Artificial Intelligence

Scalab Group

Universidad Carlos III de Madrid

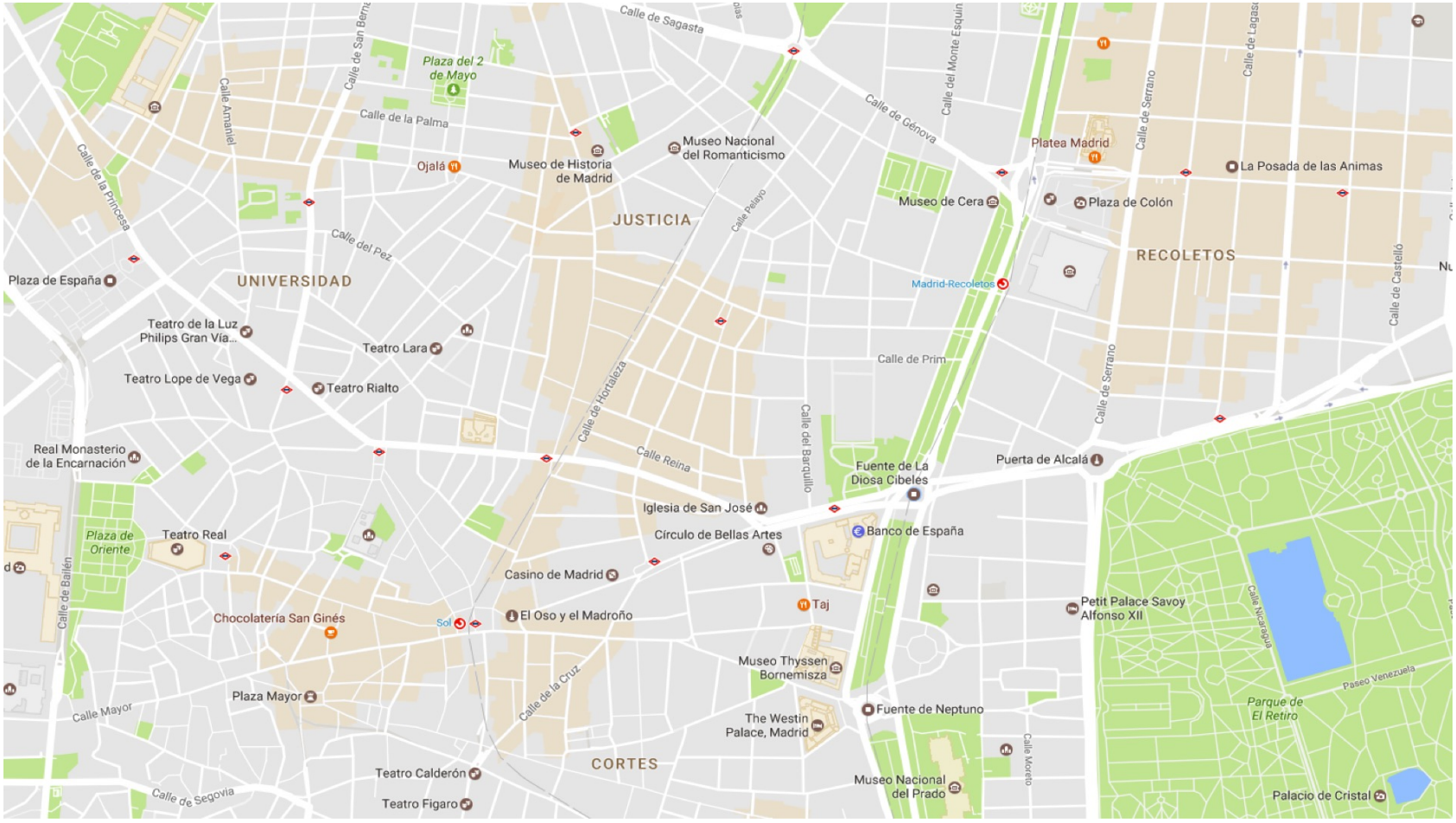
AI

uc3m

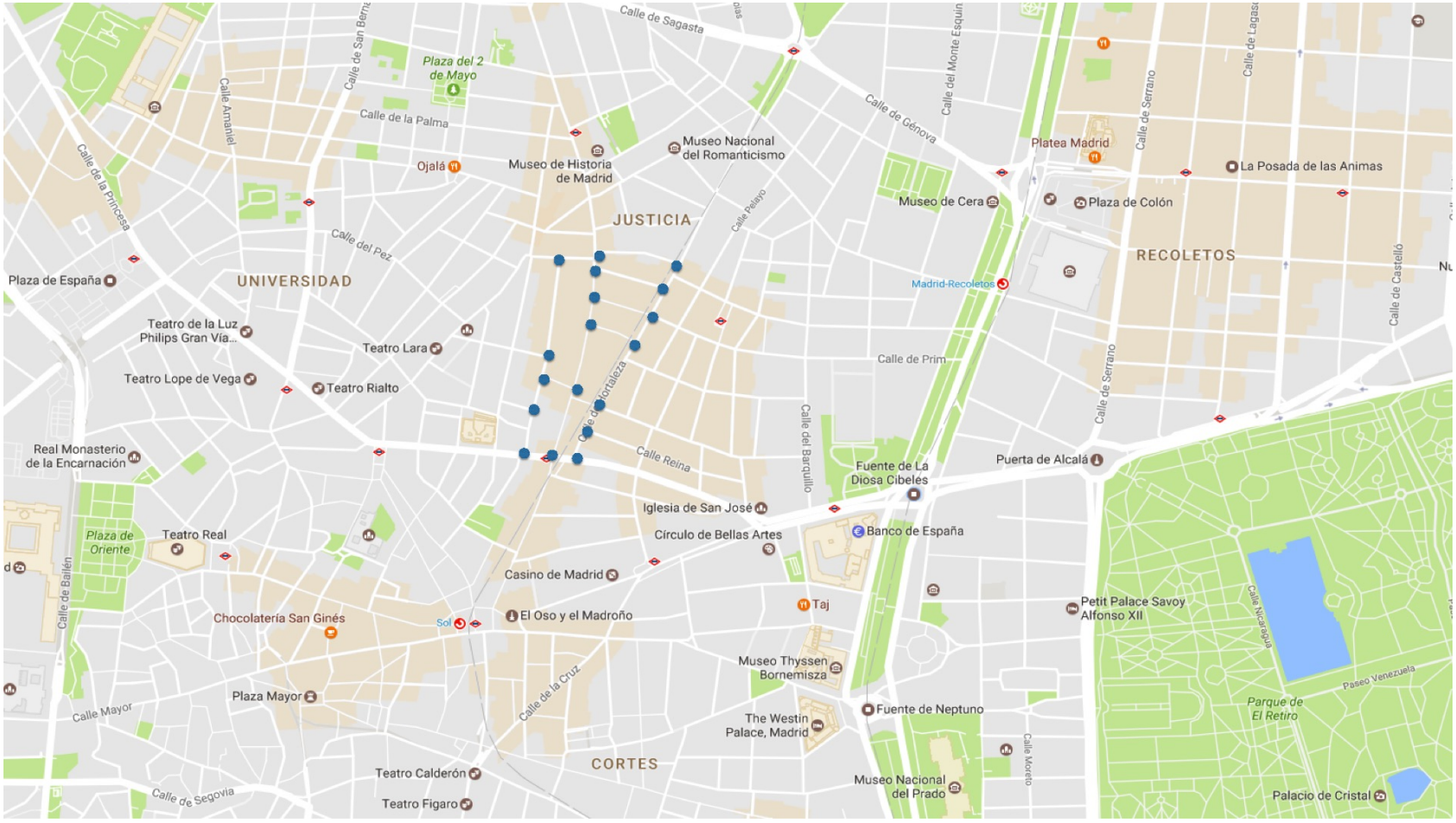
Outline

Outline

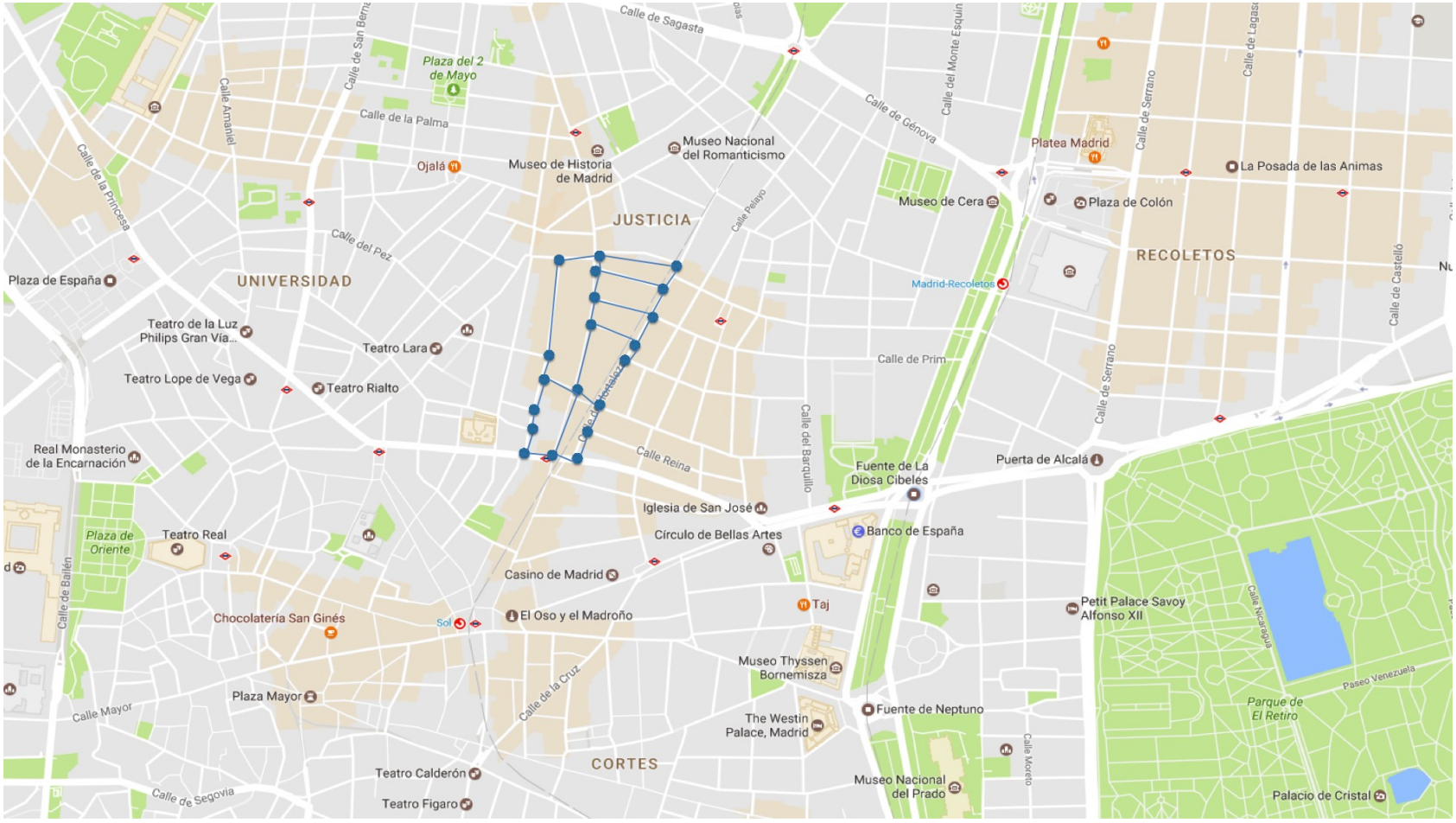
Problem Spaces



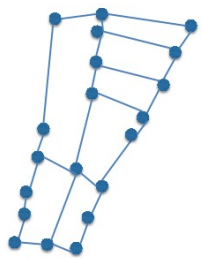
Problem Spaces



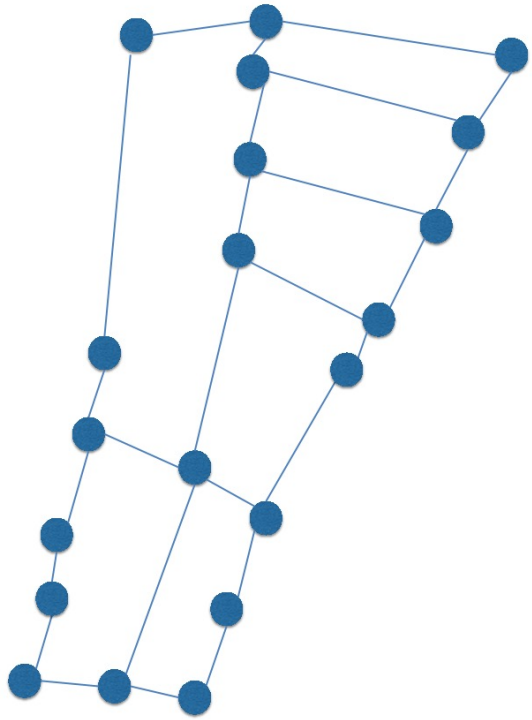
Problem Spaces



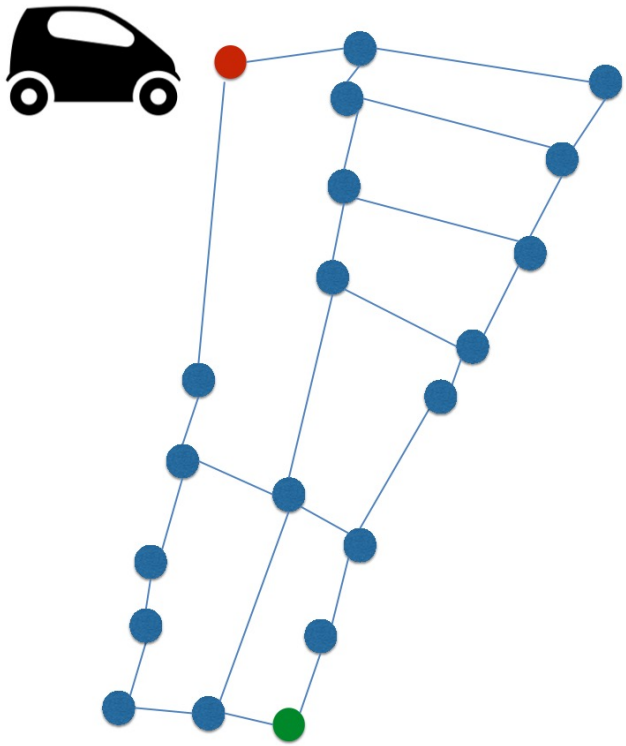
Problem Spaces



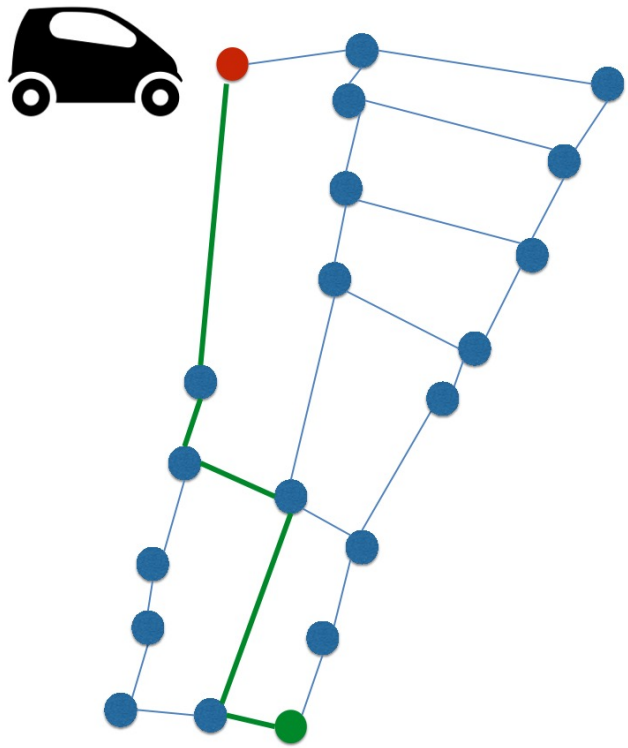
Problem Spaces



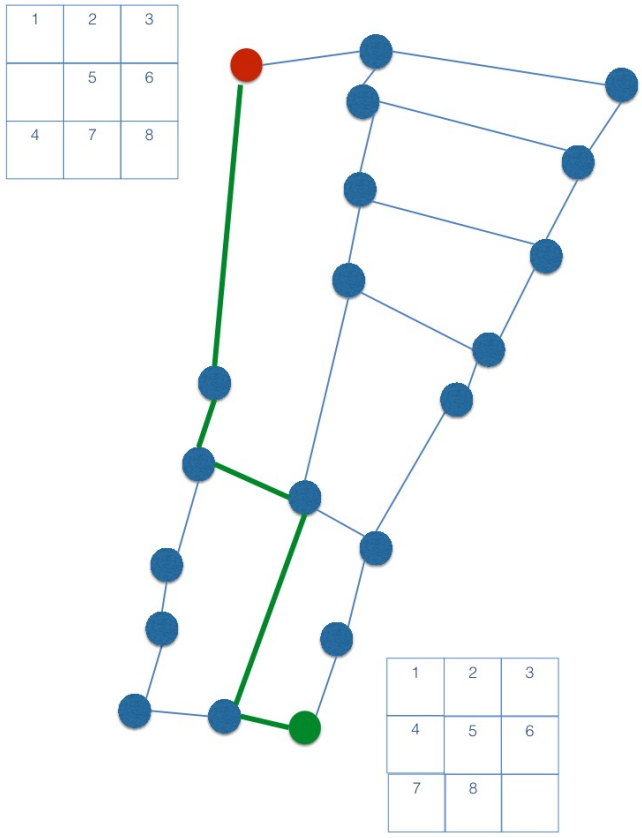
Problem Spaces



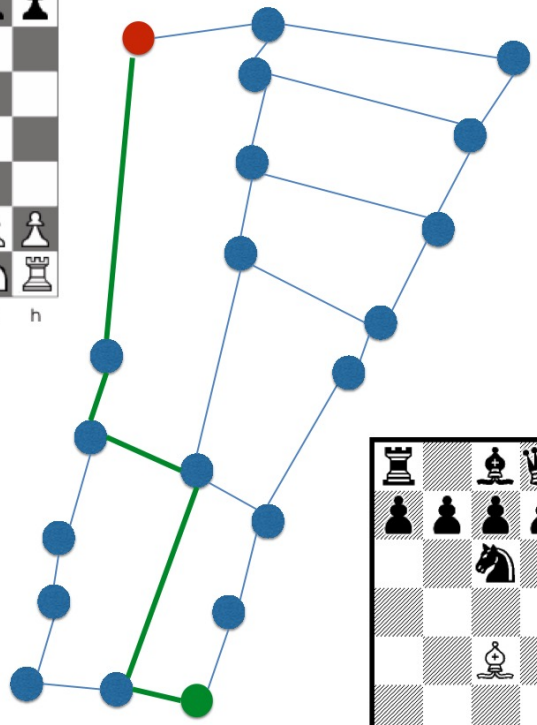
Problem Spaces

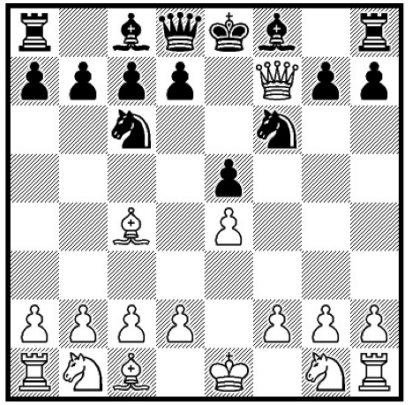


Problem Spaces

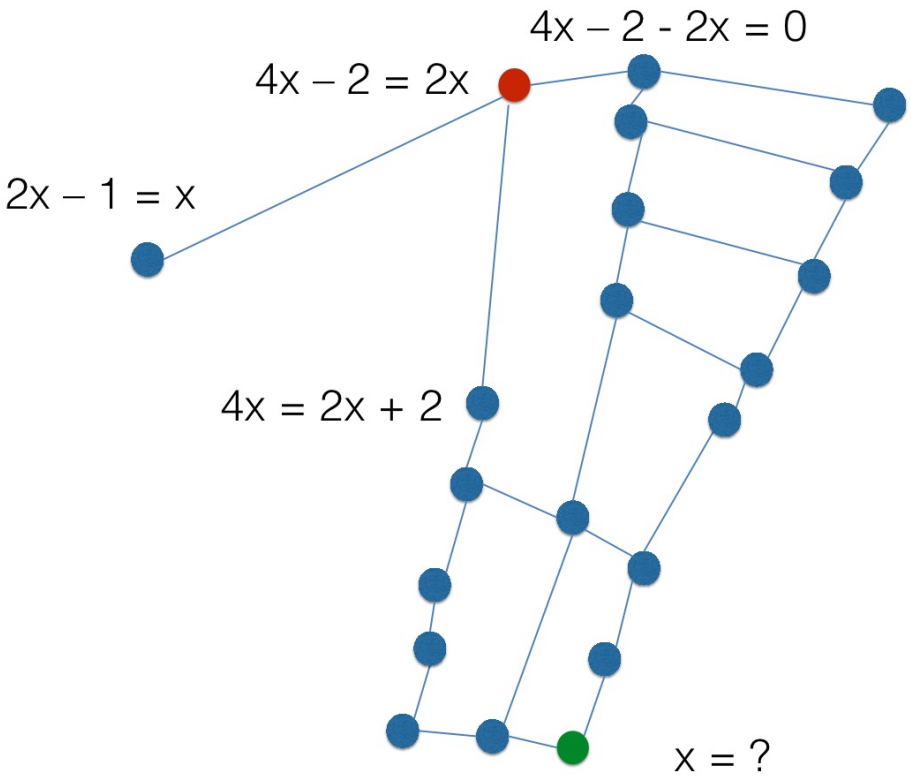


Problem Spaces

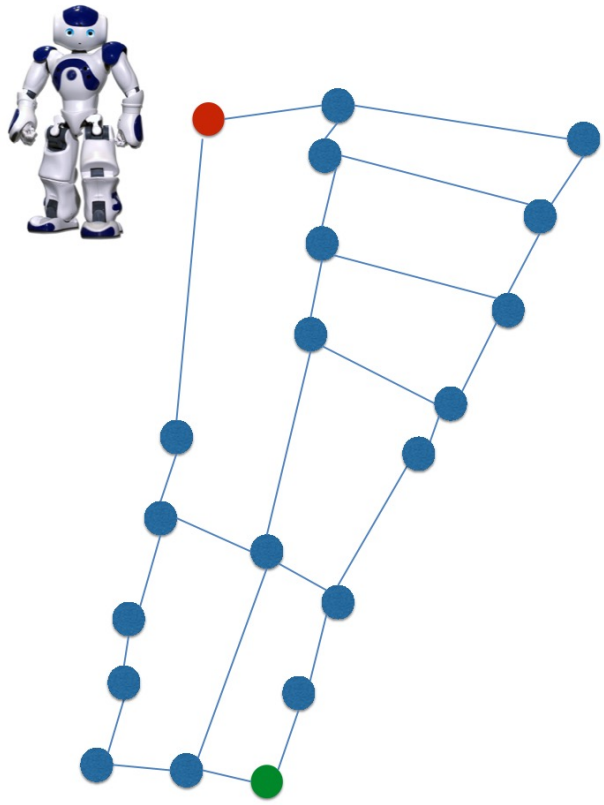




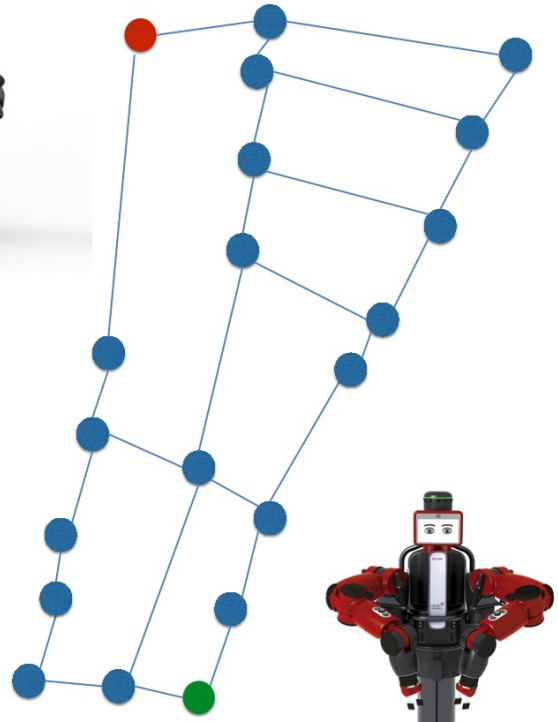
Problem Spaces



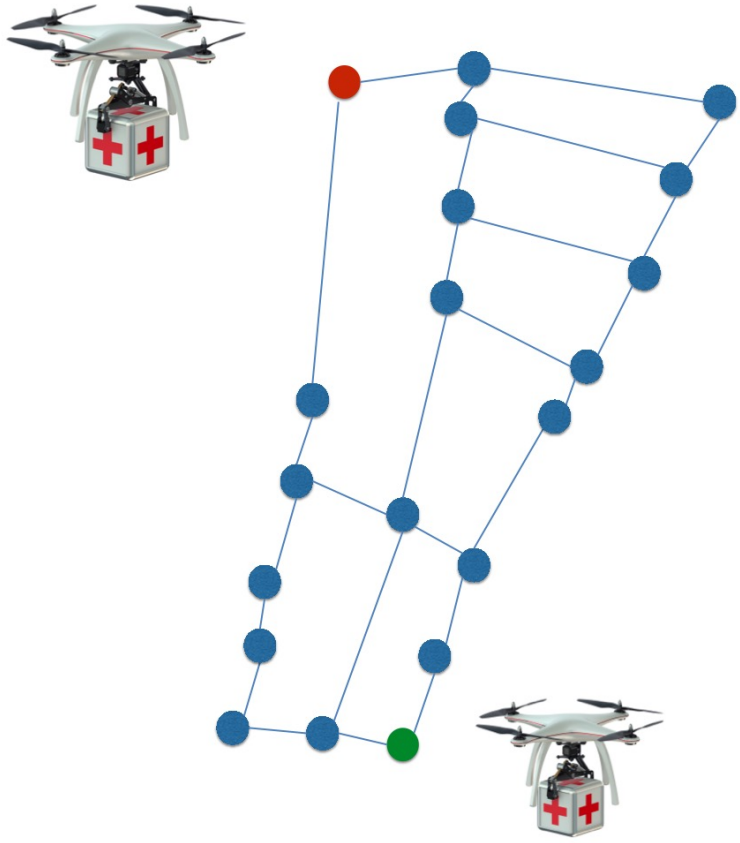
Problem Spaces



Problem Spaces

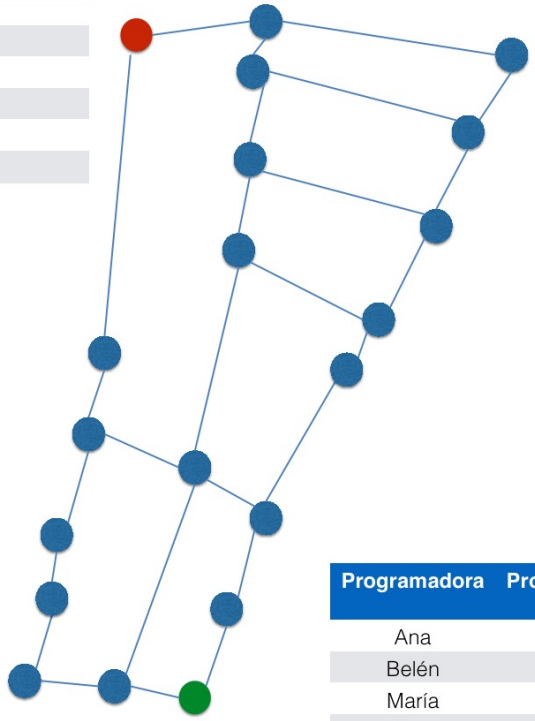


Problem Spaces



Problem Spaces

Programadora	Proyecto 1	Proyecto 2	...	Proyecto N
Ana				
Belén				
María				
Eva				
Magdalena				
Margarita				



Programadora	Proyecto 1	Proyecto 2	...	Proyecto N
Ana	?	?	?	?
Belén	?	?	?	?
María	?	?	?	?
Eva	?	?	?	?
Magdalena	?	?	?	?
Margarita	?	?	?	?

