes

Can be easily implemented using a LIFO (=stack) queue of open nodes

Properties:

- Not complete, unless m is finite (and using graph search)
- Not optimal

- Time complexity: $O(b^m)$
- Space complexity: $O(b \times m)$

TÉCNICO LISBOA

Uniformed search strategies

Depth-first search – select latest expanded node first

Notes:

- if there are many solutions, the probability of finding one as soon as it reaches level d is greater; then depth-first search tends to be faster in this situation
- when the solutions are at different levels of depth, or the tree is very large or even infinite, depth-first search can be very inefficient

TÉCNICO LISBOA

Uniformed search strategies

Backtrack search – an improvement of depth-first

Same search strategy, but

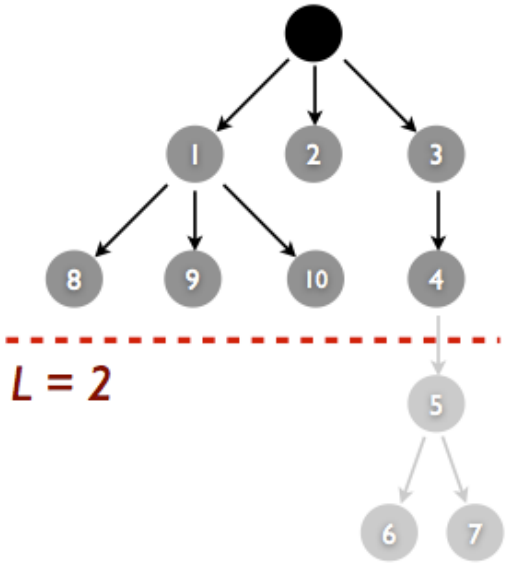
- expand one node at a time → space complexity reduces to *O(m)*
- store a single node in memory → space complexity reduces to *O(1)*
- expand by modifying node, backtrack by undoing modification

Not complete, not optimal

Uniformed search strategies

Depth-limited search – another improvement of depth-first

Strategy: limit depth of search tree to a limit *L*



Properties:

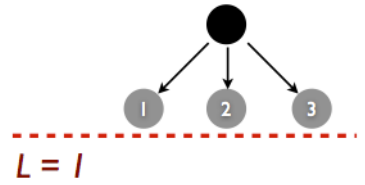
- Complete, if d ≤ L
- Not optimal

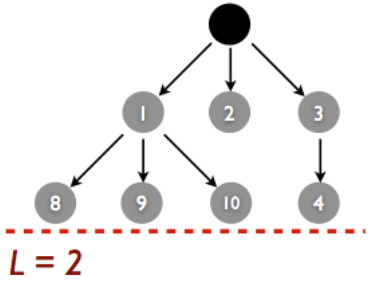
- Time complexity: $O(b^L)$
- Space complexity: O(b × L)

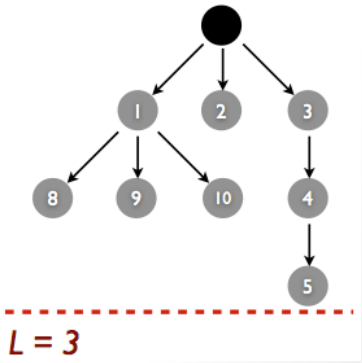
Uniformed search strategies

Iterative deepening depth-first search – improvement of depth-limited search

Strategy: run depth-limited search for *L=1, 2, ...* until a solution is found







Properties:

- Complete
- Optimal, if path cost is a non-decreasing function of depth

- Time complexity: $O(b^d)$
- Space complexity: O(bd)

Uniformed search strategies

Iterative deepening depth-first search – improvement of depth-limited search

Strategy: run depth-limited search for *L=1, 2, …* until a solution is found

Node expansion:

Breadth-first

$$1 + b + b^2 + b^3 + \cdots + b^{d-1} + b^d$$

Iterative deepening

$$(d+1)1 + (d)b + (d-1)b^2 + (d-2)b^3 + \cdots + (2)b^{d-1} + (1)b^d$$

Example: b = 10 and d = 5

breadth-first = 111 111

iterative deepening = 123 456 (+ 11%)

Uniformed search strategies

Bidirectional search – search both from initial node and from the goal node

Can only be used when
- a goal node is known
- parent node can be computed given its child and the corresponding action
  i.e., need a way to specify the predecessors of G

Properties:
- Complete, if breadth-first in both directions
- Optimal, if step costs are all equal

- Time and space complexity: $O(b^{d/2})$

Example: $b = 10$ and $d = 5$

$b^d = 1\ 111\ 111$

$2\ b^{d/2} = 2\ 222$

TÉCNICO LISBOA

Comparison of uninformed methods

Criterion	Breadth-first	Uniform-cost	Depth-first	Depth-limited	Iterative deepening	bidirectional
Complete?	✓	✓[1]	×	×	✓	✓[3]
Optimal?	✓[2]	✓	×	×	✓[2]	✓[2,3]
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\varepsilon \rfloor})$	$O(b^m)$	$O(b^L)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\varepsilon \rfloor})$	$O(b.m)$	$O(b.L)$	$O(b.d)$	$O(b^{d/2})$

[1] for strictly positive step costs
[2] for path costs a non-decreasing function of depth
[3] for breadth-first in both directions

General search algorithm

function General-search (***problem***, ***strategy***) **returns** a solution or failure
    insert the root node into the open list
        (the root node contains the initial state of problem)
    loop do
        if there are no candidate nodes for expansion **then return** failure
        choose a node for expansion according to strategy (using strategy function)
        if the node contains a goal state (using goal checking function) **then**
            return the corresponding solution
        else
            for each operator in the list of operators (or successor function)
                create a child node (for the new child state)
                update child node path cost (using g-function)
                add the resulting node to the open list [*unless… see graph search versions*]
    end

domain (problem) independent

problem argument should include at least:

- initial state (using a specific state representation)
- successor function: new state = succ (current state, operator)
- path cost function (g-function)

domain (problem) dependent

strategy argument (e.g., FIFO, LIFO, priority queue (by path cost), etc.)

algorithm selection