Problem Spaces

8	2	5	9

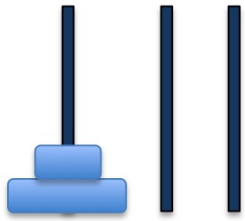
3x+4=5x

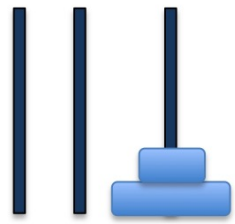
2	5	8	9

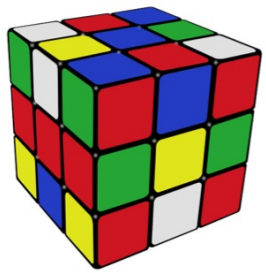
x=?

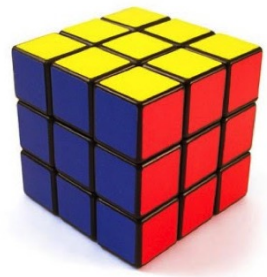
1	2	6
3	5	
4	7	8

1	2	3
4	5	6
7	8	

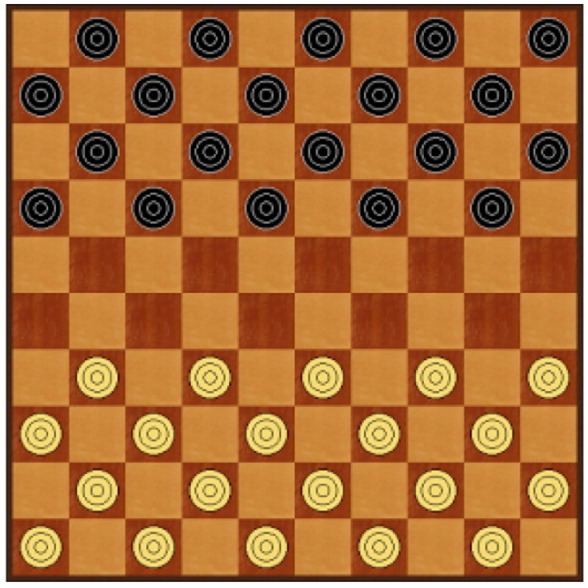


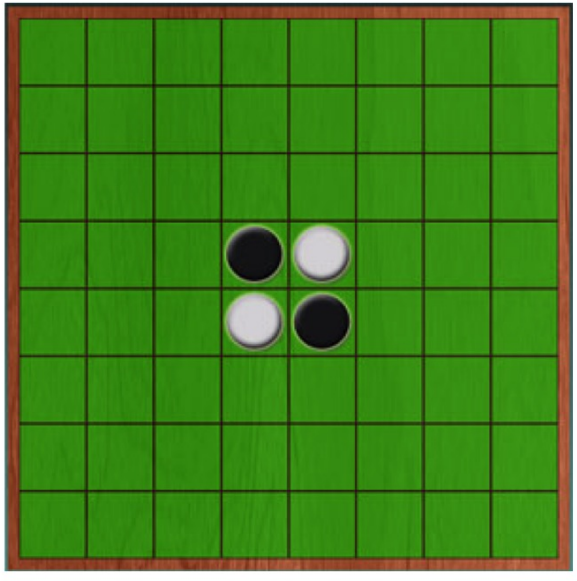


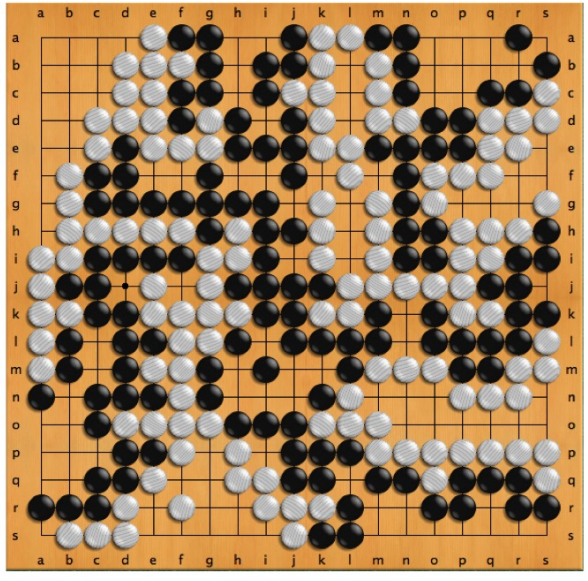




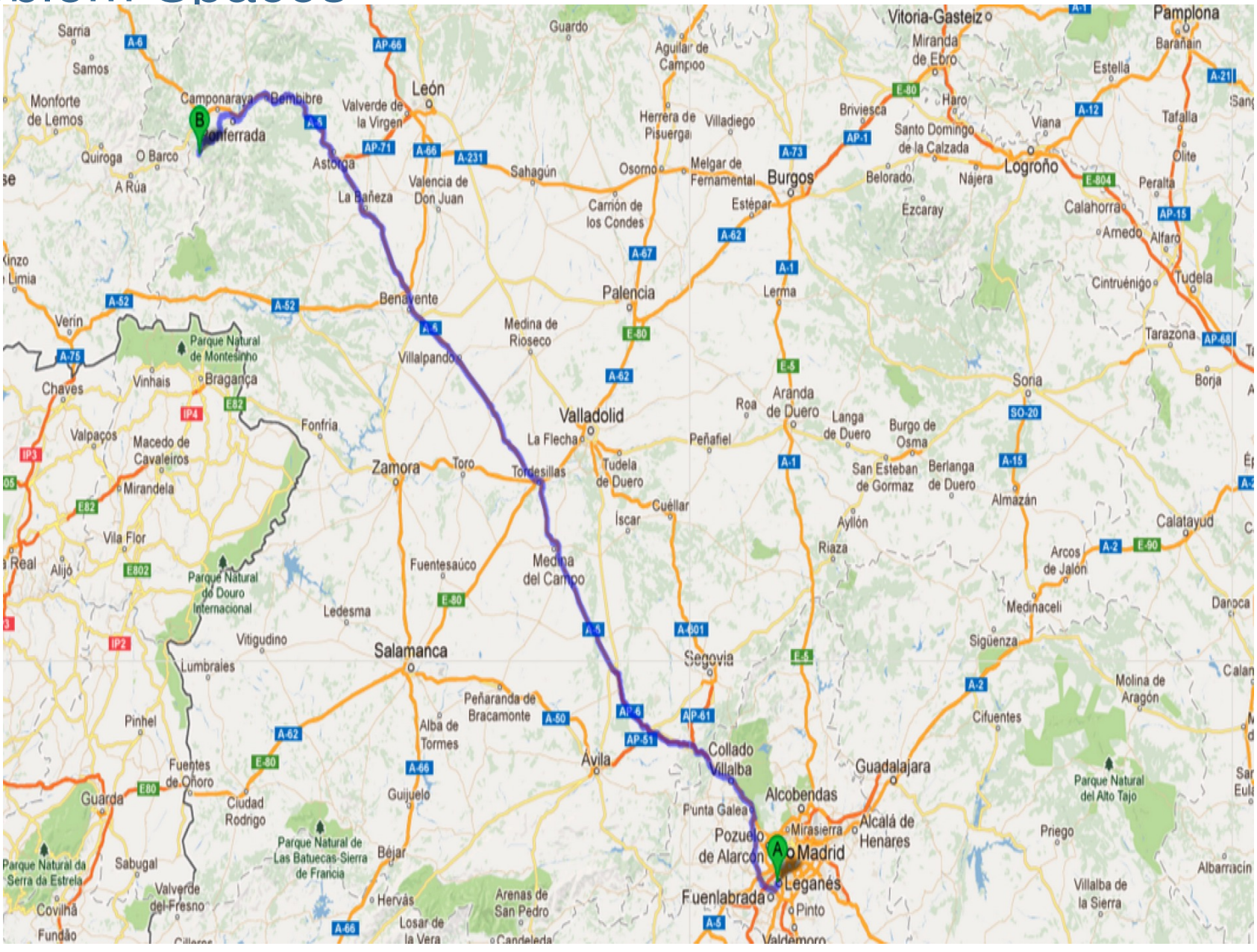
Problem Spaces



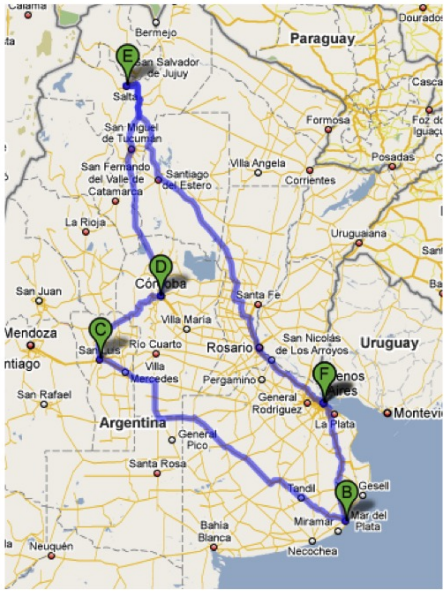


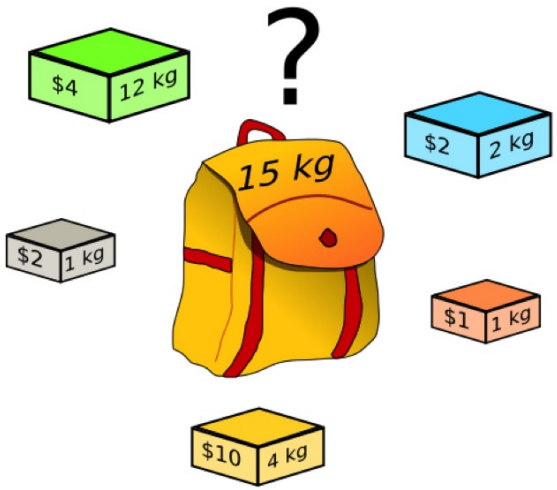


Problem Spaces

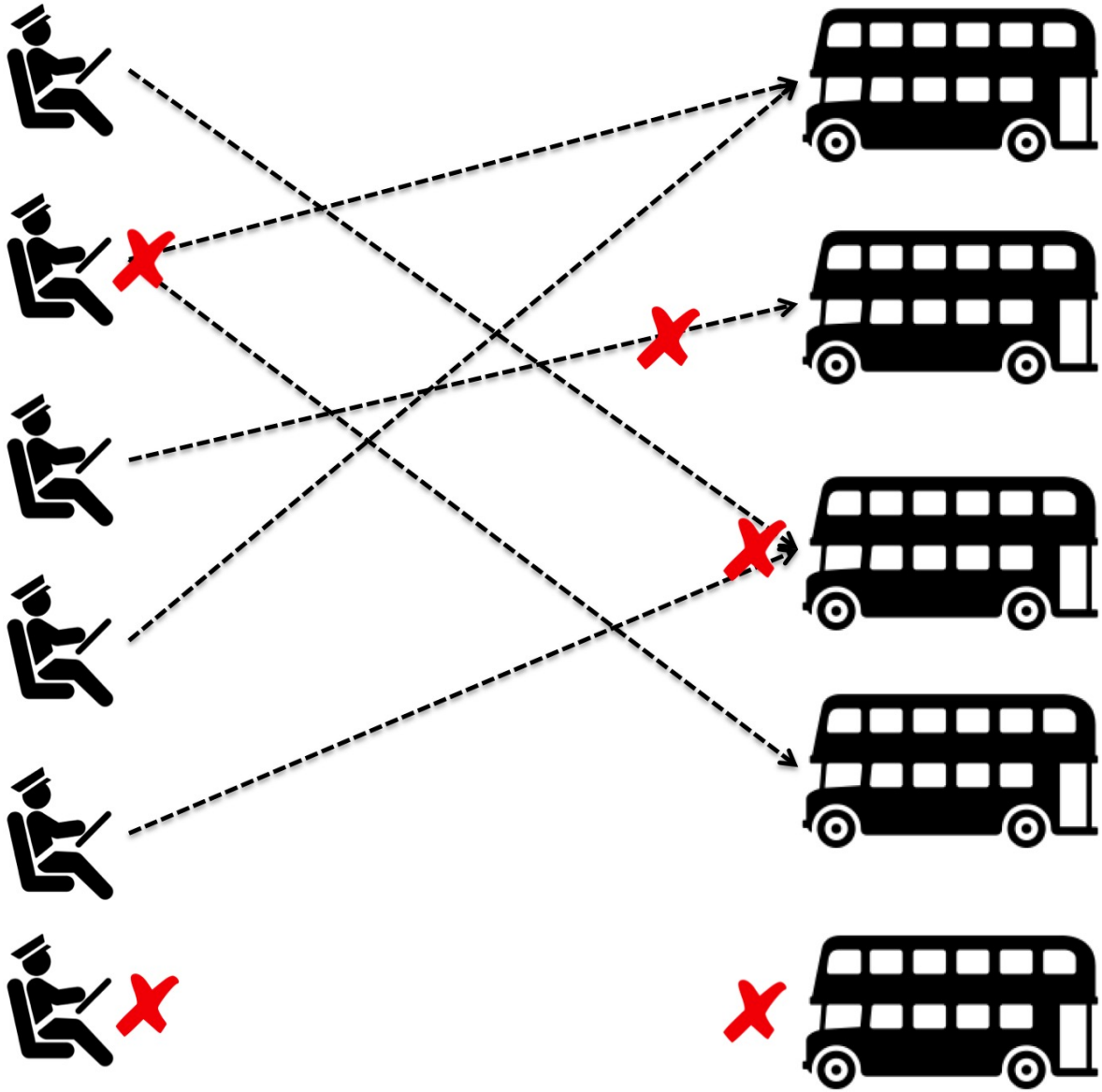


Problem Spaces

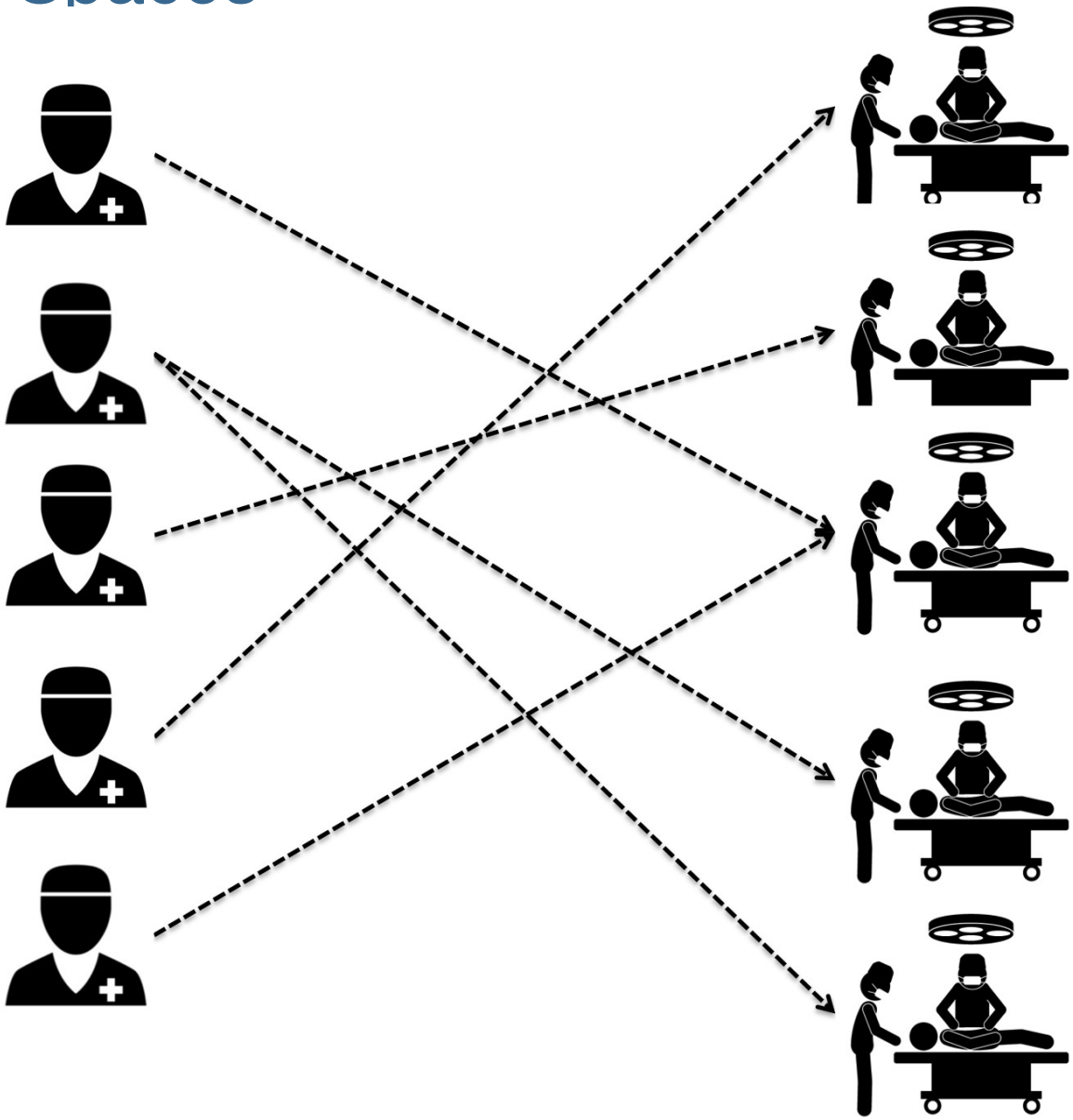




Problem Spaces



Problem Spaces

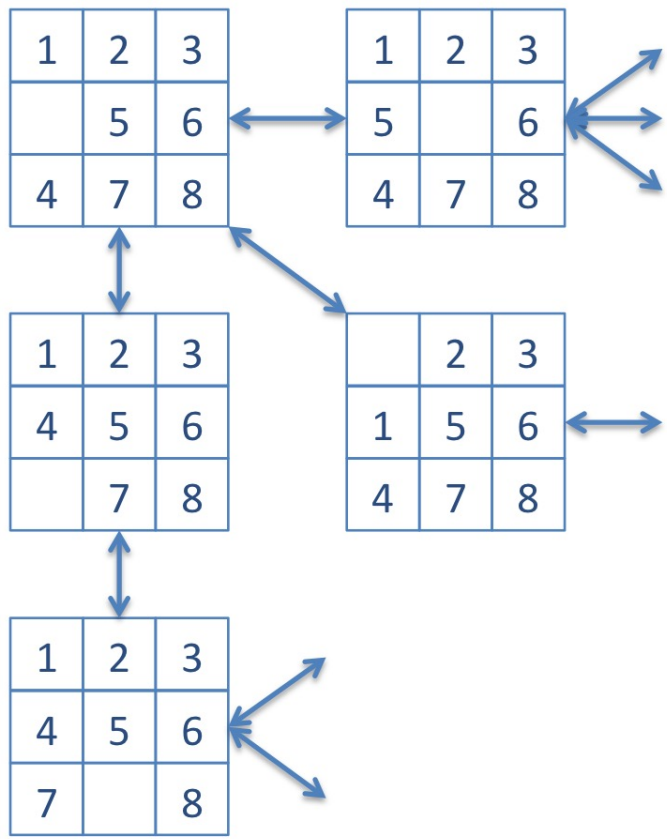


Introduction

- Problem space
  - a set of states
  - a set of operators

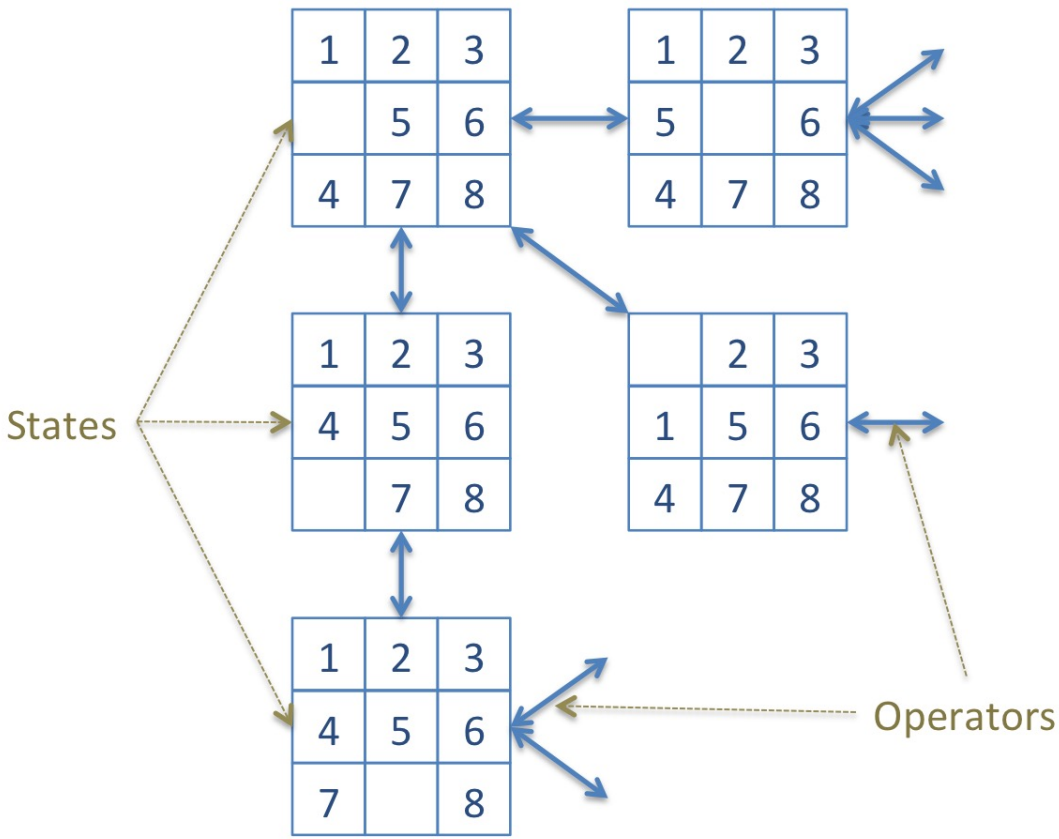
Introduction

- Problem space: 8-puzzle
  - a set of states
  - a set of operators



Introduction

- Problem space: 8-puzzle
  - a set of states
  - a set of operators



Introduction

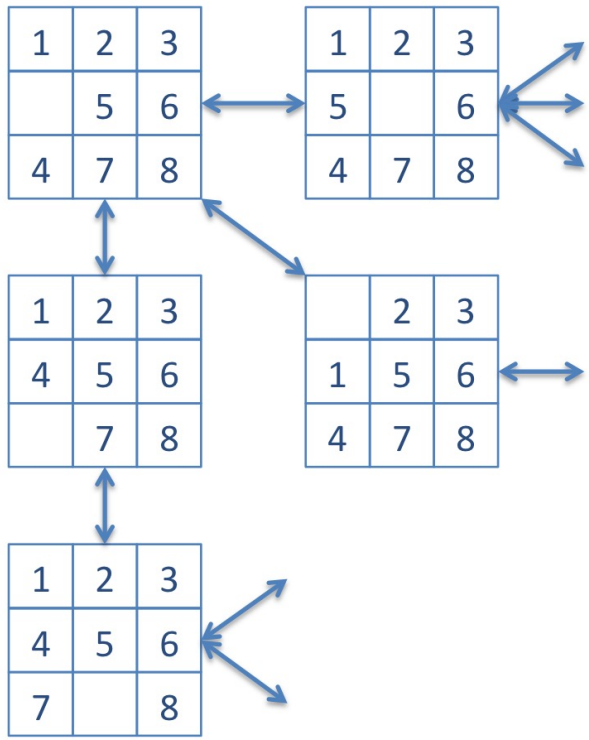
- Problem space
  - a set of states
  - a set of operators
- Problem
  - initial state (s)
  - goal (s) or final state(s)

1	2	3
	5	6
4	7	8

1	2	3
4	5	6
7	8	

Introduction

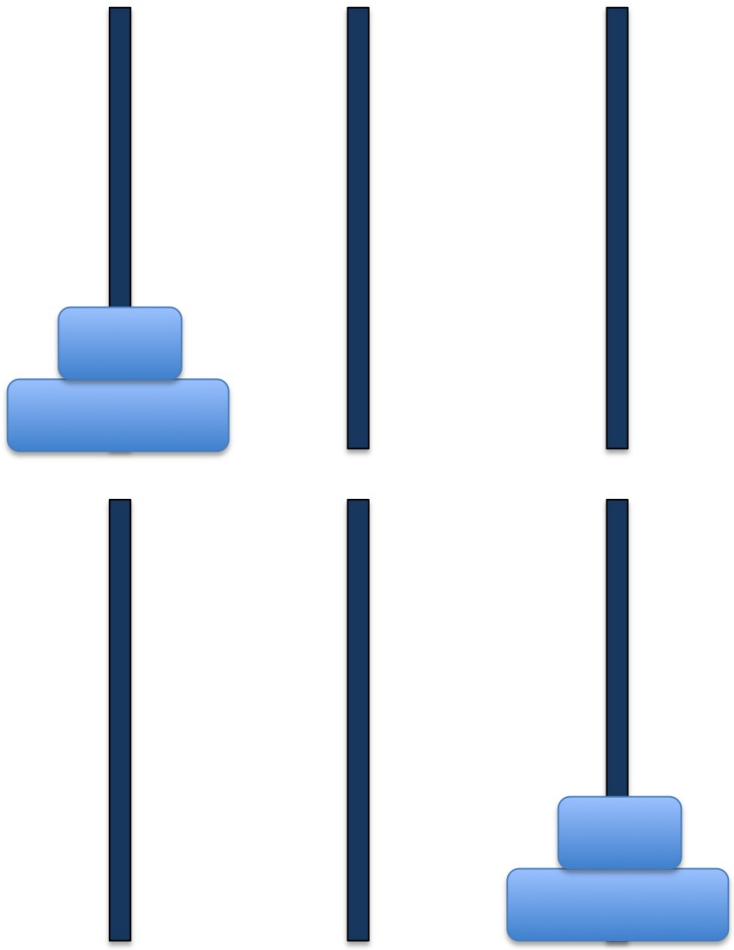
- Problem space
  - a set of states
  - a set of operators
- Problem
  - initial state (s)
  - goal (s) or final state(s)
- Graph representation



Introduction

- Problem space
  - a set of states
  - a set of operators
- Problem
  - initial state (s)
  - goal (s) or final state(s)
- Graph representation
- Problem solving = graph search
- Search generates a tree
- Parameters
  - branching factor, b
  - depth of the search tree, d

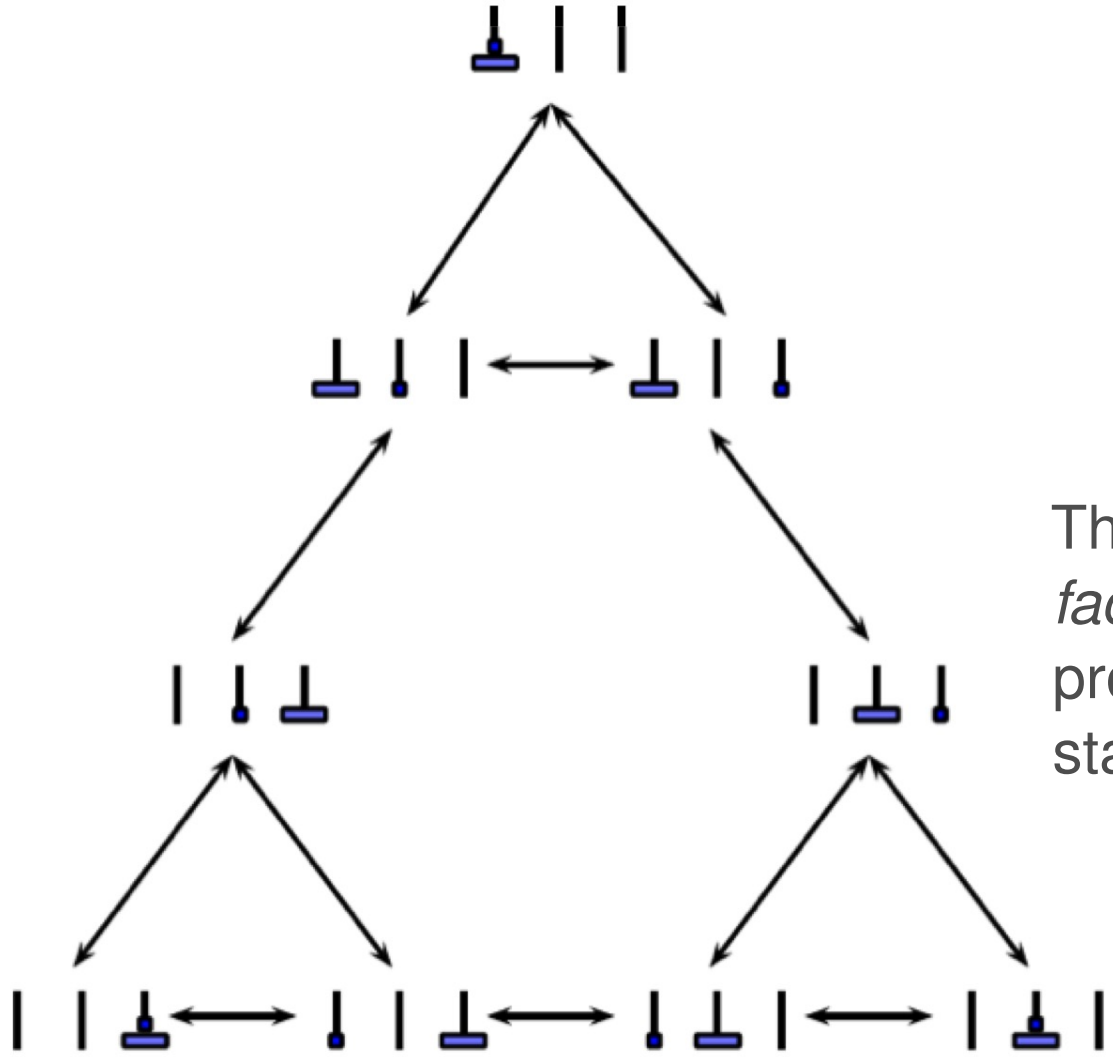
Example: Towers of Hanoi (3,2)



Initial state

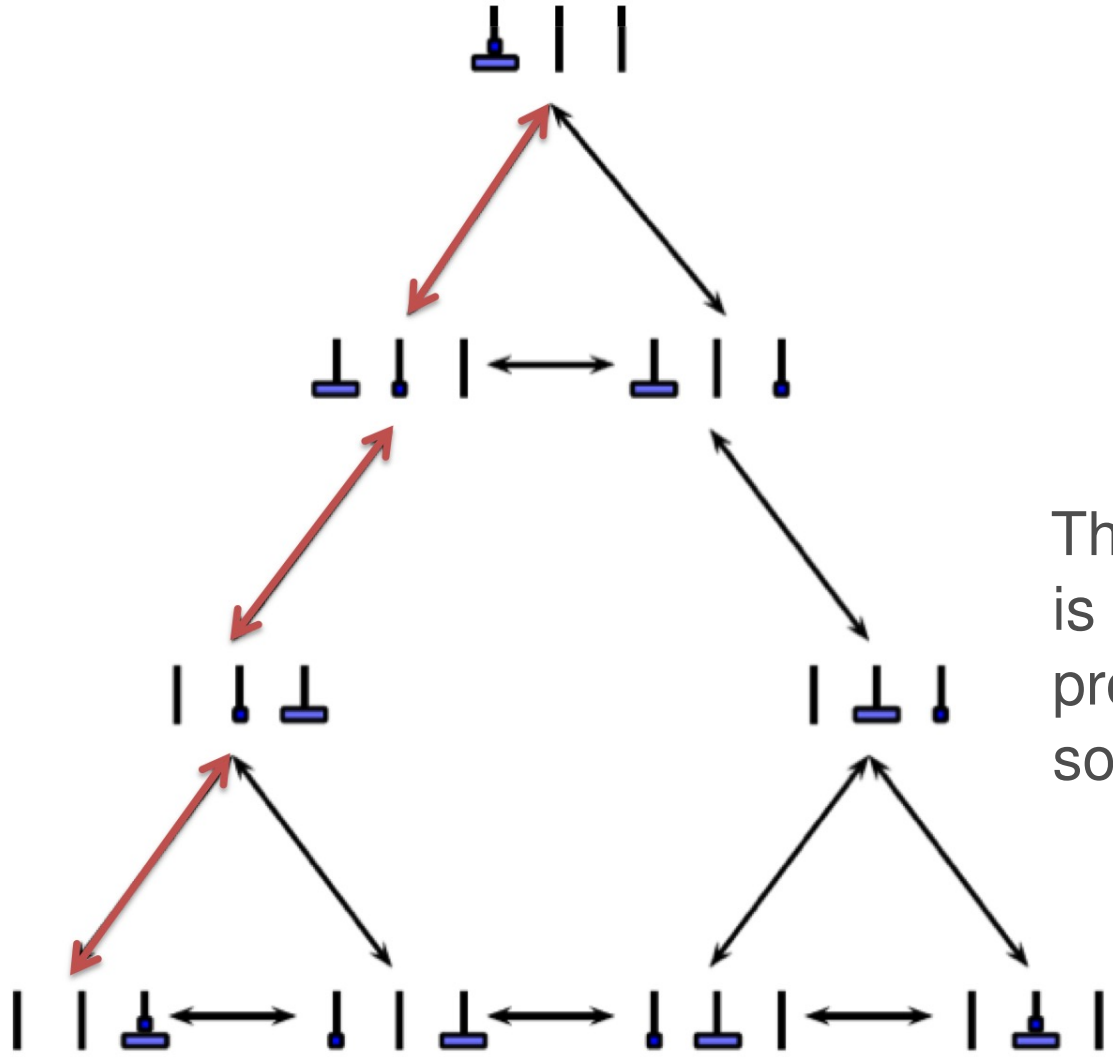
Final state

Example: Towers of Hanoi (3,2)



The *branching factor* ($b = \frac{8}{3}$) is a property of the state graph

Example: Towers of Hanoi (3,2)

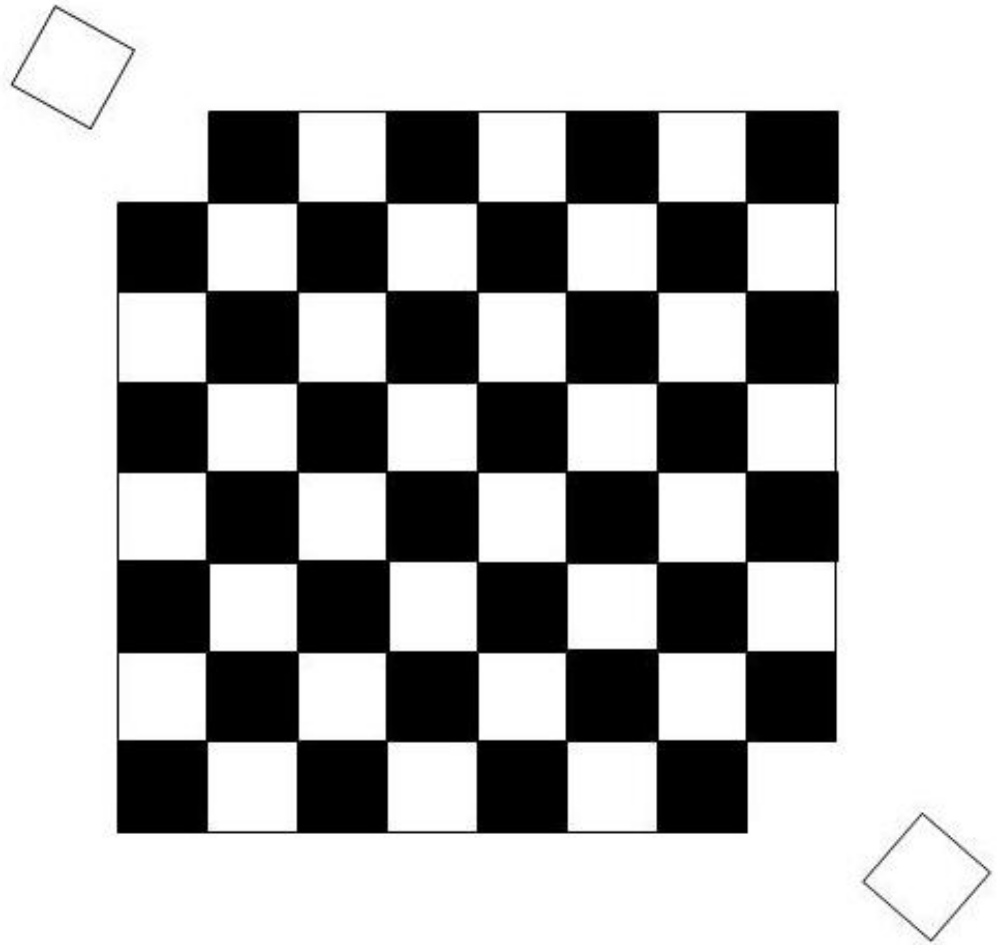


The *depth* ($d = 3$) is a property of the problem to be solved

Combinatorial explosion

Domain	**Number of states**	**Time (10^7 nodes)**
8-puzzle	$\left(\frac{N!}{2}\right)\Big\vert_{N=3} = 181.440$	0.01 seconds
15-puzzle	$\left(\frac{N!}{2}\right)\Big\vert_{N=4} = 10^{13}$	11,5 days
24-puzzle	$\left(\frac{N!}{2}\right)\Big\vert_{N=5} = 10^{25}$	$31,7 \times 10^9$ years
Hanoi (3,2)	$(3^n)\vert_{n=2} = 9$	9×10^{-7} seconds
Hanoi (3,4)	$(3^n)\vert_{n=4} = 81$	$8,1 \times 10^{-6}$ seconds
Hanoi (3,8)	$(3^n)\vert_{n=8} = 6561$	$6,5 \times 10^{-4}$ seconds
Hanoi (3,16)	$(3^n)\vert_{n=16} = 4,3 \times 10^7$	4,3 seconds
Hanoi (3,24)	$(3^n)\vert_{n=24} = 2,824 \times 10^{11}$	0,32 days
hline Rubik cube $2 \times 2 \times 2$	10^6	0,1 seconds
Rubik cube $3 \times 3 \times 3$	$4,32 \times 10^{19}$	31.000 years

The role of representation. Mutilated chessboard

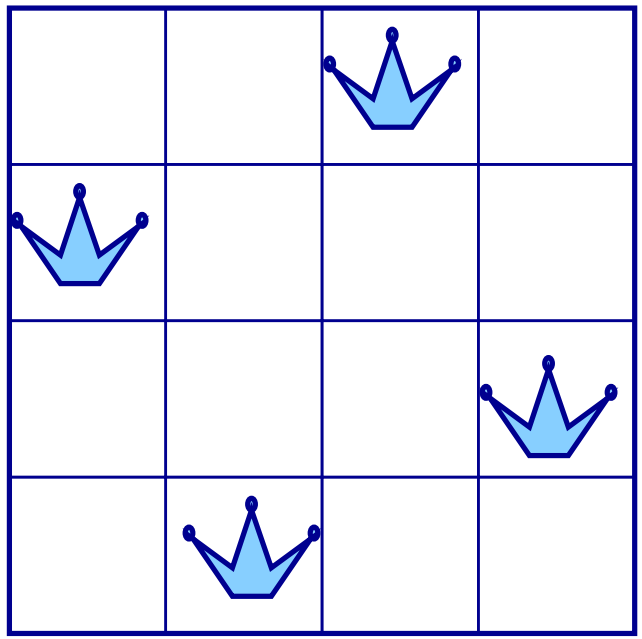


can you fill it with dominoes pieces?

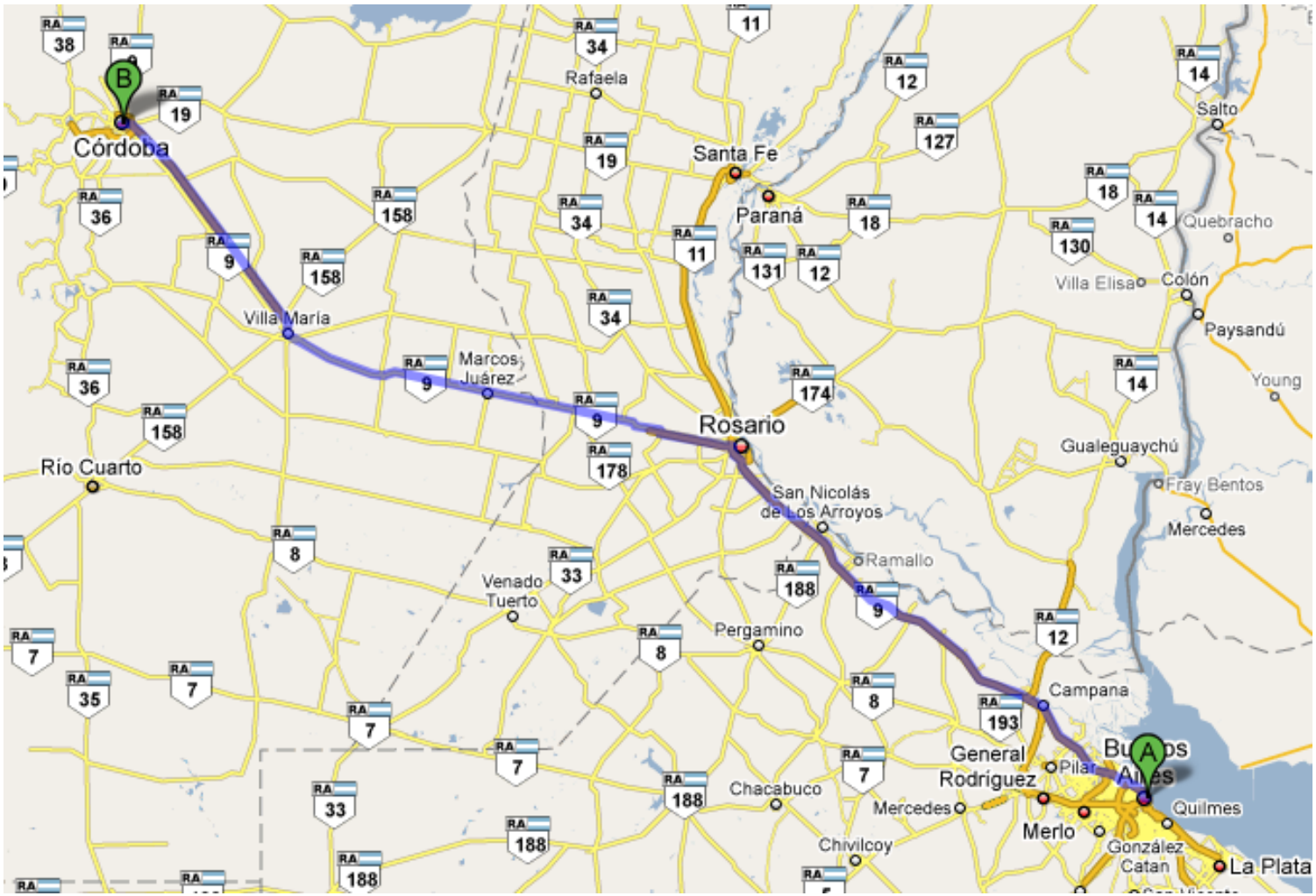
Example: 8 queens

- Objective: Place 8 queens on a chess board in a way that they do not attack each other (a queen attacks another if they both are on the same row, column or diagonal)
- Two possible formulations of the problem:
  - Formulation of complete states: place the 8 queens on the table and move them
  - Incremental formulation: start with an empty table, and place the queens one at a time
- There is no explicit description of the goal state and the solution path is not relevant

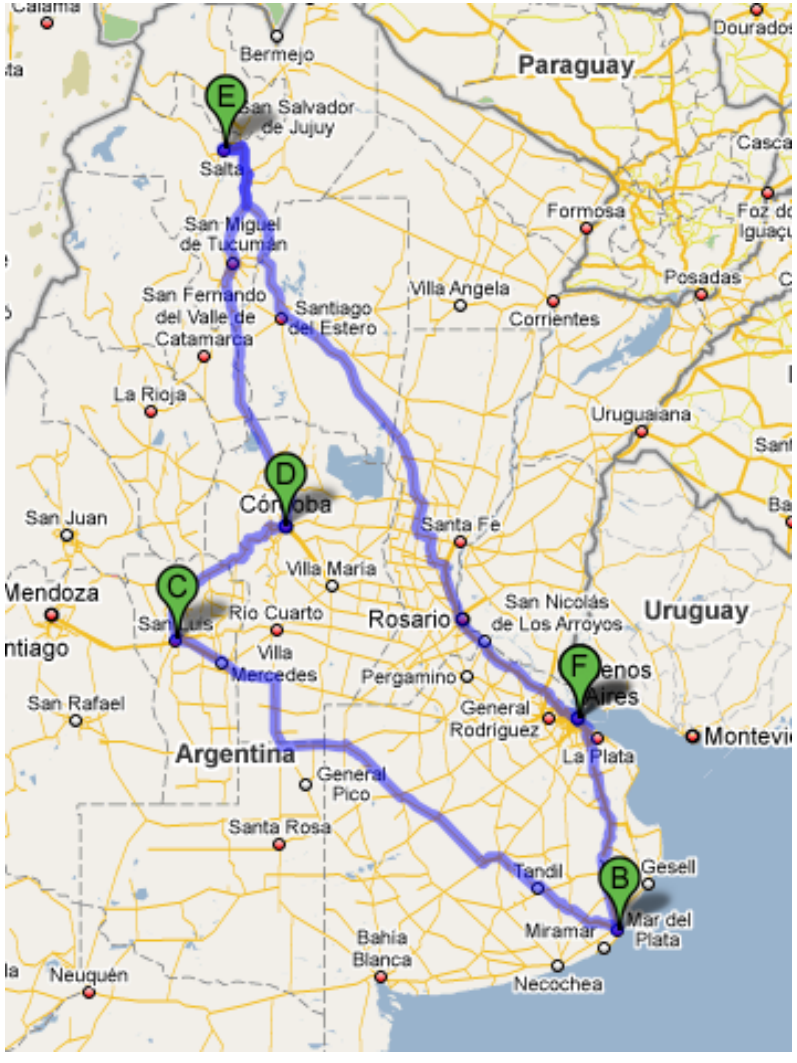
Constraints satisfaction. 4-queens



Shortest path

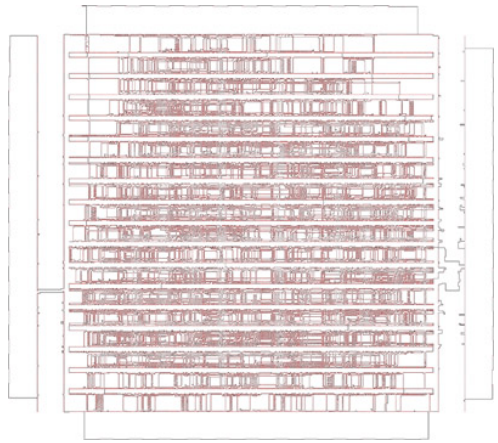


Travelling Salesman Problem



Travelling salesperson

- Complexity: 49-city map (1950s); 2,392-city map (1980s); and 85,900-city map (2006)



- Layout of cities (BELL Labs): design of a customized computer chip
- Solution: shortest path for a laser to follow as it sculpts the chip

Genetics

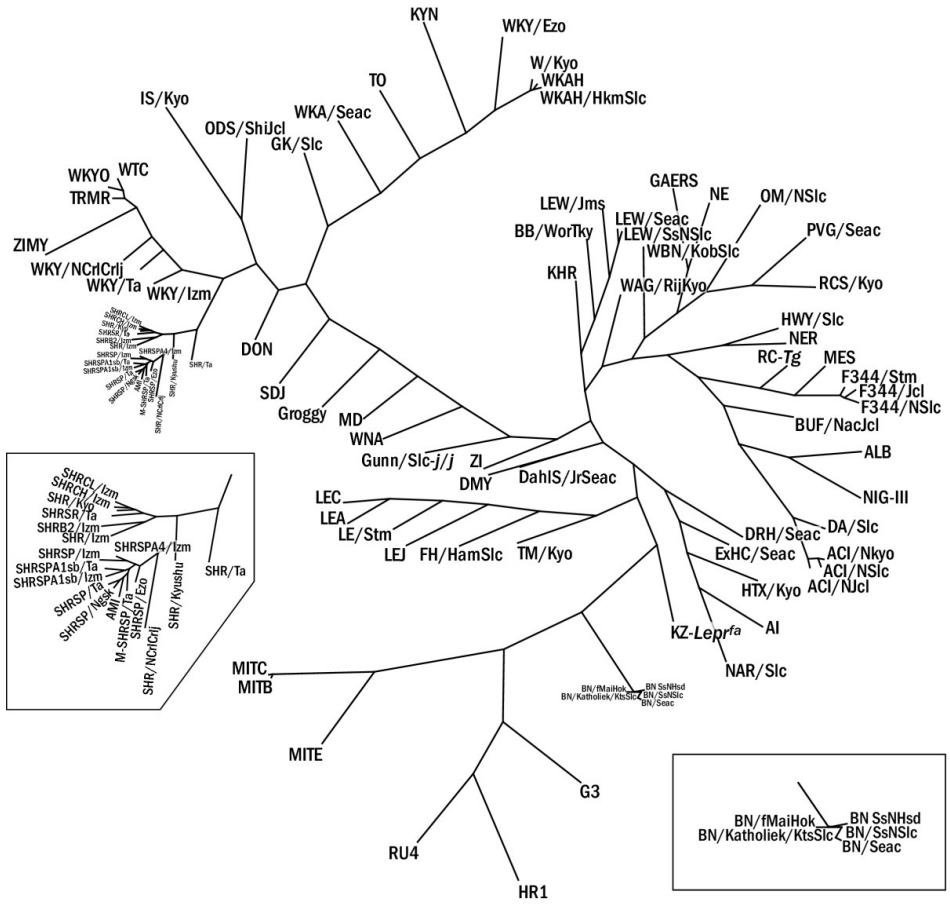


Figure 4

Games


: 3073 (+65/Turn)
Research: Military Tradition (2)
00:57 - Turn: 260 - 1750 AD
:90% (+1162/Turn)
:0% (+1035/Turn)
:0% (+83/Turn)
Deal Canceled: Ivory to Brennus for Deer
Deal Canceled: Pig to Brennus for Sheep
Autosaving...
Our scientists have developed better ways of processing coal fuel. Our economy stands ready to benefit from this development.
Put it to use at our Coal-fueled power plants. Production is power!
I suppose we can make our Drydocks a more healthy place to work and slightly more productive, then.
Drydock: +2
Drydock: +1
+60%
7
Bombay
Observatory
3322: [Terra]
1737: Ramesses II
1612: Sitting Bull
1410: Mansa Musa
1159: Brennus -Guilds (41)
978: Cyrus
880: Asoka -Scientific Method (1)
+40%
10
Calcutta
Buddhist Ter

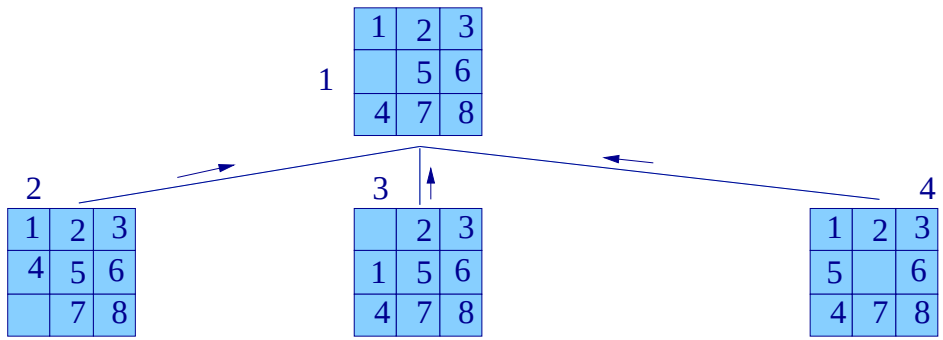
Autonomous vehicles

http://www.youtube.com/watch?v=7R7d-bYSyUE

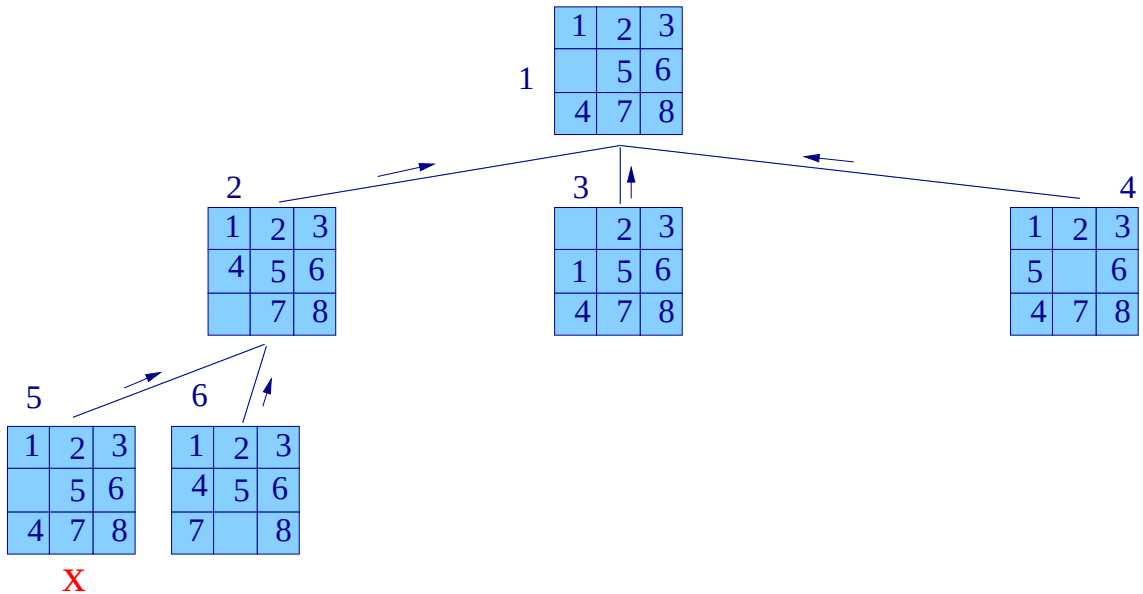
Outline

Outline

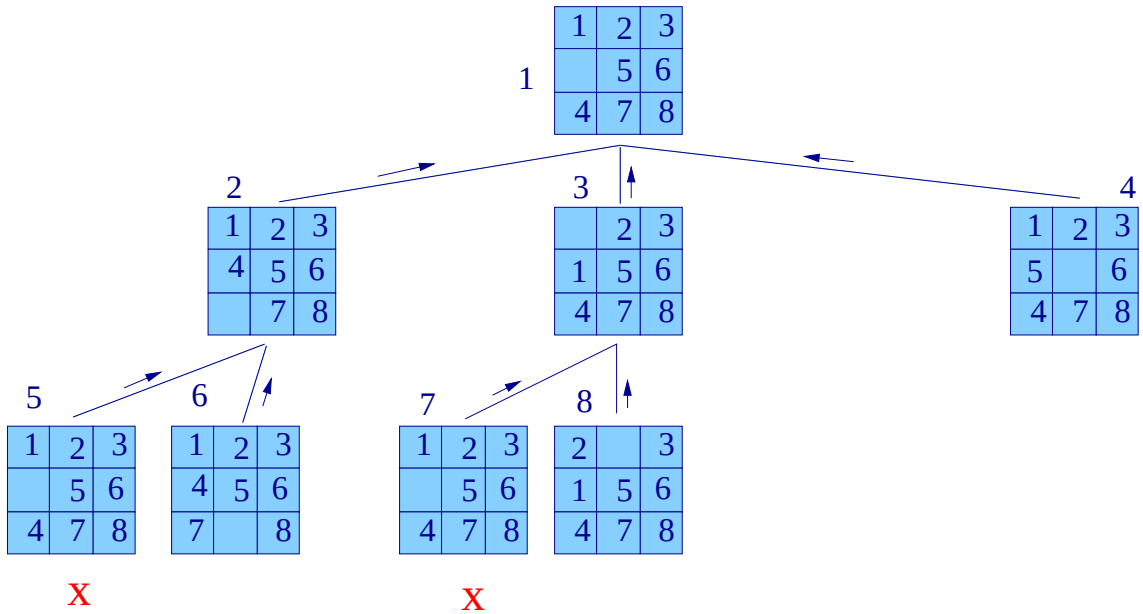
8-Puzzle – Breadth-first



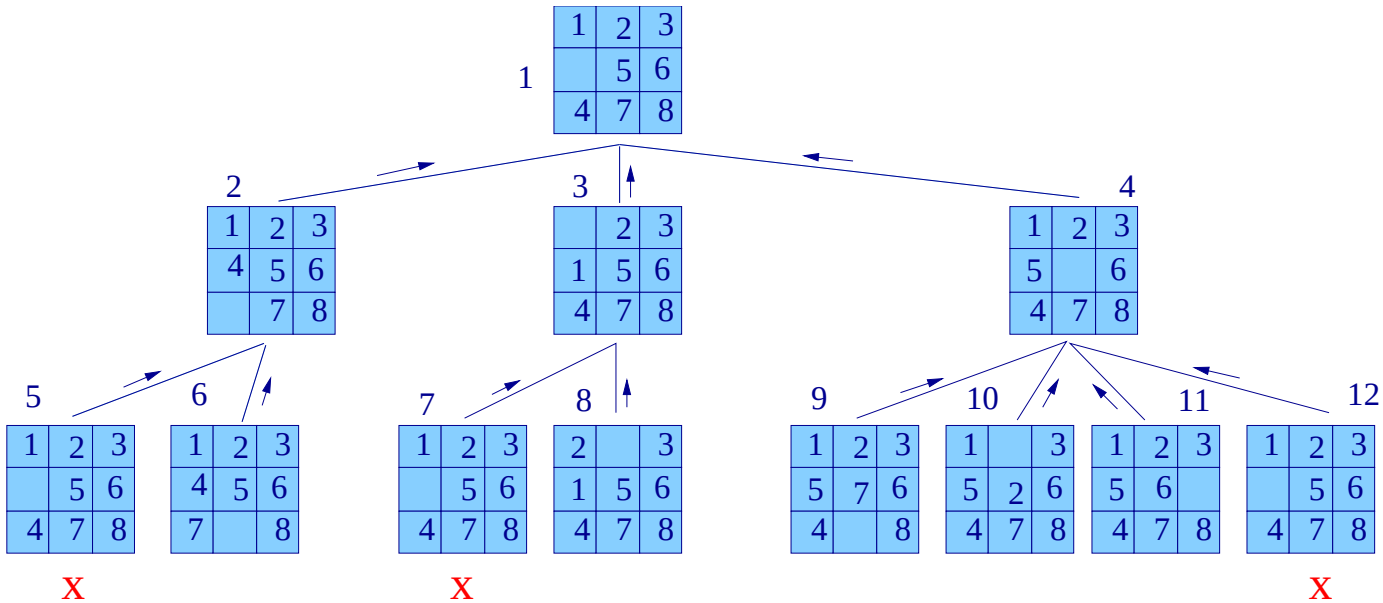
8-Puzzle – Breadth-first



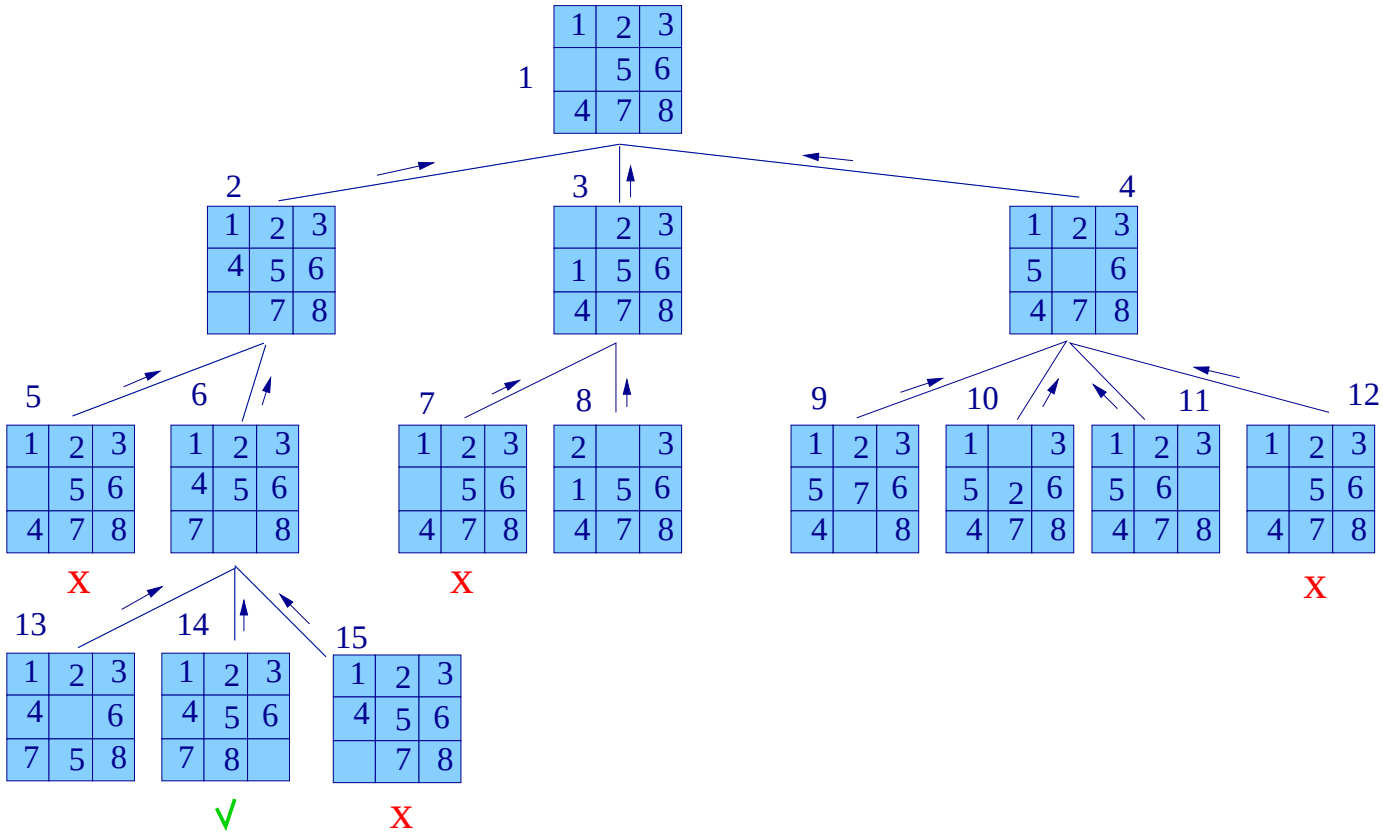
8-Puzzle – Breadth-first



8-Puzzle – Breadth-first



8-Puzzle – Breadth-first



Breadth-first Search

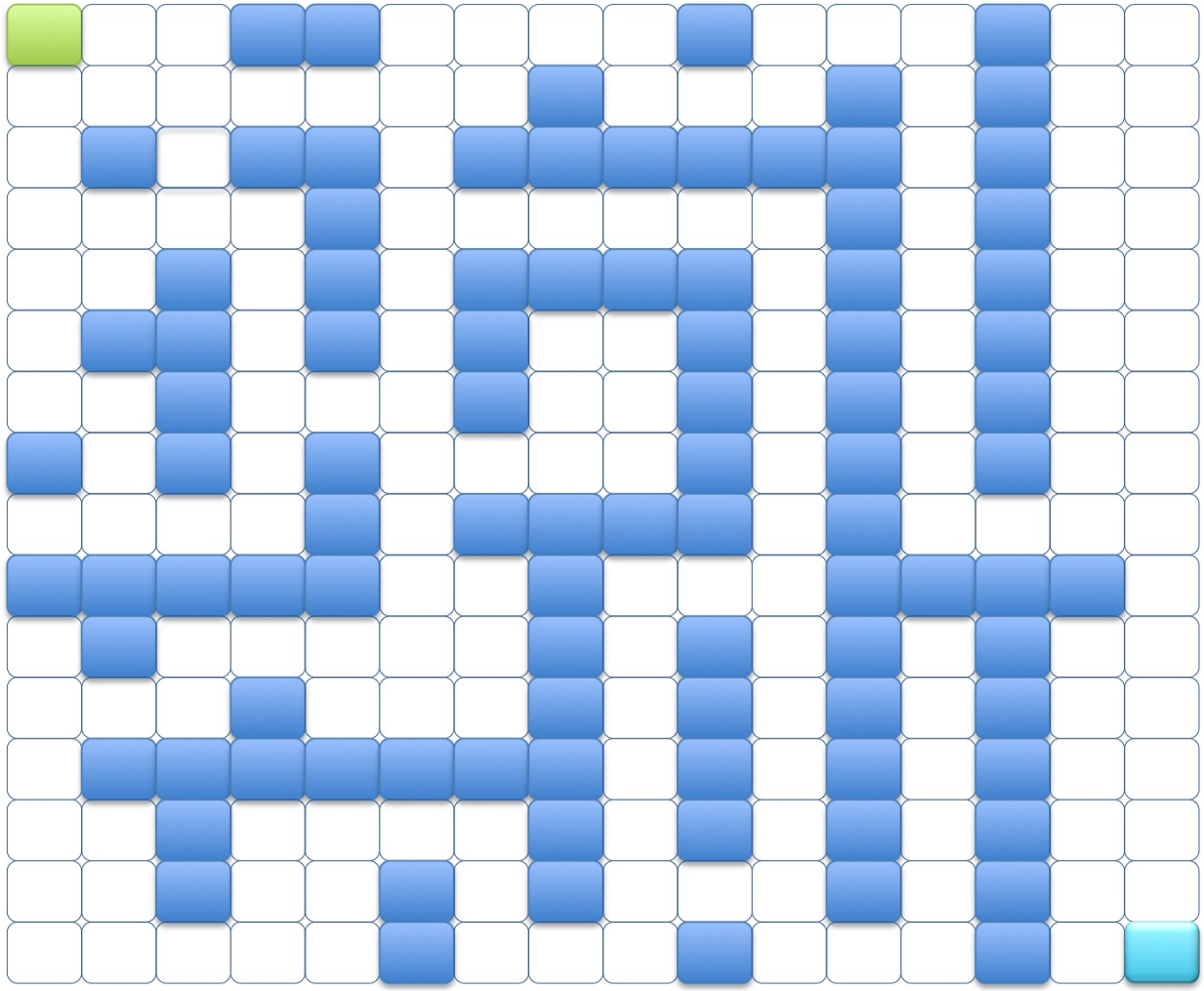
Breadth-first search (Initial-state, Final-state)

1. Create an OPEN list with the initial node, I, (initial-state)
2. OUTCOME=False
3. Repeat until OPEN list is empty or OUTCOME is true

   Remove the first node from OPEN list, N
   If N has successors
   Then Generate the successors of N
   Create pointers from the successors to N
   If a successor is a goal node
   Then OUTCOME=True
   Else add successors at end of OPEN list

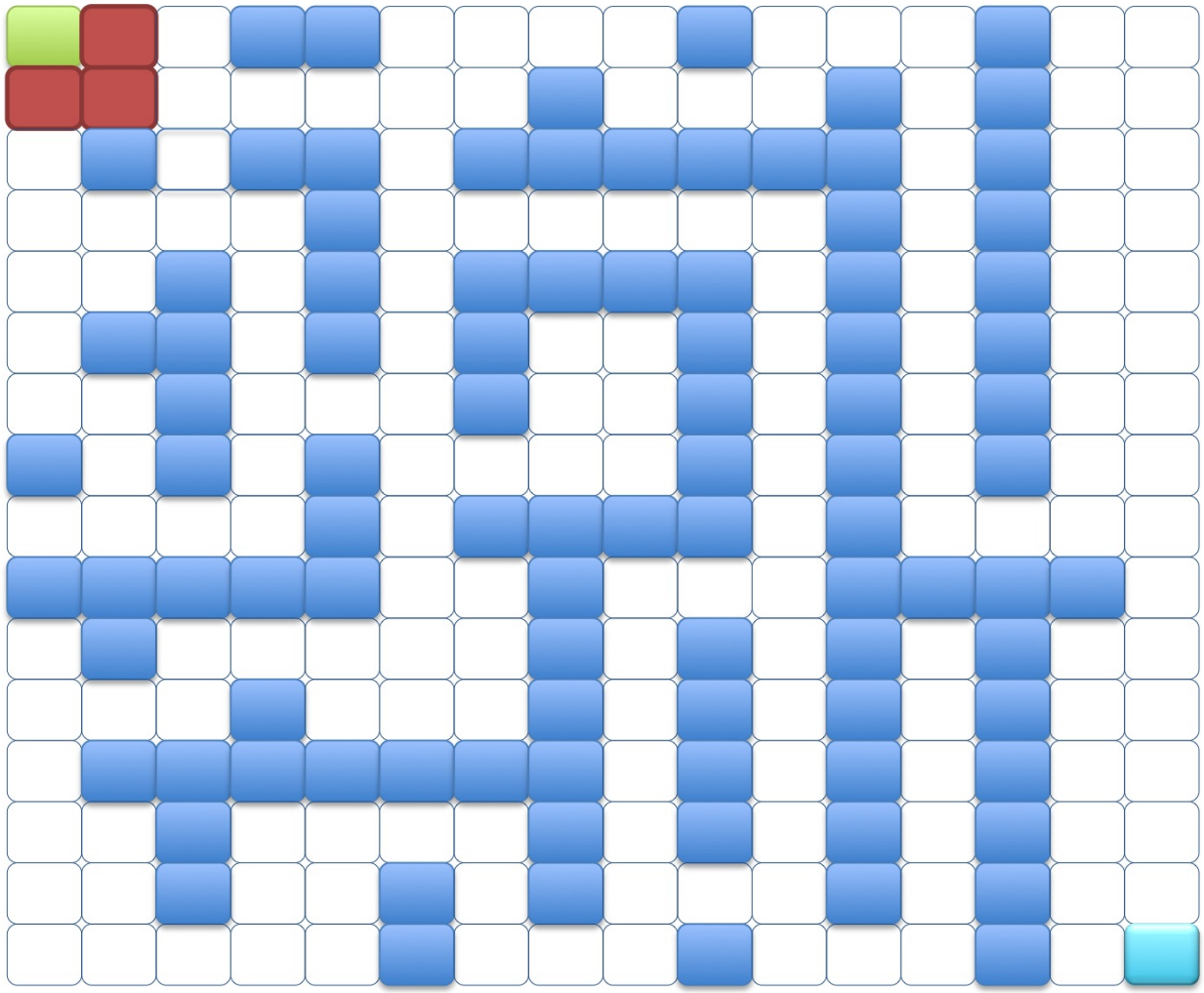
4. If OUTCOME
   Then Solution = path from I to N through the pointers
   Else Solution = False

Labyrinth

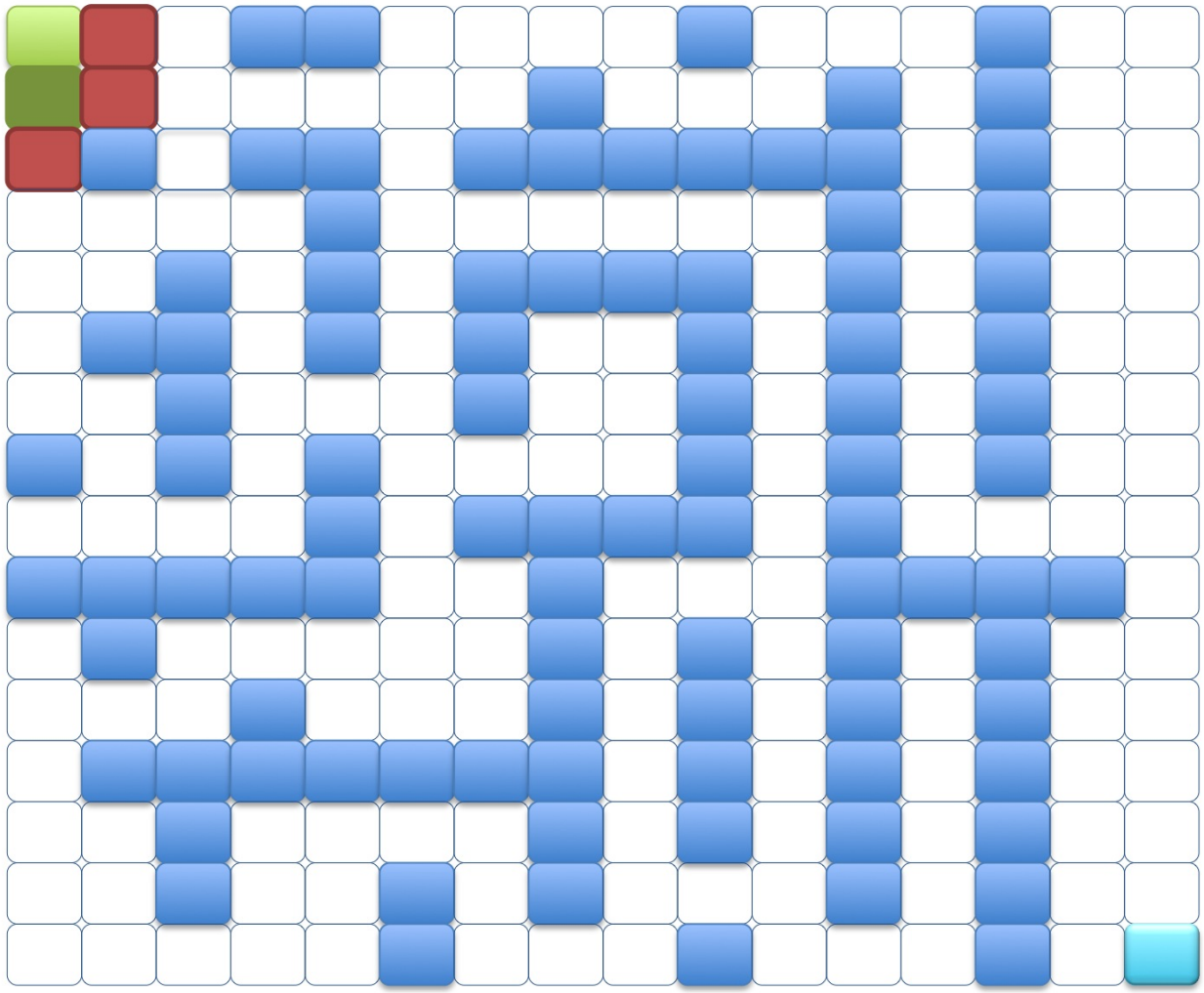


Labyrinth

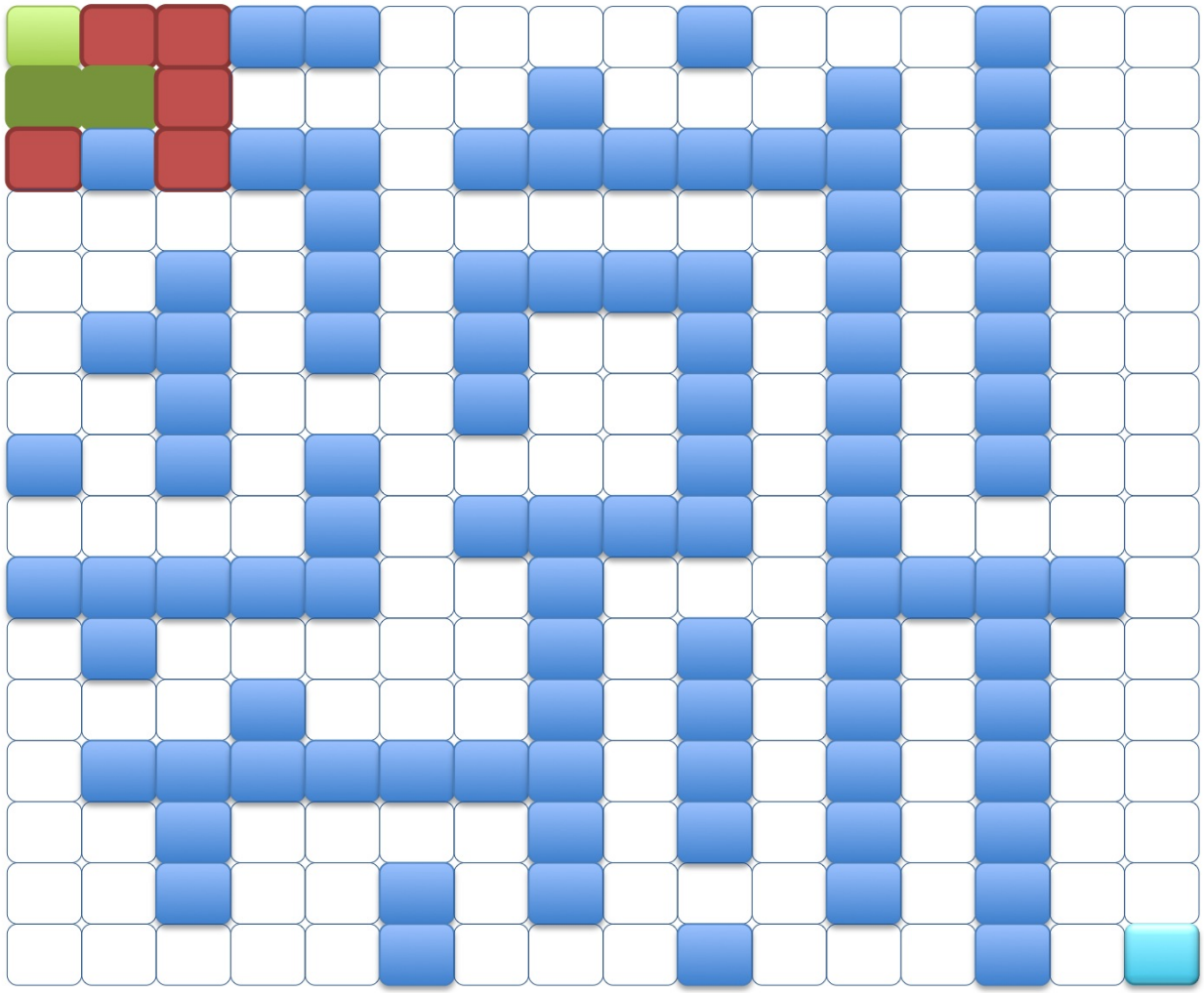
Labyrinth



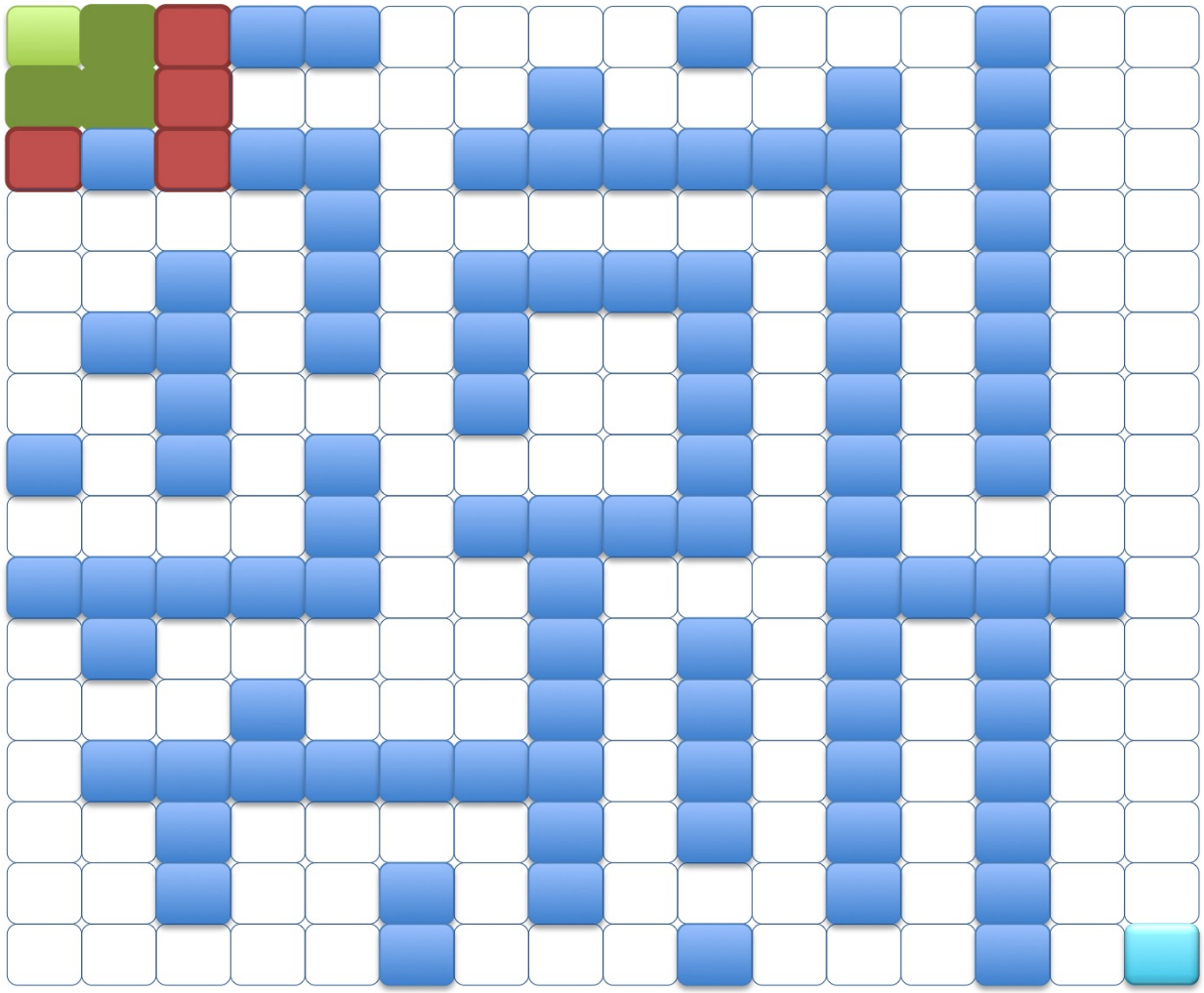
Labyrinth. Repeated states



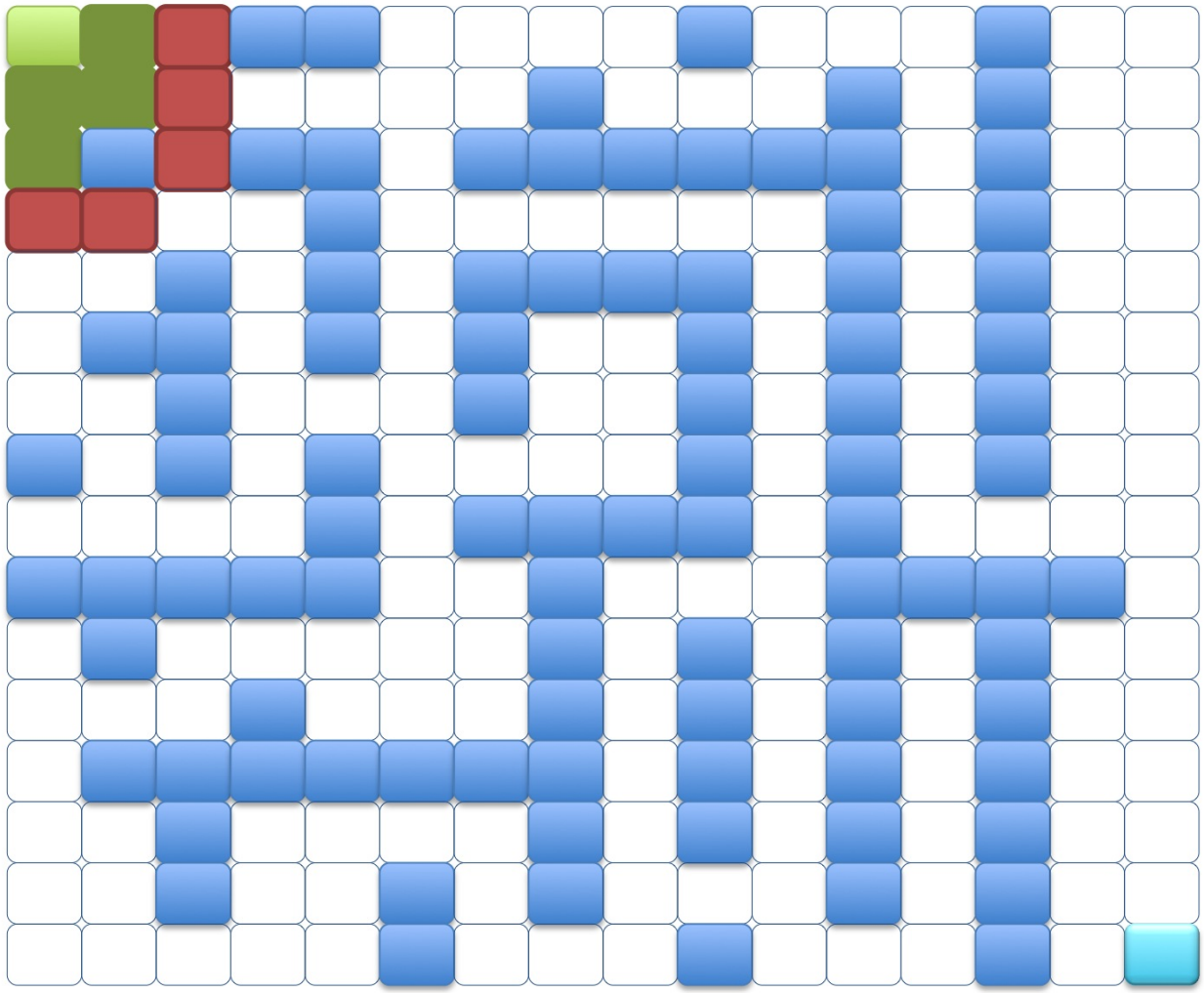
Labyrinth



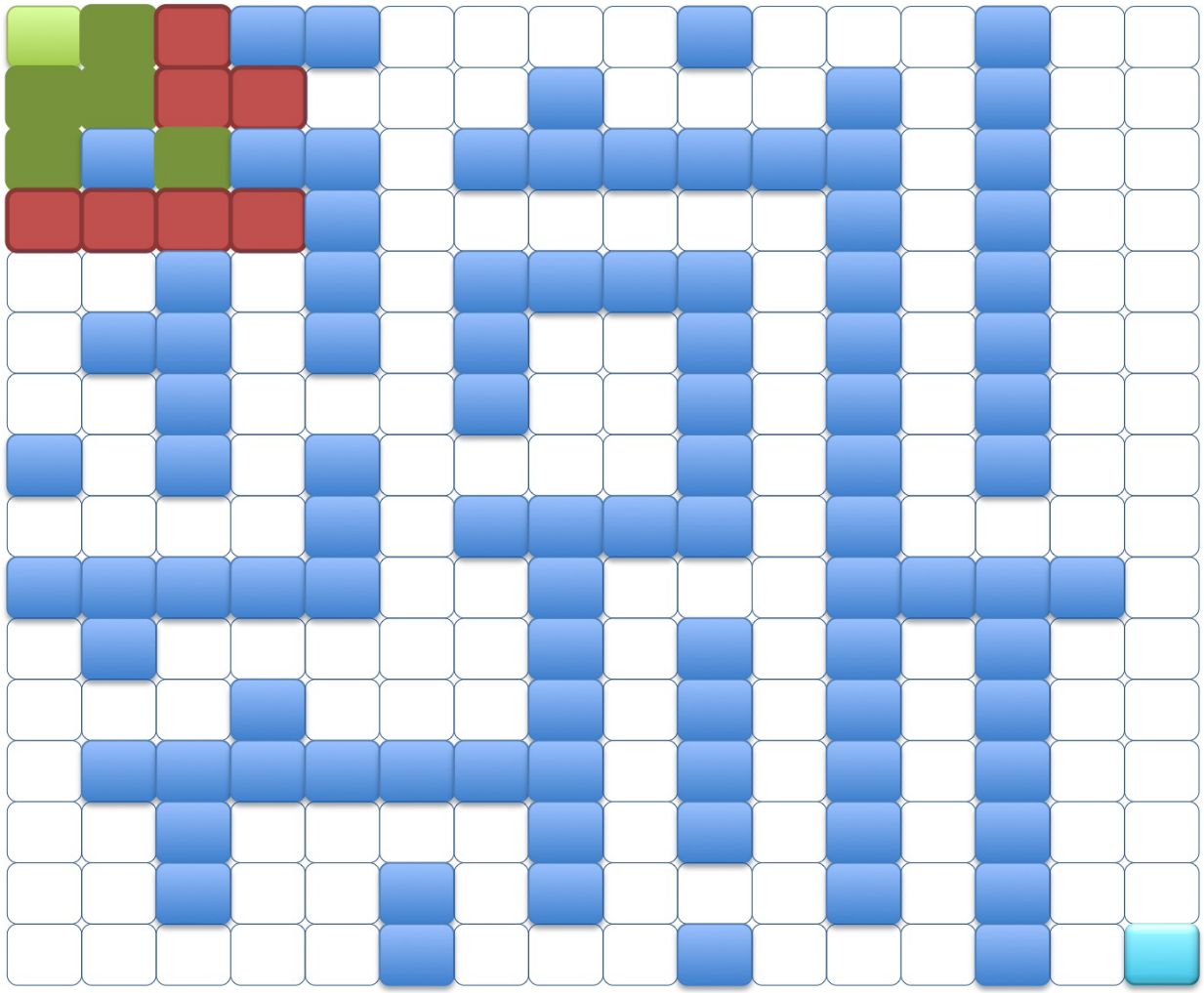
Labyrinth



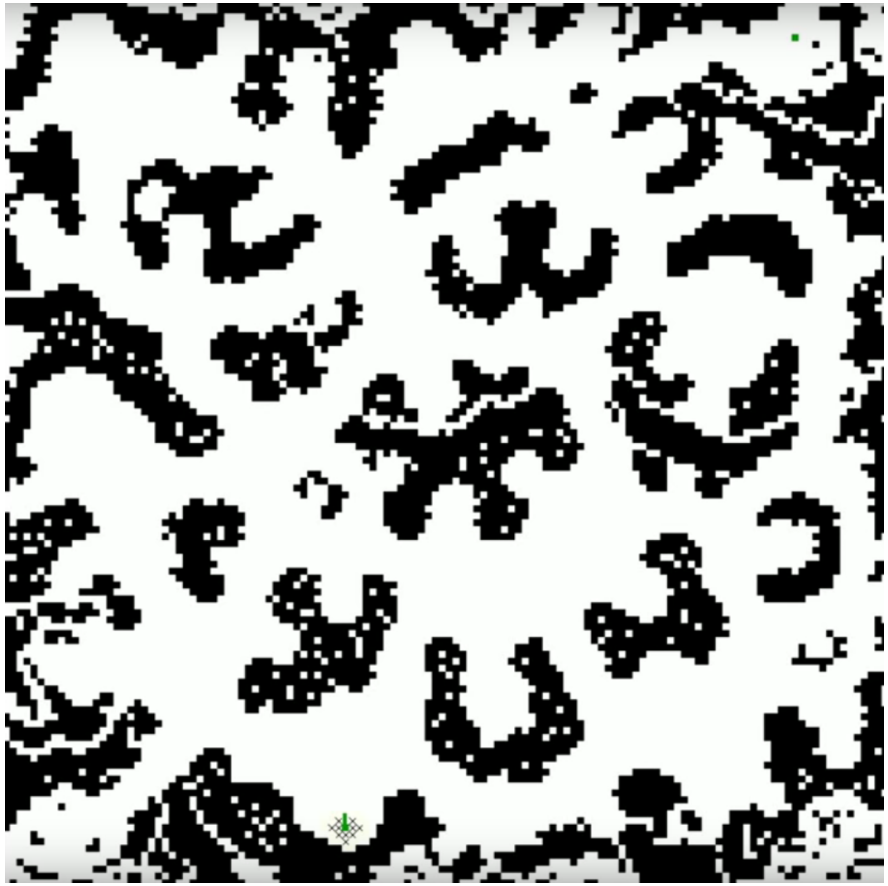
Labyrinth



Labyrinth



Example in a game map



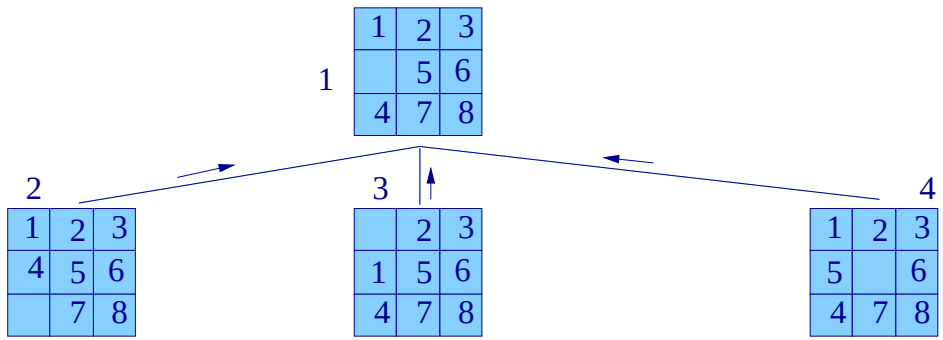
https://www.youtube.com/watch?v=z6lUnb9ktkE

Characteristics

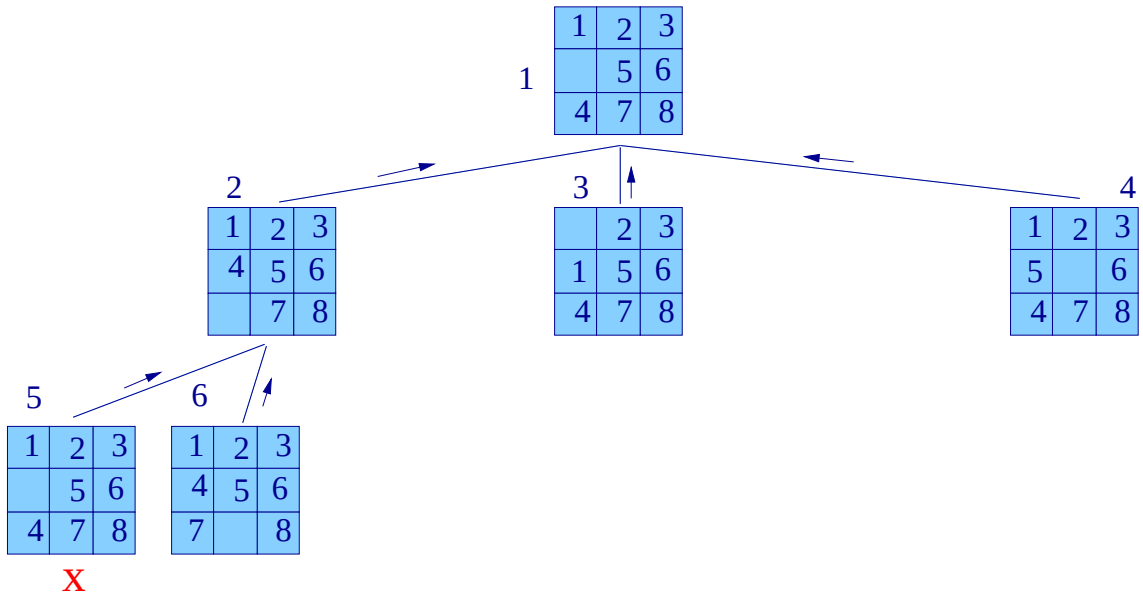
- Completeness: if a solution exists and the branching factor is finite at each node, then it will find it
- Admissibility: if all the nodes have the same cost, it will find an optimal solution
- Efficiency: good if the goals are near
- Problem: exponential memory consumption

Outline

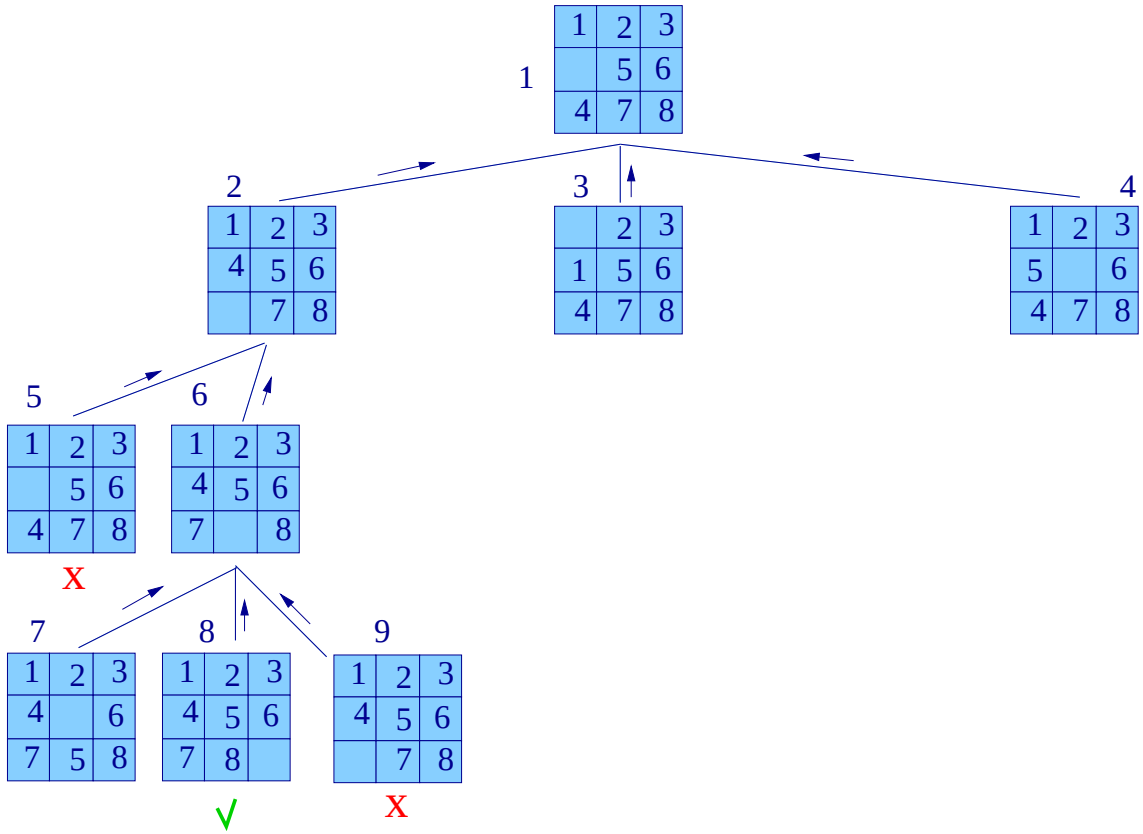
8-Puzzle – Depth-First



8-Puzzle – Depth-First



8-Puzzle – Depth-First



Depth-first Search

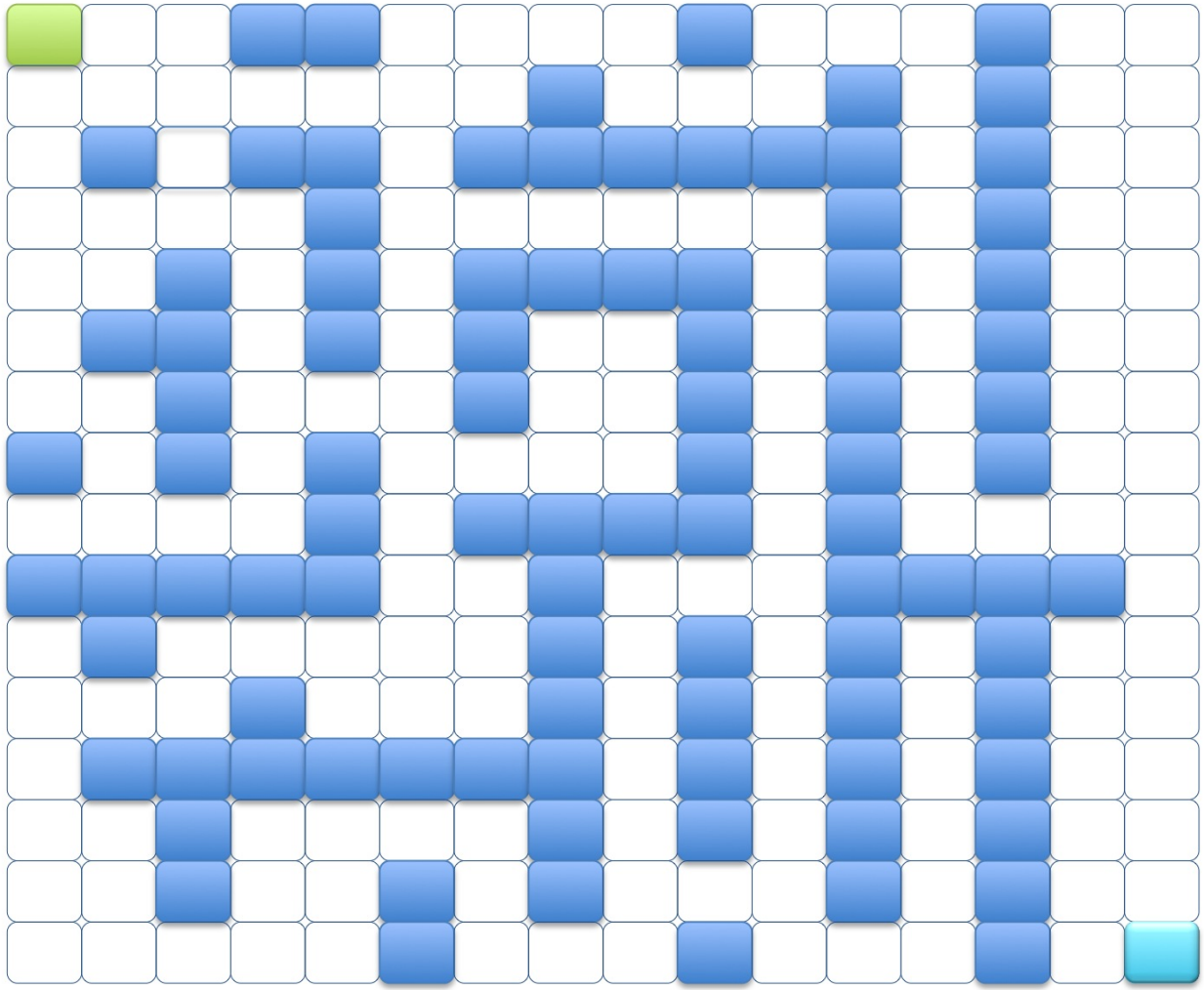
Depth-first search (Initial-state, Goal-state, Maximum-depth)

1. Create an OPEN list with the initial node, I, and depth=0
2. OUTCOME=False
3. Repeat until the OPEN list is empty or OUTCOME is true

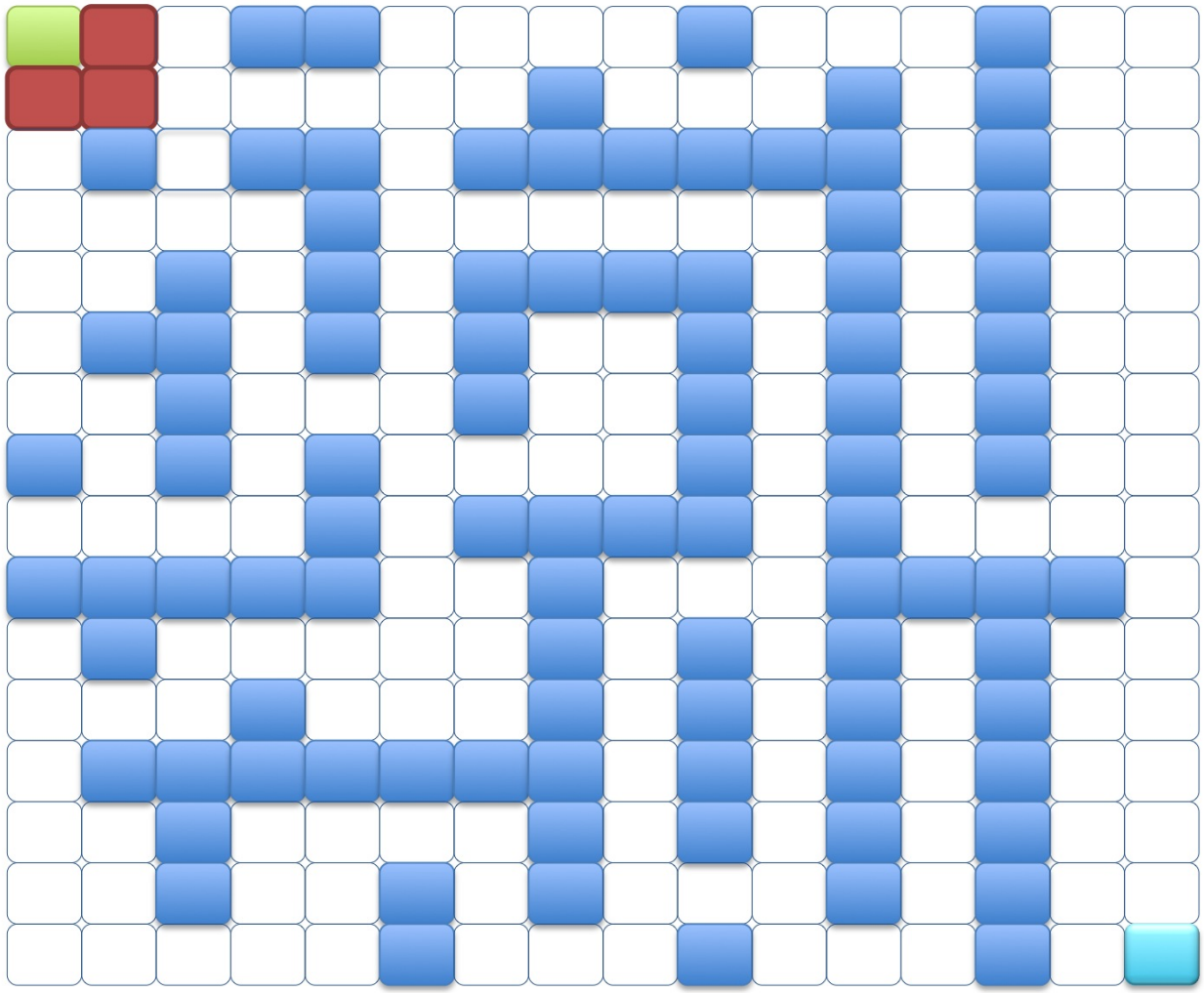
   Remove the first node N from the OPEN list
   Call P its depth
   If P $<$ Maximum-depth and N has successors
   Then Generate the successors of N
   Create pointers from the successors to N
   If a successor is goal state
   Then OUTCOME=True
   Else Add successors at beginning of the OPEN list
   Assign it depth P+1

4. If OUTCOME
   Then Solution = path from I to N through the pointers
   Else Solution = False

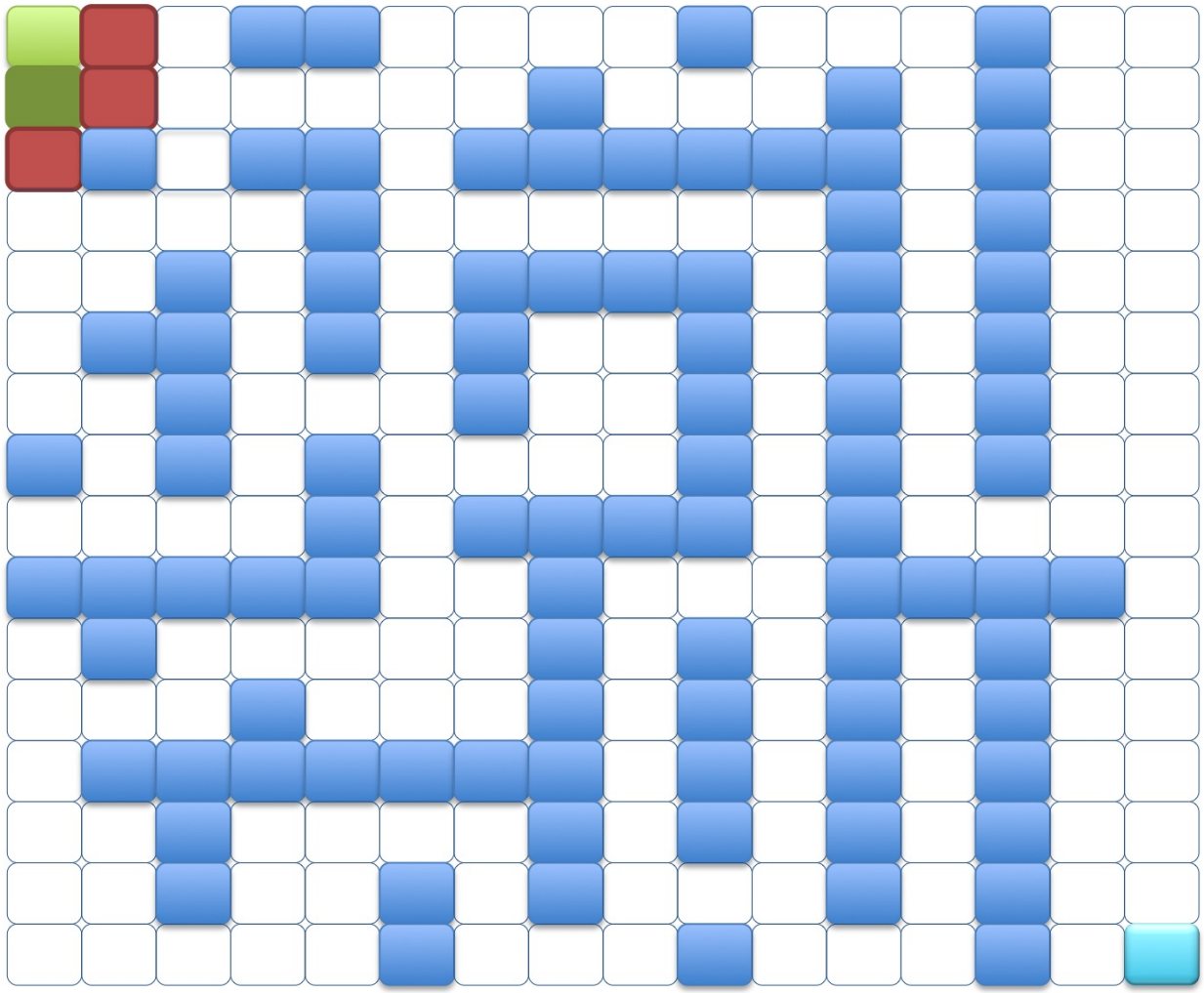
Labyrinth



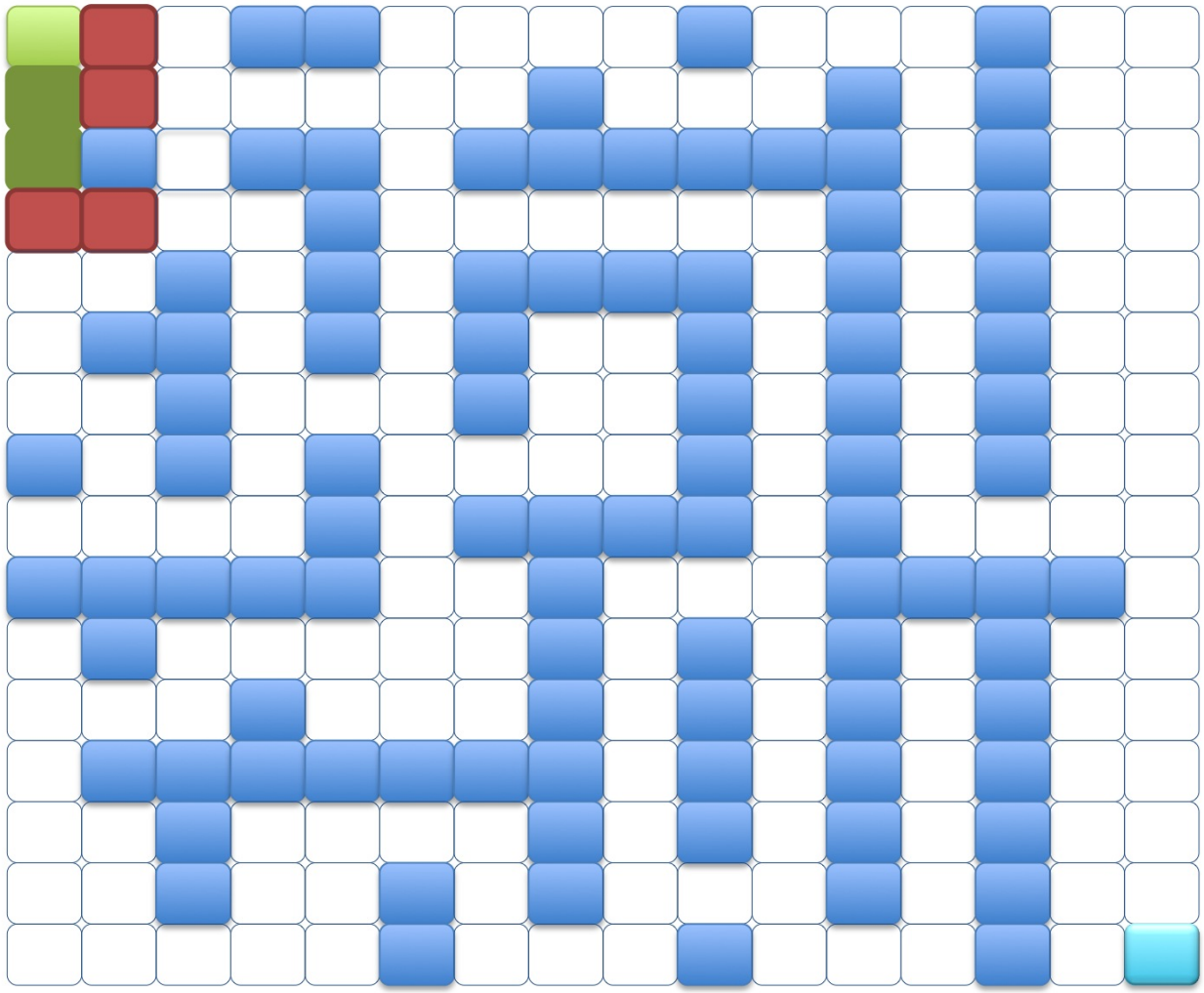
Labyrinth



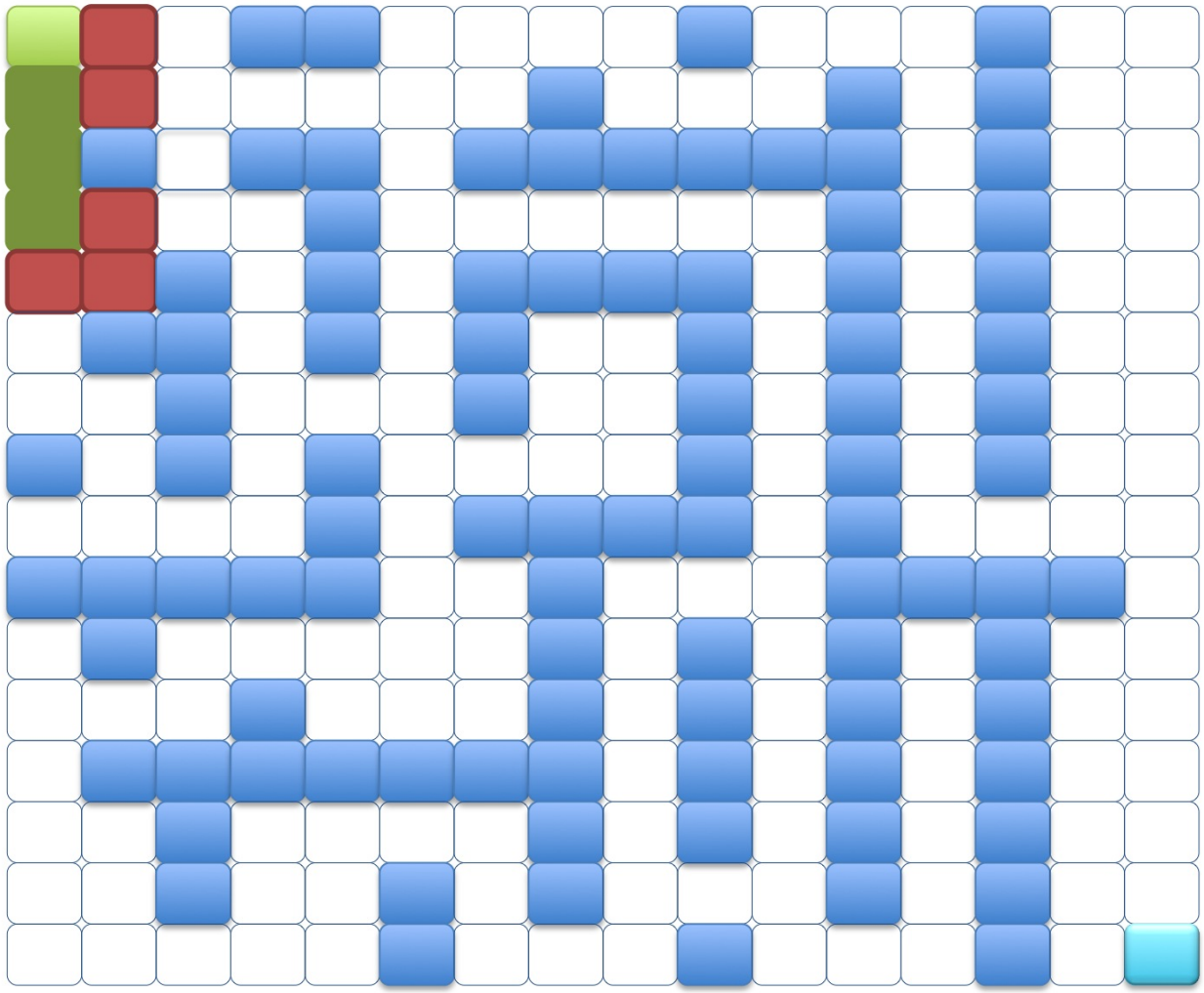
Labyrinth



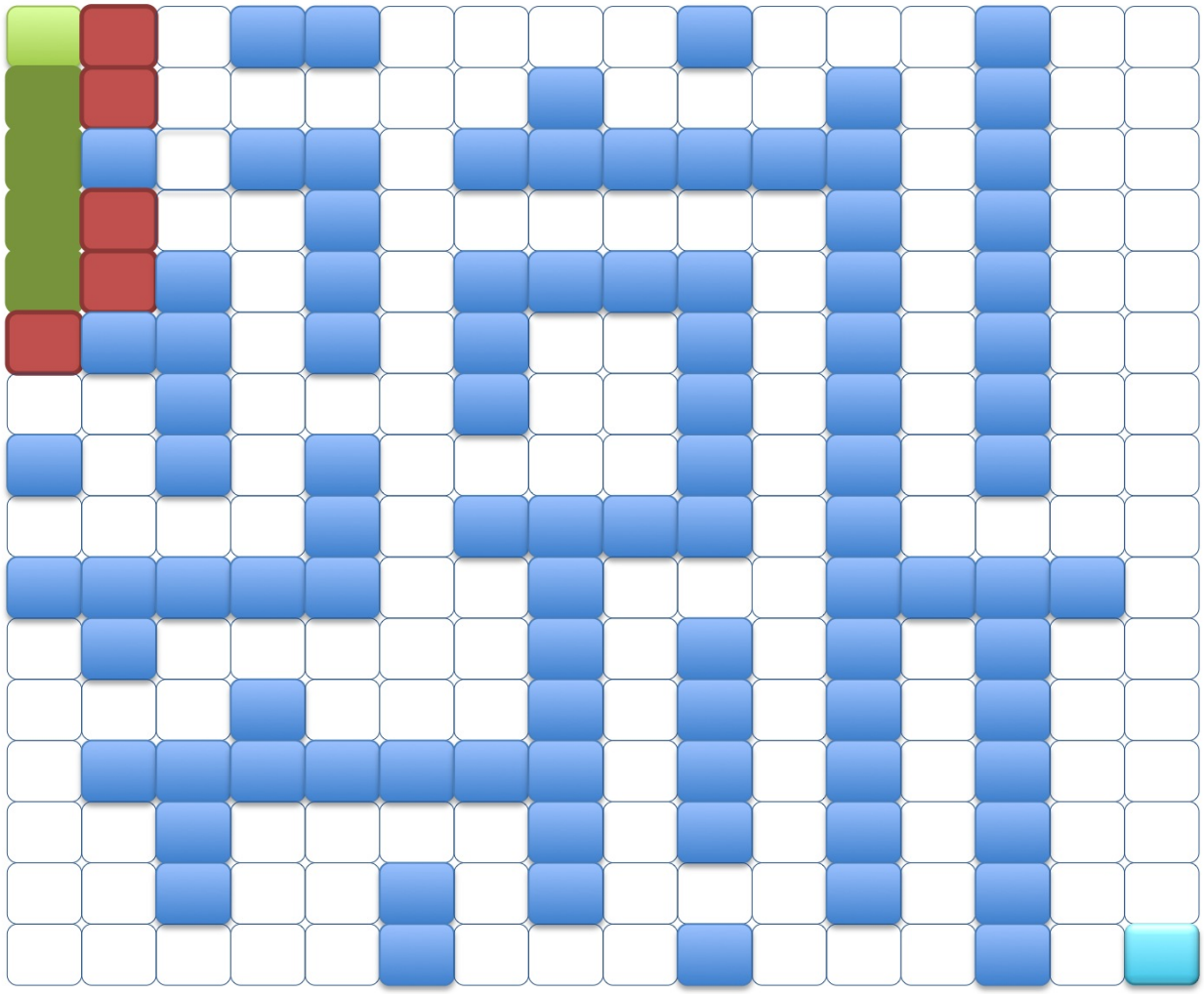
Labyrinth



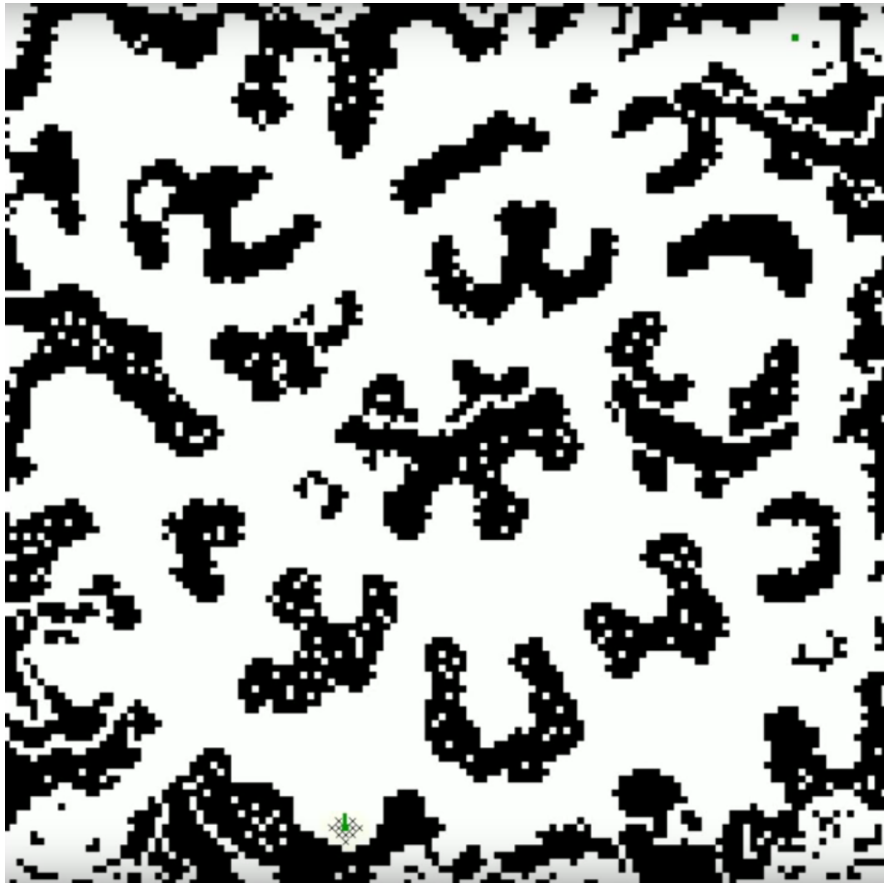
Labyrinth



Labyrinth



Example in a game map



https://www.youtube.com/watch?v=dtoFAvtVE4U

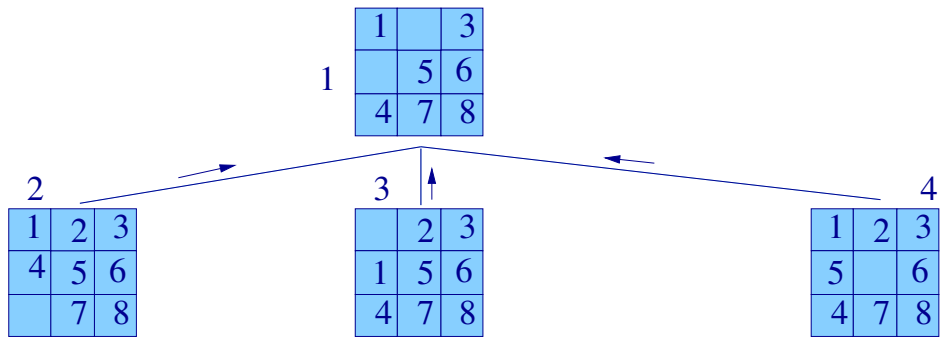
Characteristics

- Requires backtracking
- Reasons for backtracking:
  - it has reached the depth limit
  - it has studied all the successors of the node but no solution was found
  - it generates a duplicate state

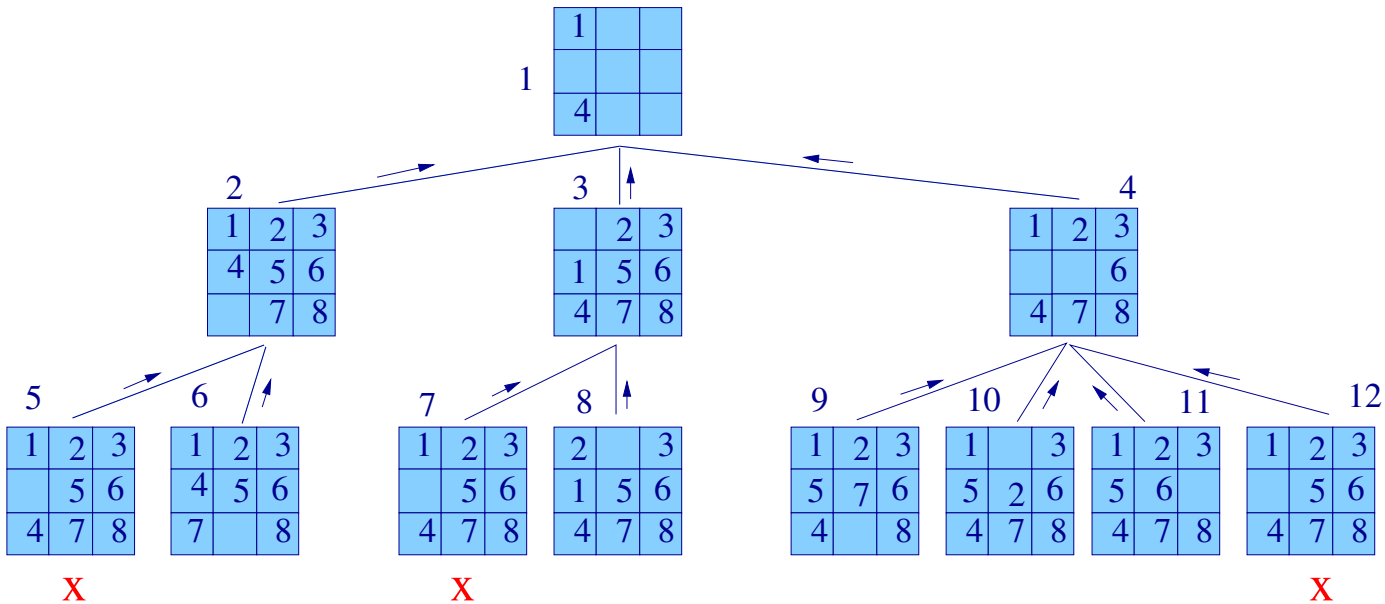
Characteristics

- Requires backtracking
- Reasons for backtracking:
  - it has reached the depth limit
  - it has studied all the successors of the node but no solution was found
  - it generates a duplicate state
- Completeness: it does not ensure a solution (incomplete)
- Admissibility: it does not ensure an optimal solution (inadmissible)
- Efficiency: good when the goals are far from the initial state, or there are memory limitations. Bad when there are cycles

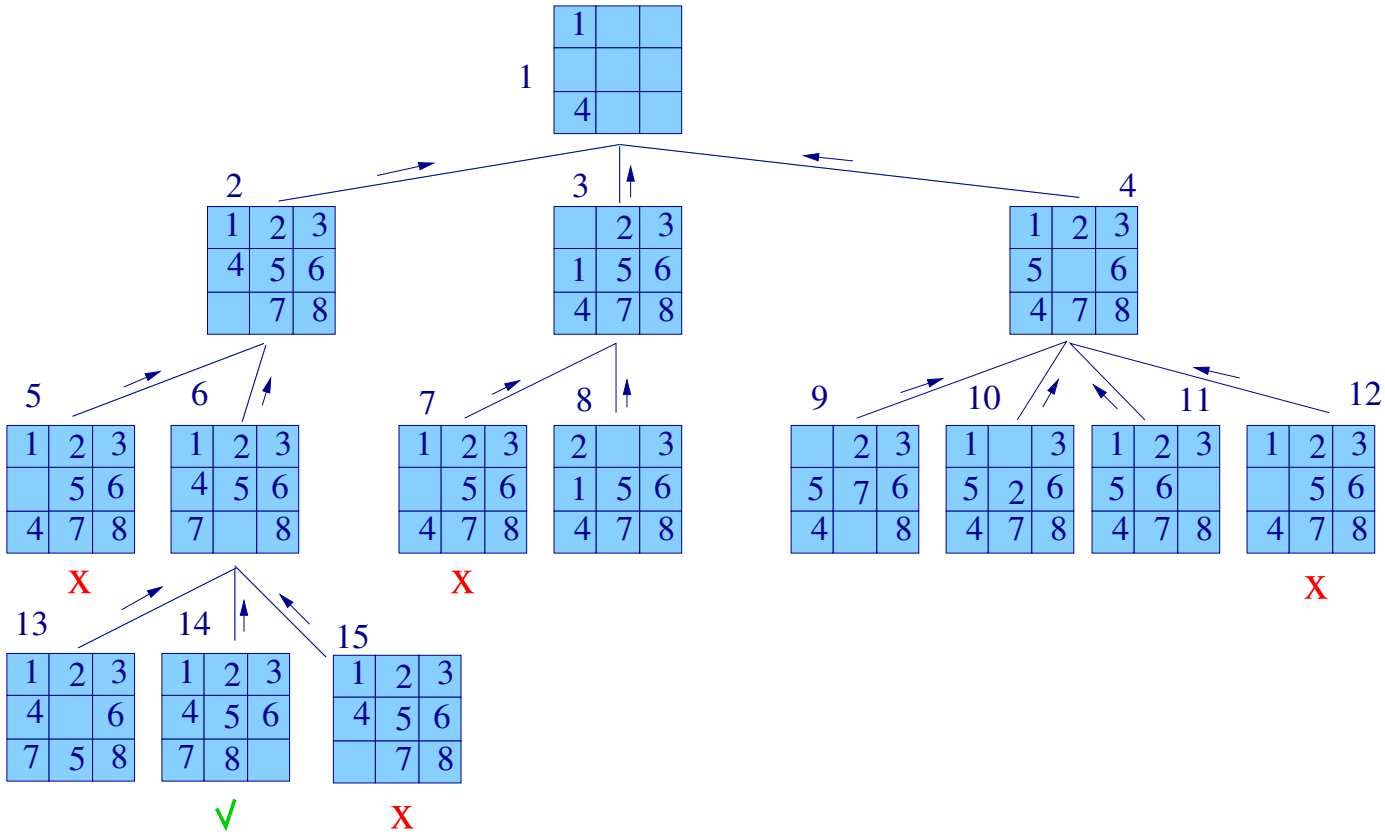
Iterative deepening depth-first



Iterative deepening depth-first



Iterative deepening depth-first



Iterative deepening depth-first

Iterative deepening depth-first search (Initial-state, Goal-state, Increment)

1. Maximum-depth = Increment
2. OUTCOME = False
3. While OUTCOME = False

   OUTCOME = Depth-first search (Initial-state, Goal-state, Maximum-depth)

   Maximum-depth = Maximum-depth + Increment
4. If OUTCOME
   Then solution = path from I to N through the pointers
   Else Solution = False

Characteristics

- Completeness: finds a solution, if one exists
- Admissibility: finds an optimal solution,
  if `Increment = 1`
- Efficiency:

$$\frac{\text{Time(Iterative deepening depth} - \text{first)}}{\text{Time(Breadth} - \text{first)}} = \frac{b}{b-1}$$

- Problem: generates many duplicate nodes

Outline

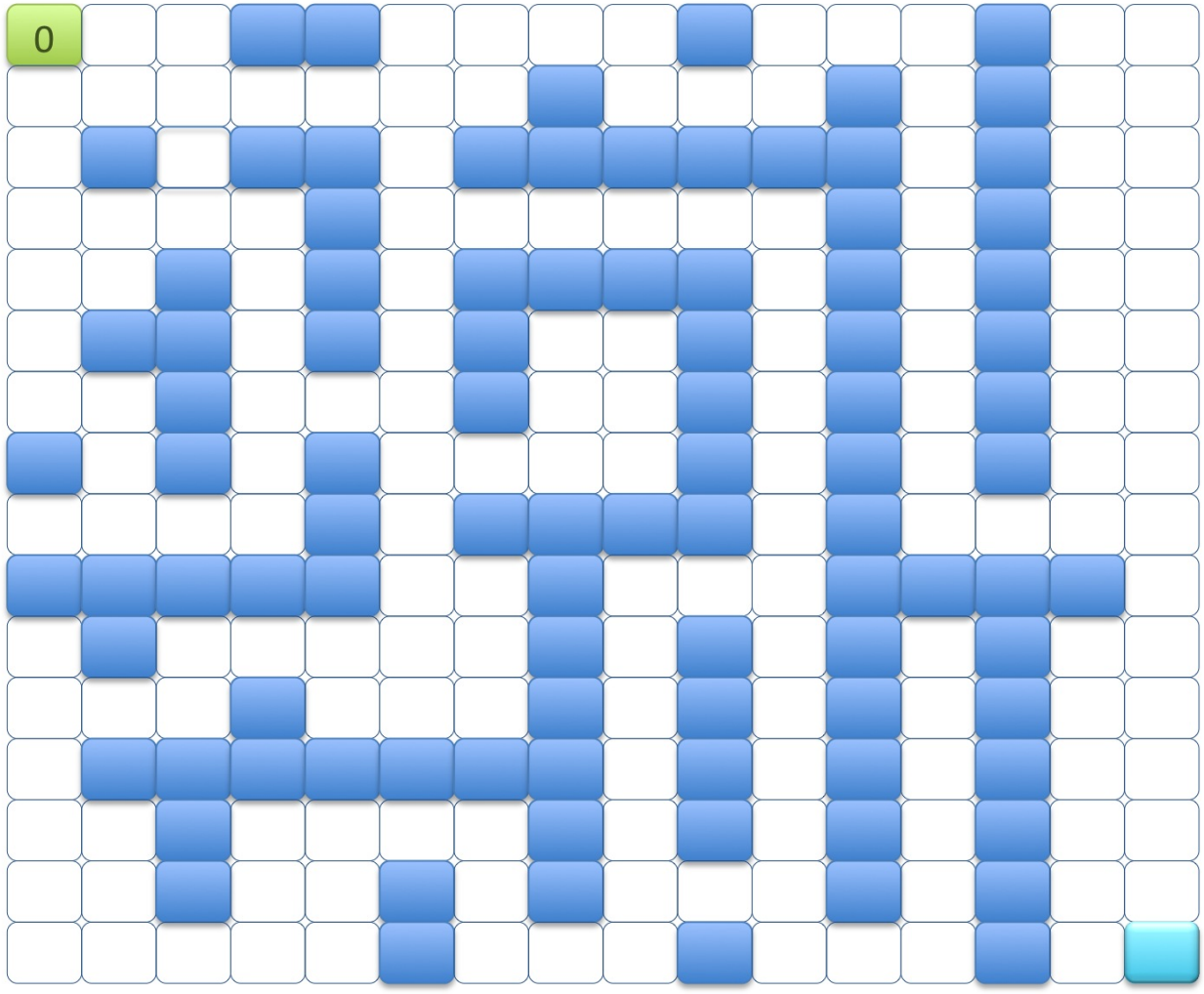
Non-uniform Costs Search

- Dijkstra:
  - breadth-first
  - $g(n)$: cost of reaching n from root
  - chooses node with least $g(n)$

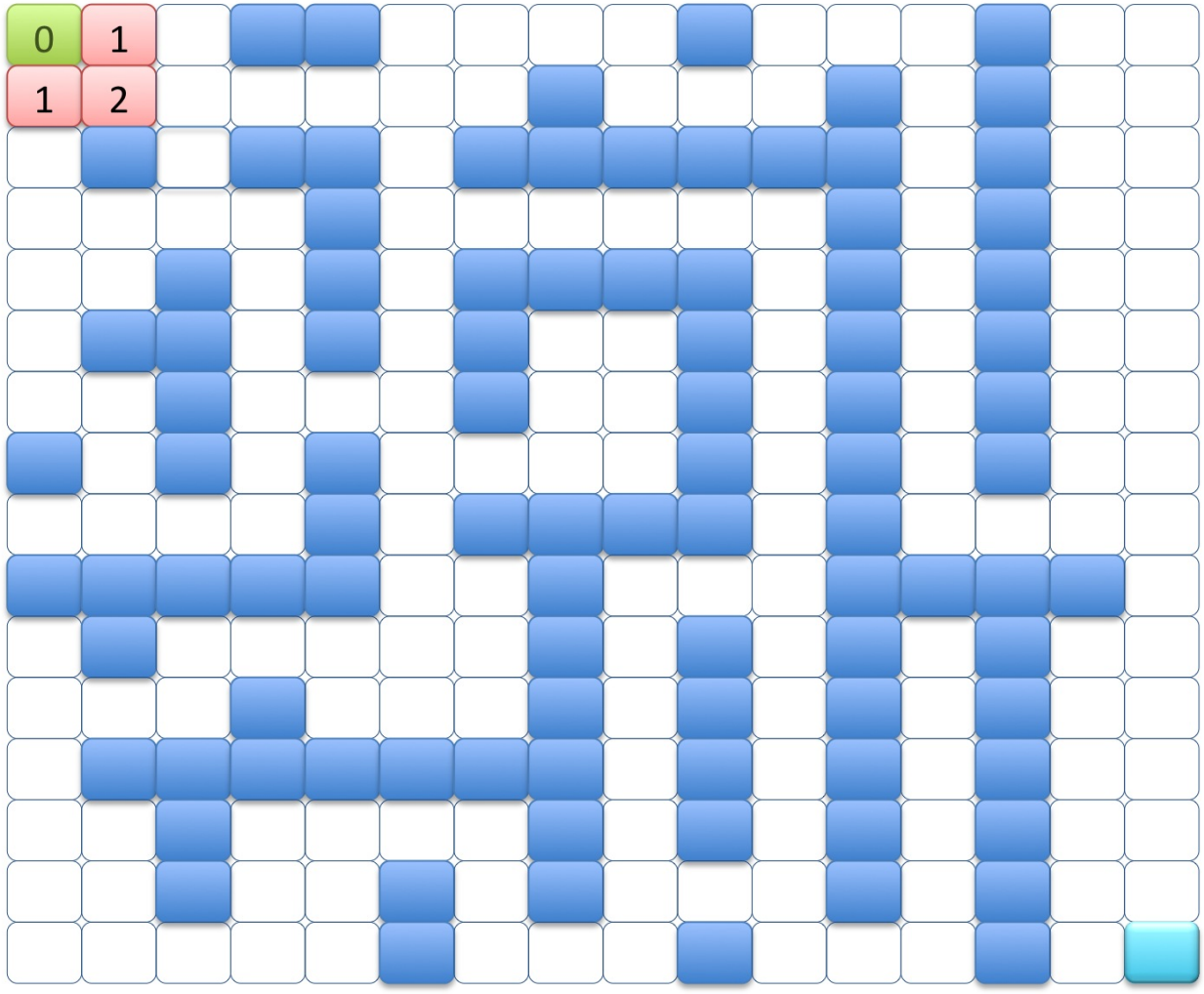
Non-uniform Costs Search

- Dijkstra:
  - breadth-first
  - $g(n)$: cost of reaching n from root
  - chooses node with least $g(n)$
- Branch and bound, B&B
  - search (usually depth-first)
  - when a solution is found, its cost is used as a limit (top or bottom) for the subsequent nodes
  - the search is abandoned if a successive node has a cost from the root greater/less (or equal) to the limit

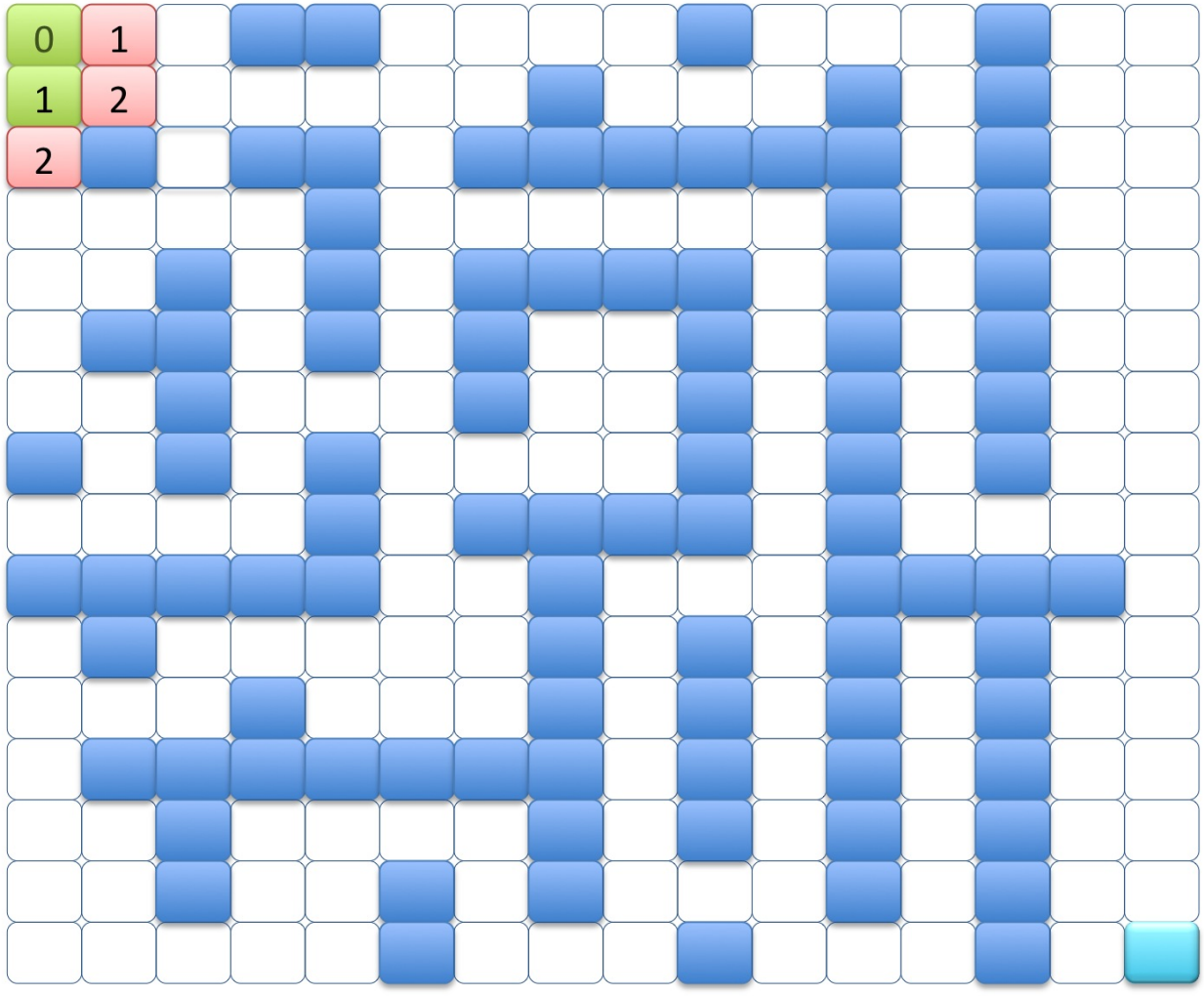
Labyrinth



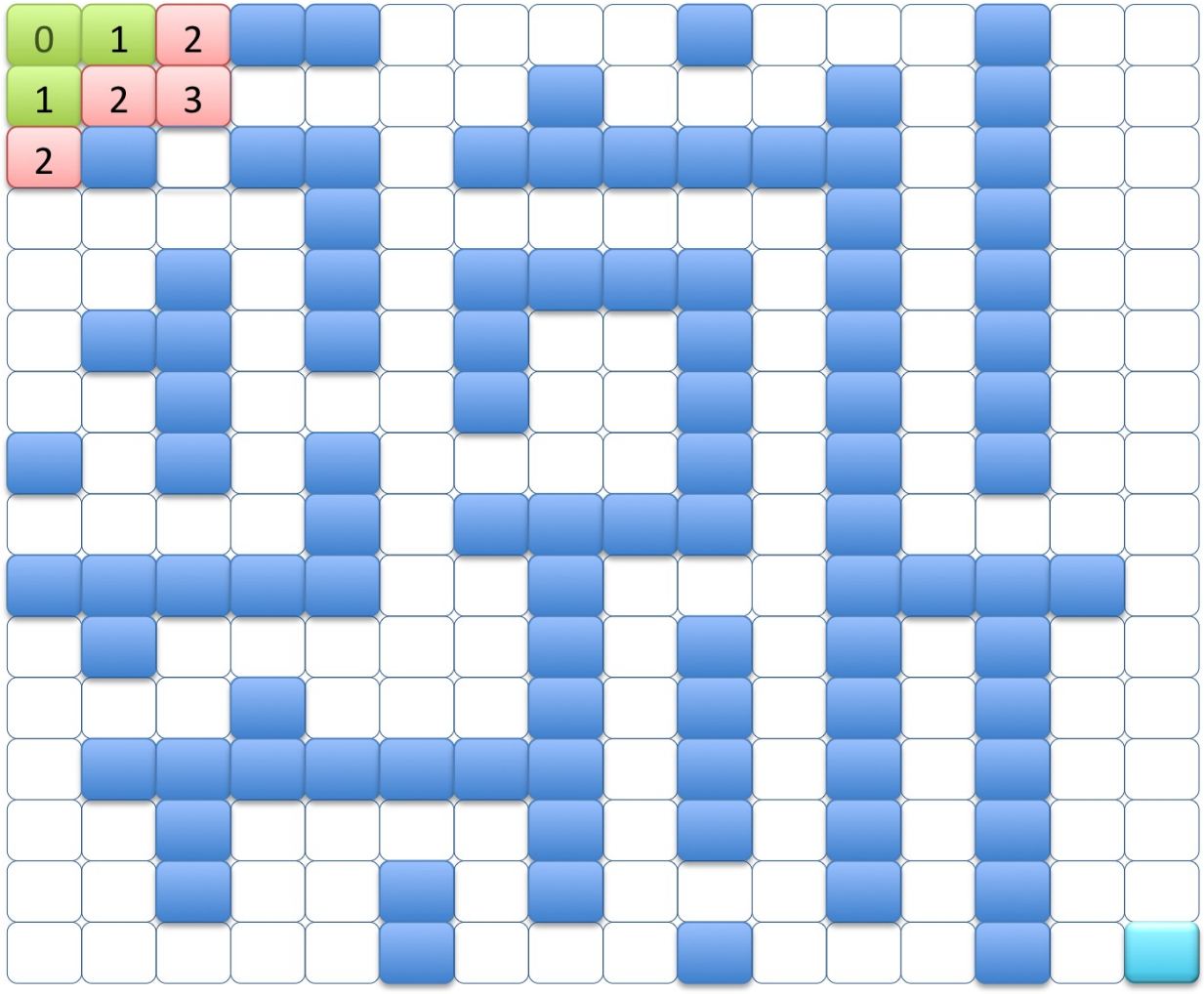
Labyrinth



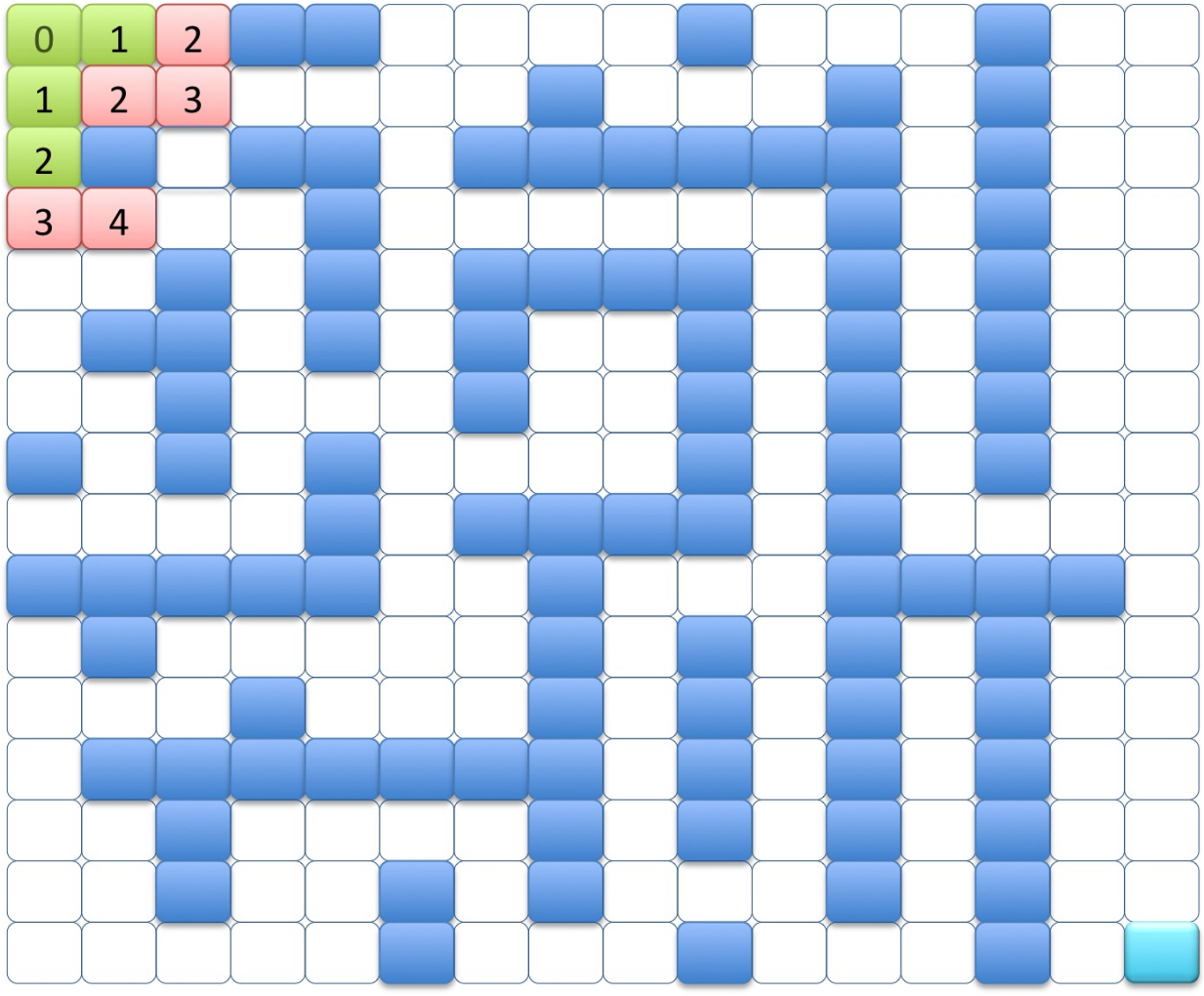
Labyrinth



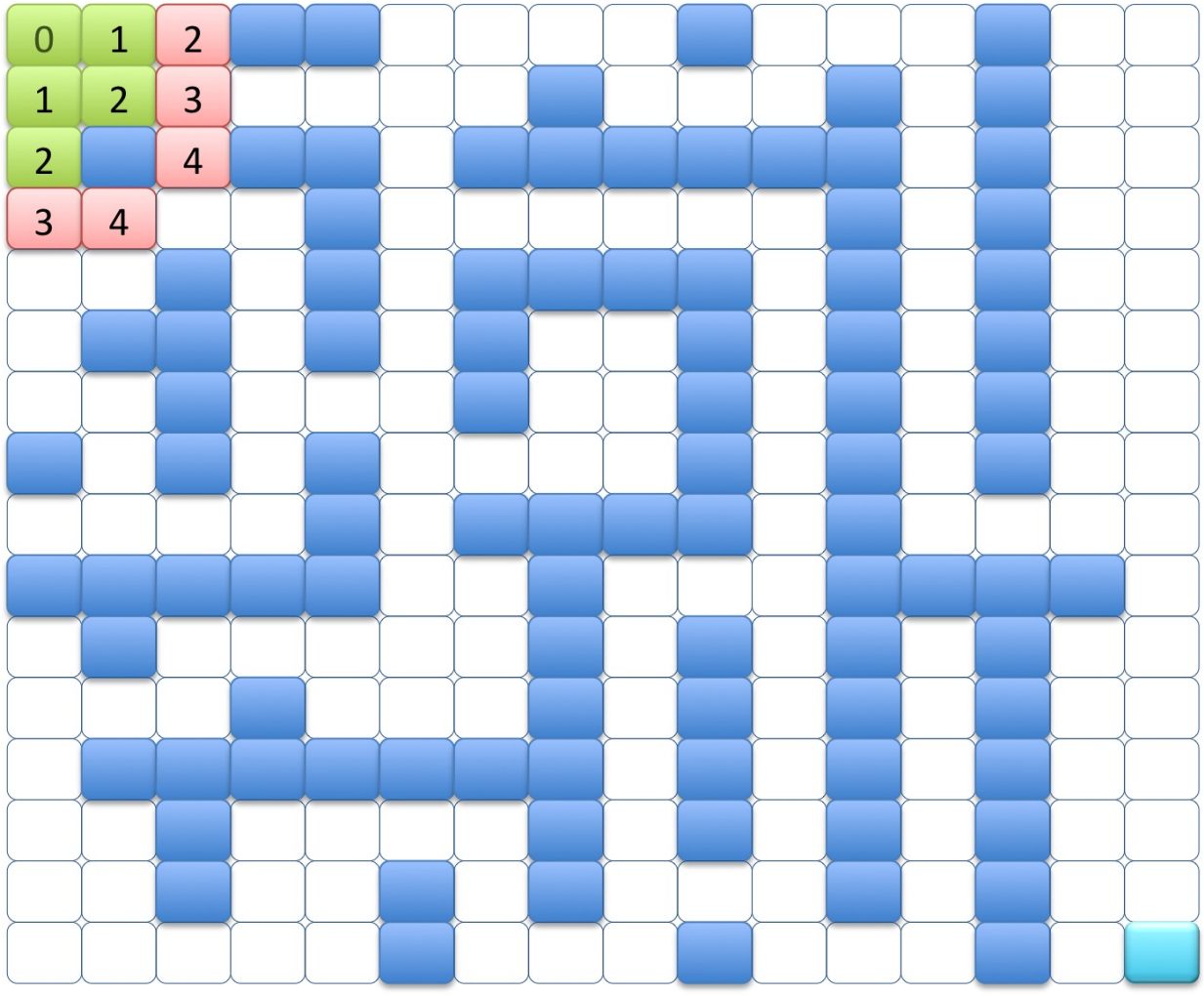
Labyrinth



Labyrinth



Labyrinth



Complexity Analysis

Search method	Time complexity	Space complexity
Breadth-first	$O(b^d)$	$O(b^d)$
Depth-first	$O(b^d)$	$O(d)$

- When the time and space complexities are equal, the memory is exhausted before time bound

Outline

Outline

Heuristics

- If you lack knowledge $\longrightarrow$ uninformed search
- If you have perfect knowledge $\longrightarrow$ correct algorithm
- Most problems solved by humans are in the middle

Heuristics

- If you lack knowledge $\longrightarrow$ uninformed search
- If you have perfect knowledge $\longrightarrow$ correct algorithm
- Most problems solved by humans are in the middle
- Heuristic: (from the Greek "*heurisko*" ($\varepsilon\upsilon\rho\acute{\iota}\sigma\kappa\omega$): "**i find**") partial problem/domain knowledge that allows for efficient resolution of problems related with the problem/domain
- Representation of heuristics
  - meta-rules
  - functions $h(n)$

Heuristics

- If you lack knowledge $\longrightarrow$ uninformed search
- If you have perfect knowledge $\longrightarrow$ correct algorithm
- Most problems solved by humans are in the middle
- Heuristic: (from the Greek "*heurisko*" ($\varepsilon\upsilon\rho\acute{\iota}\sigma\kappa\omega$): "**i find**") partial problem/domain knowledge that allows for efficient resolution of problems related with the problem/domain
- Representation of heuristics
  - meta-rules
  - functions $h(n)$
- How can I generate them?
  - optimal solution of simplified (relaxed) problems

Relaxed problems. 8-puzzle

- Constraints
  - only the blank cell can be moved
  - movement can only be done to either horizontally or vertically adjacent cells
  - at each step, only two cells are taken into account
- Relaxations
  - if first two constraints are removed

Relaxed problems. 8-puzzle

- Constraints
  - only the blank cell can be moved
  - movement can only be done to either horizontally or vertically adjacent cells
  - at each step, only two cells are taken into account
- Relaxations
  - if first two constraints are removed
    heuristic is number of misplaced tiles
  - if first constraint is removed
    heuristic is Manhattan distance

Relaxation and 8-puzzle. Misplaced digits

- In the N-Puzzle, there is only one operator:

  Move(x,y,z): moves digit x, that is in position y to position z
  if On(x,y), Free(z), Adjacent(y,z)
  then On(x,z), Free(y), NOT On(x,y), NOT Free(z)

- We can relax Move in many different ways: combinations of removing the three preconditions

Relaxation and 8-puzzle. Misplaced digits

- In the N-Puzzle, there is only one operator:

  Move(x,y,z): moves digit x, that is in position y to position z
  if On(x,y), Free(z), Adjacent(y,z)
  then On(x,z), Free(y), NOT On(x,y), NOT Free(z)

- We can relax Move in many different ways: combinations of removing the three preconditions
- One alternative: we relax Free(z) and Adjacent(y,z)

Relaxation and 8-puzzle. Misplaced digits

- In the N-Puzzle, there is only one operator:

  Move(x,y,z): moves digit x, that is in position y to position z
  if On(x,y), Free(z), Adjacent(y,z)
  then On(x,z), Free(y), NOT On(x,y), NOT Free(z)

- We can relax Move in many different ways: combinations of removing the three preconditions
- One alternative: we relax Free(z) and Adjacent(y,z)

  Move1(x,y,z): moves digit x, that is in position y to position z
  if On(x,y)
  then On(x,z), NOT On(x,y)

Relaxation and 8-puzzle. Misplaced digits

- In the N-Puzzle, there is only one operator:

  Move(x,y,z): moves digit x, that is in position y to position z
  if On(x,y), Free(z), Adjacent(y,z)
  then On(x,z), Free(y), NOT On(x,y), NOT Free(z)

- We can relax Move in many different ways: combinations of removing the three preconditions
- One alternative: we relax Free(z) and Adjacent(y,z)

  Move1(x,y,z): moves digit x, that is in position y to position z
  if On(x,y)
  then On(x,z), NOT On(x,y)

- Optimal solution:

Relaxation and 8-puzzle. Misplaced digits

- In the N-Puzzle, there is only one operator:

  Move(x,y,z): moves digit x, that is in position y to position z
  if On(x,y), Free(z), Adjacent(y,z)
  then On(x,z), Free(y), NOT On(x,y), NOT Free(z)

- We can relax Move in many different ways: combinations of removing the three preconditions
- One alternative: we relax Free(z) and Adjacent(y,z)

  Move1(x,y,z): moves digit x, that is in position y to position z
  if On(x,y)
  then On(x,z), NOT On(x,y)

- Optimal solution: misplaced digits

Relaxation and 8-puzzle. Manhattan distance

- We relax only Free(z)

  Move2(x,y,z): moves digit x, that is in position y to position z
  if On(x,y), Adjacent(y,z)
  then On(x,z), NOT On(x,y)

- Optimal solution:

Relaxation and 8-puzzle. Manhattan distance

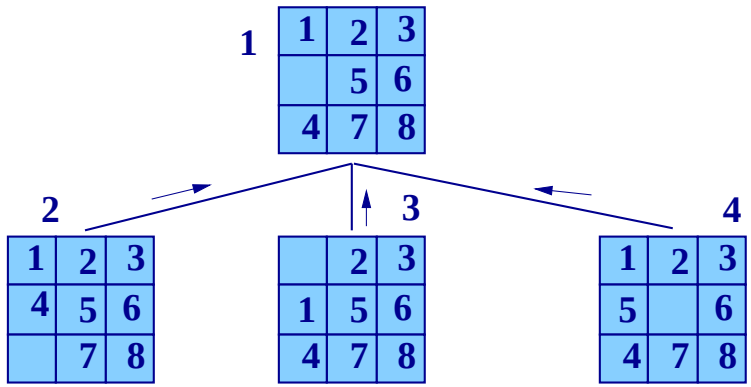
- We relax only Free(z)

  Move2(x,y,z): moves digit x, that is in position y to position z
  if On(x,y), Adjacent(y,z)
  then On(x,z), NOT On(x,y)

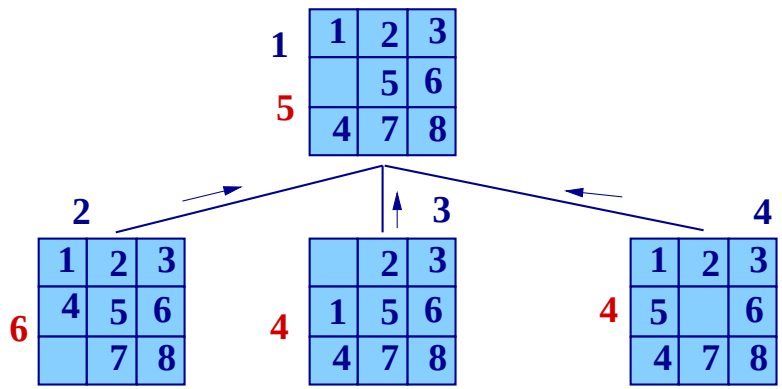
- Optimal solution: Manhattan distance

Outline

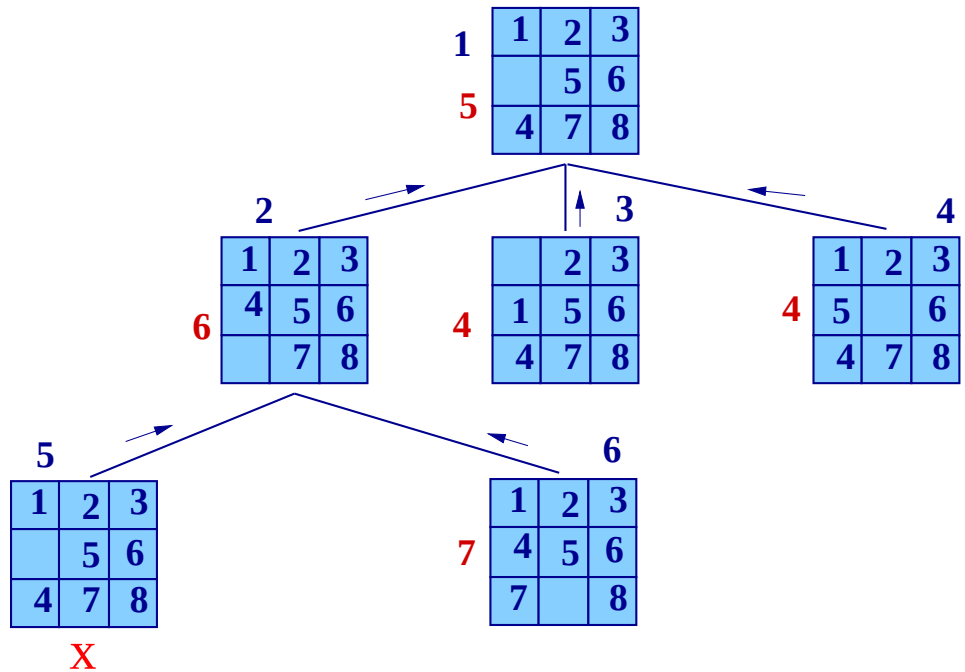
8-Puzzle – Hill-climbing



8-Puzzle – Hill-climbing



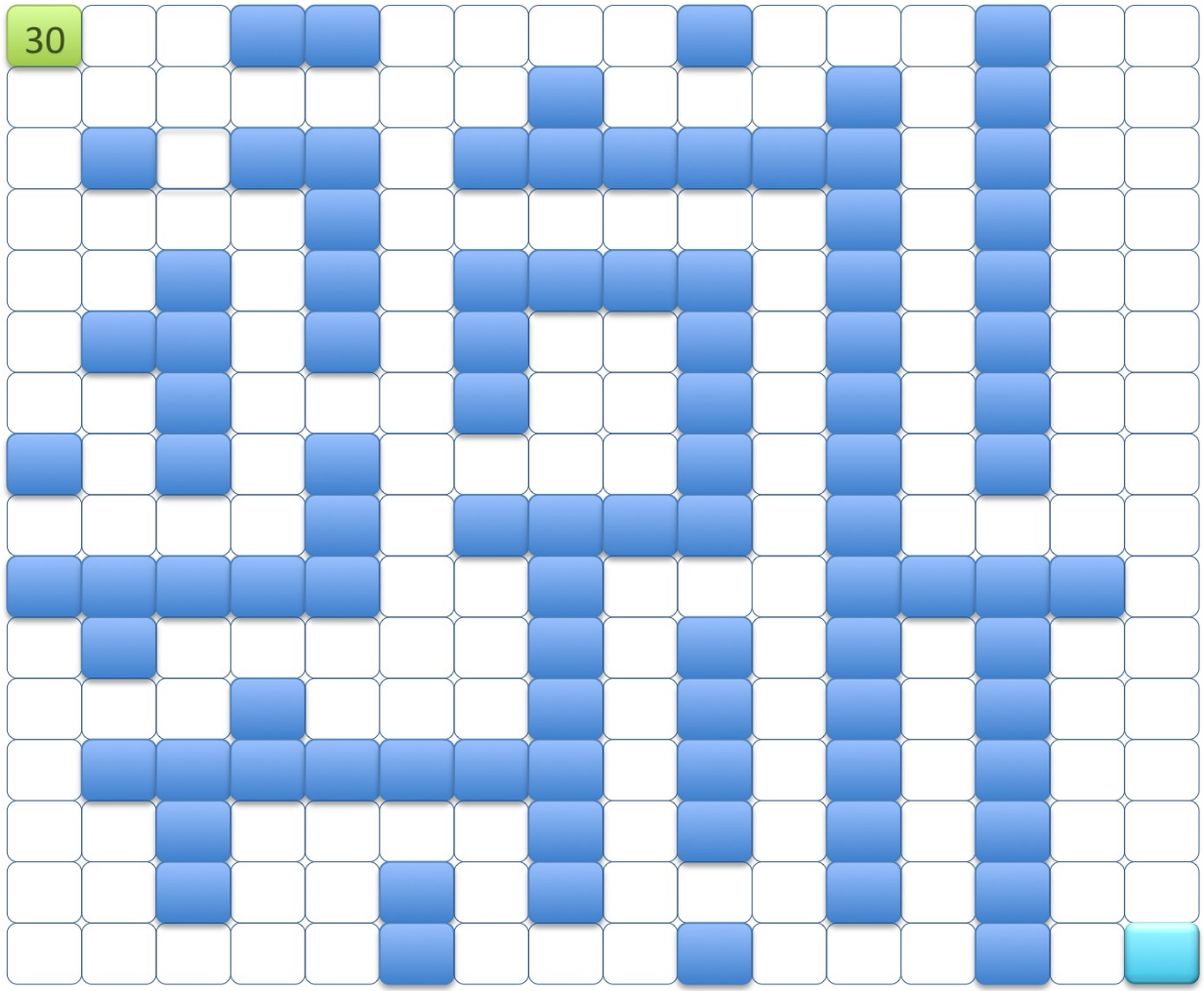
8-Puzzle – Hill-climbing



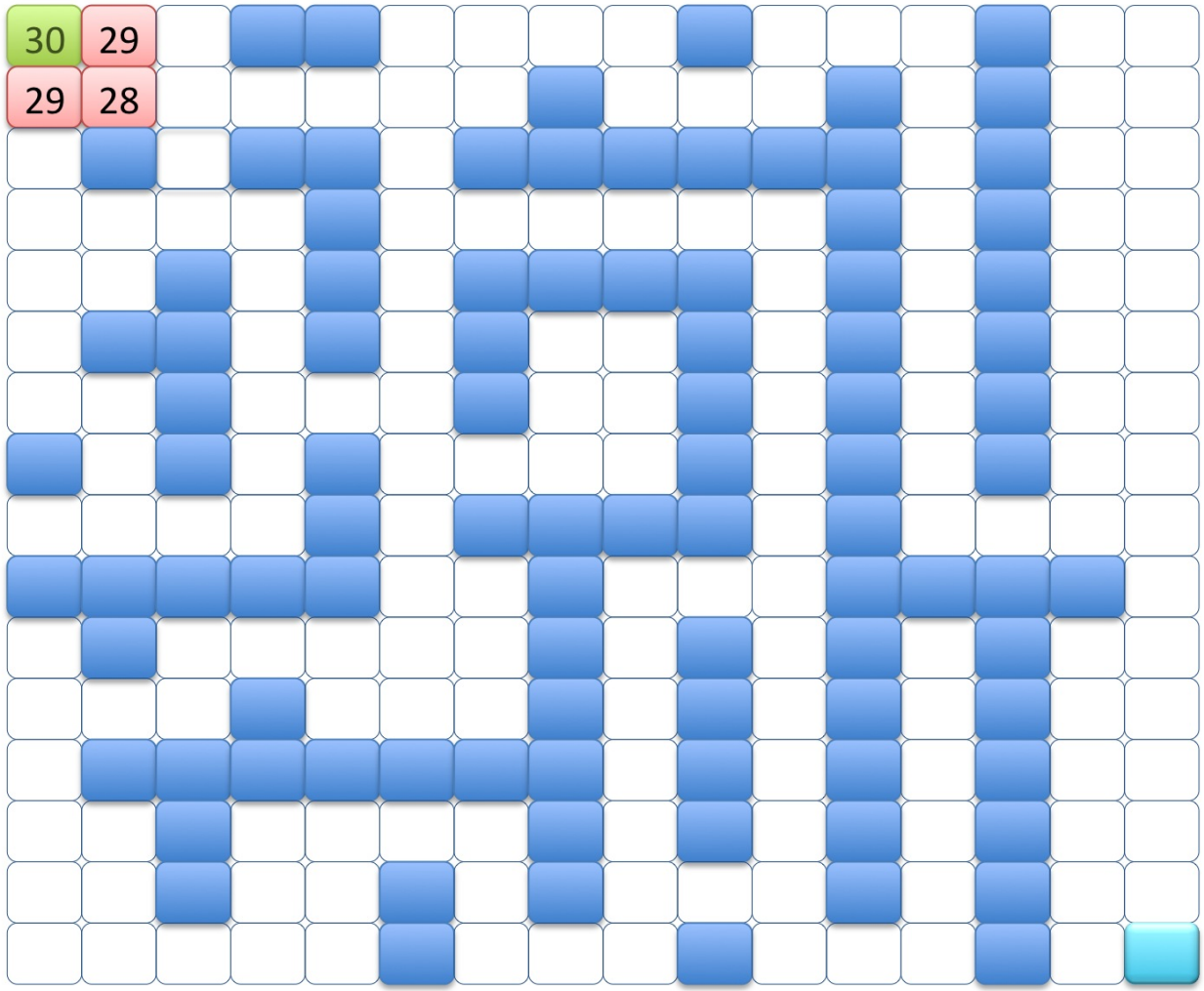
Hill-climbing search

Procedure Hill-climbing (Initial-state I, Goal-function `Goal`,
Evaluation function $f(\cdot)$)
$n = I$; SOLUTION=False; STOP=False
Until STOP or SOLUTION
    Generate the successors S of n
    If $\exists s \in S$ such that `Goal`(s)=True
    Then SOLUTION=True
    Else Evaluate each successor with $f(\cdot)$
        BS=best successor
        If $f(BS)$ better than $f(n)$
        Then $n = BS$
        Else STOP=True [or backtrack or continue]
If SOLUTION=True
Then RETURN path from I to solution node
Else RETURN Failure

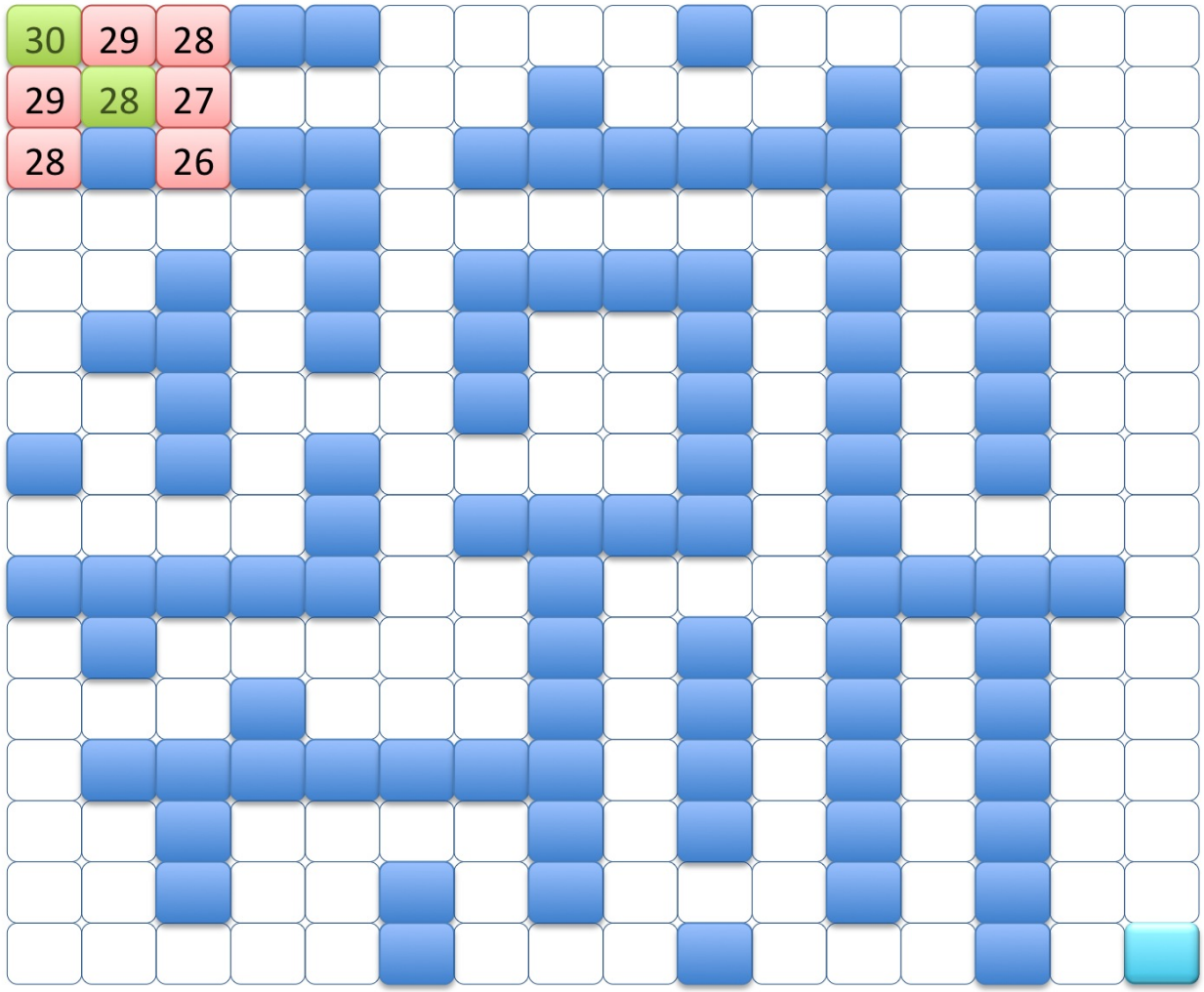
Labyrinth. $h(n = (x_i, y_i), m = (x_m, y_m)) = |x_i - x_m| + |y_i - y_m|$



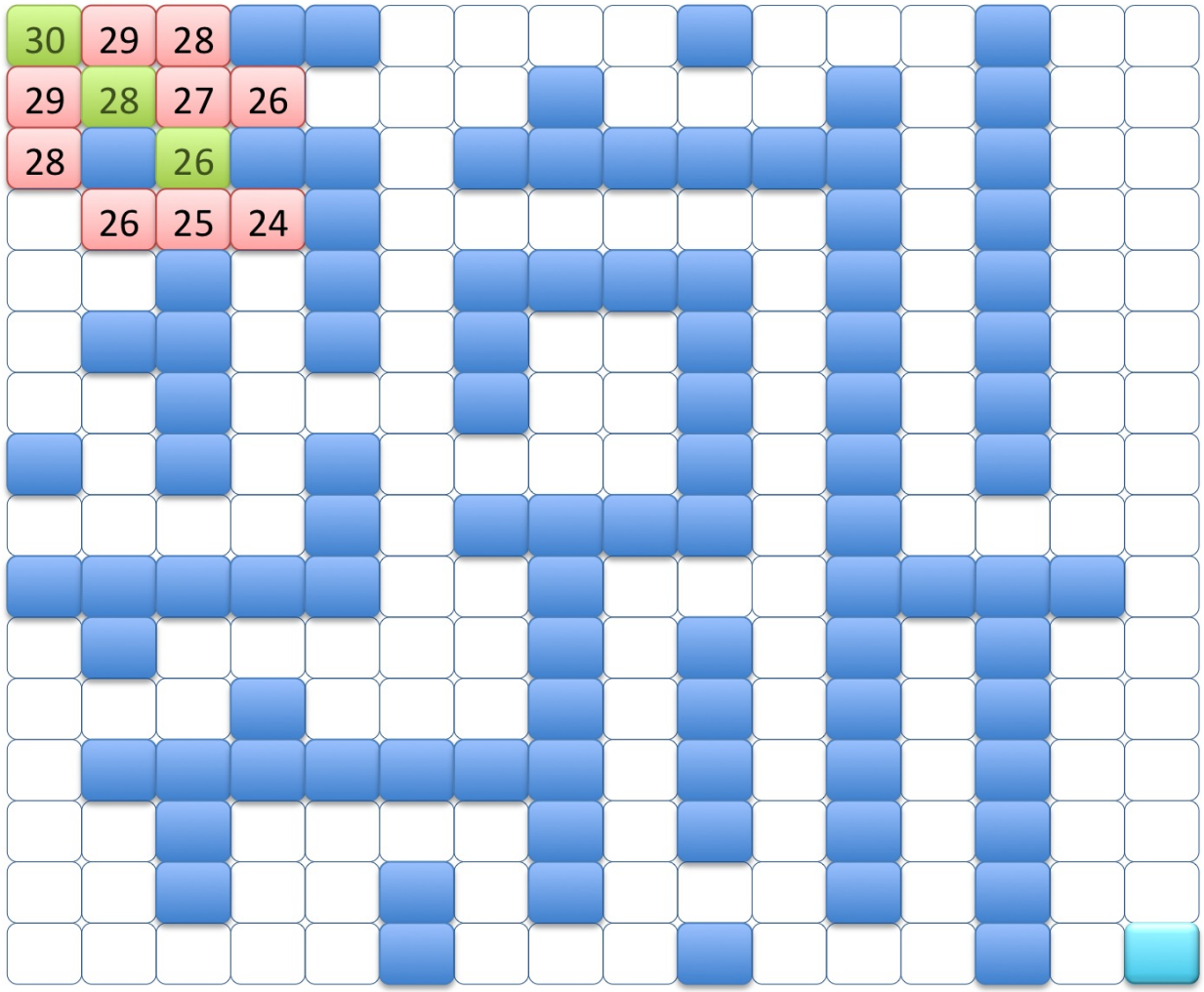
Labyrinth. $h(n = (x_i, y_i), m = (x_m, y_m)) = |x_i - x_m| + |y_i - y_m|$



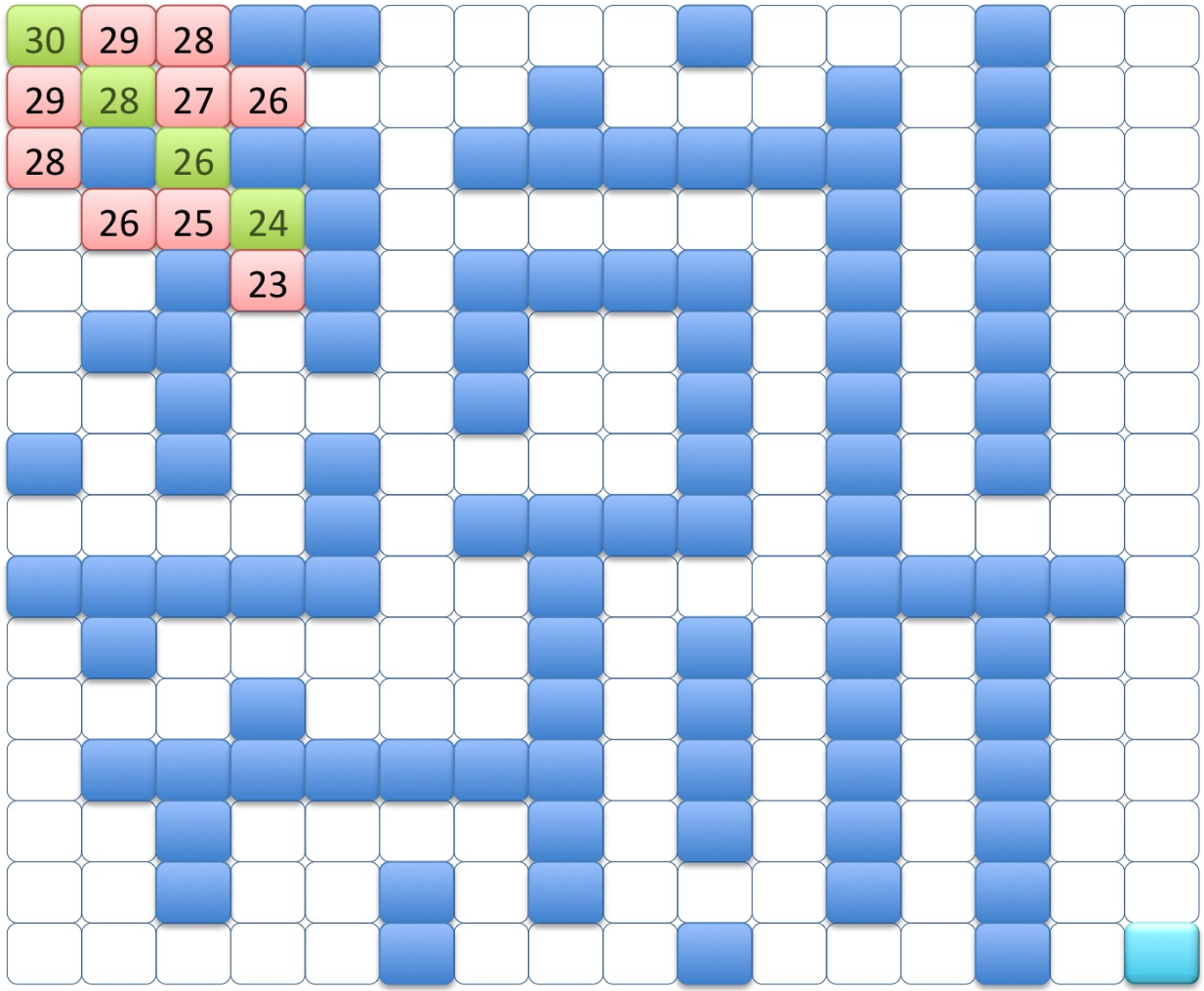
Labyrinth. $h(n = (x_i, y_i), m = (x_m, y_m)) = |x_i - x_m| + |y_i - y_m|$



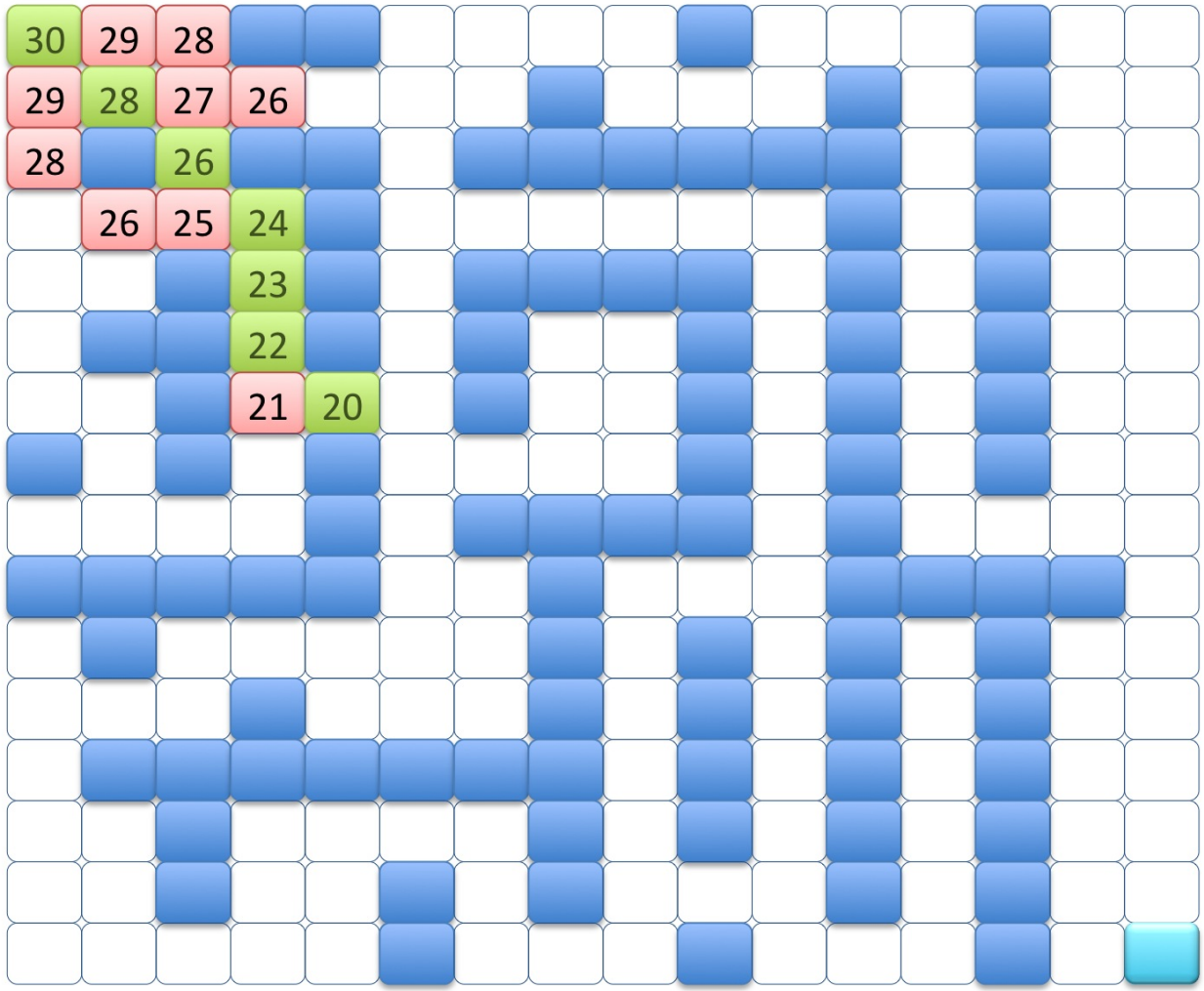
Labyrinth. $h(n = (x_i, y_i), m = (x_m, y_m)) = |x_i - x_m| + |y_i - y_m|$



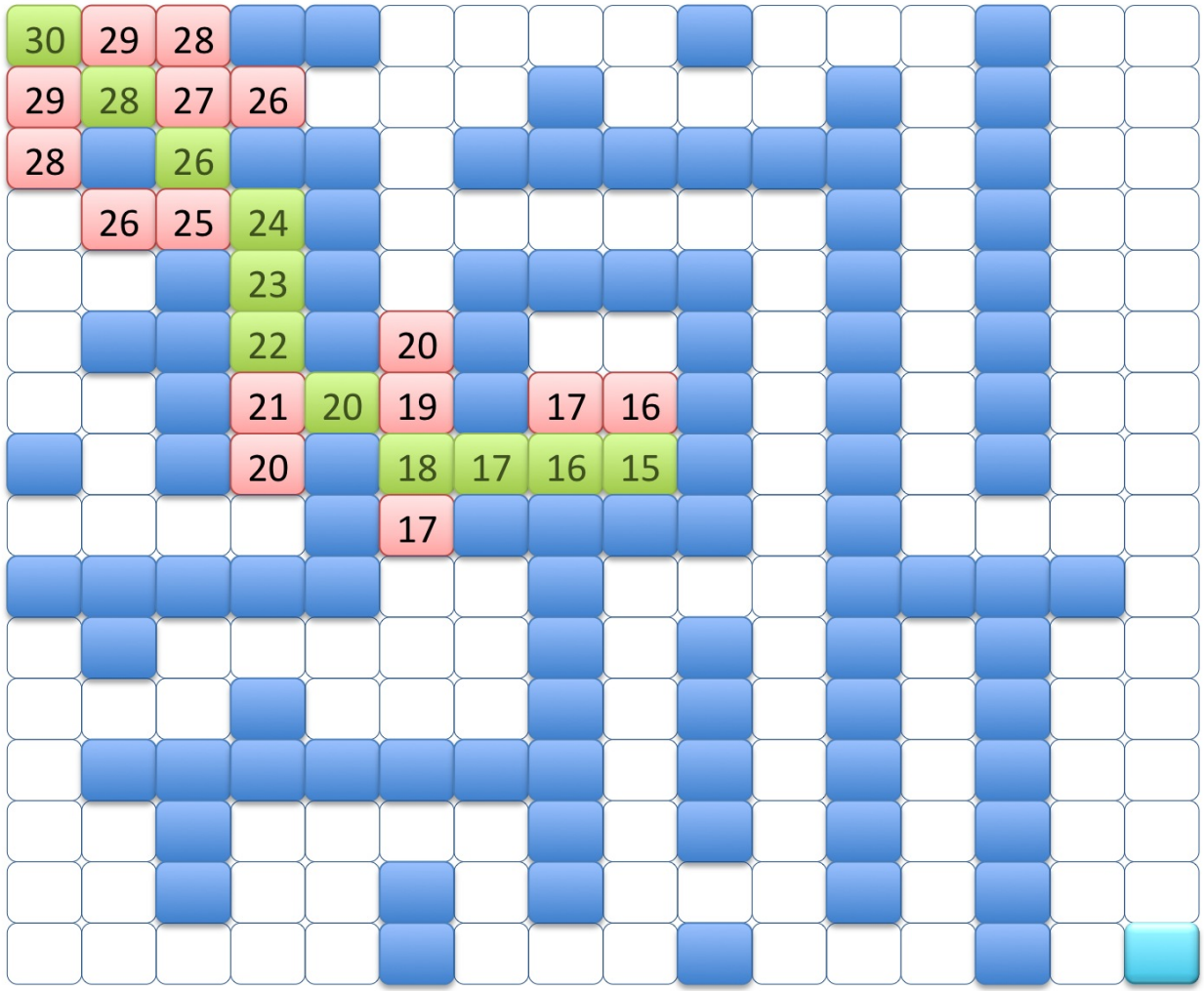
Labyrinth. $h(n = (x_i, y_i), m = (x_m, y_m)) = |x_i - x_m| + |y_i - y_m|$



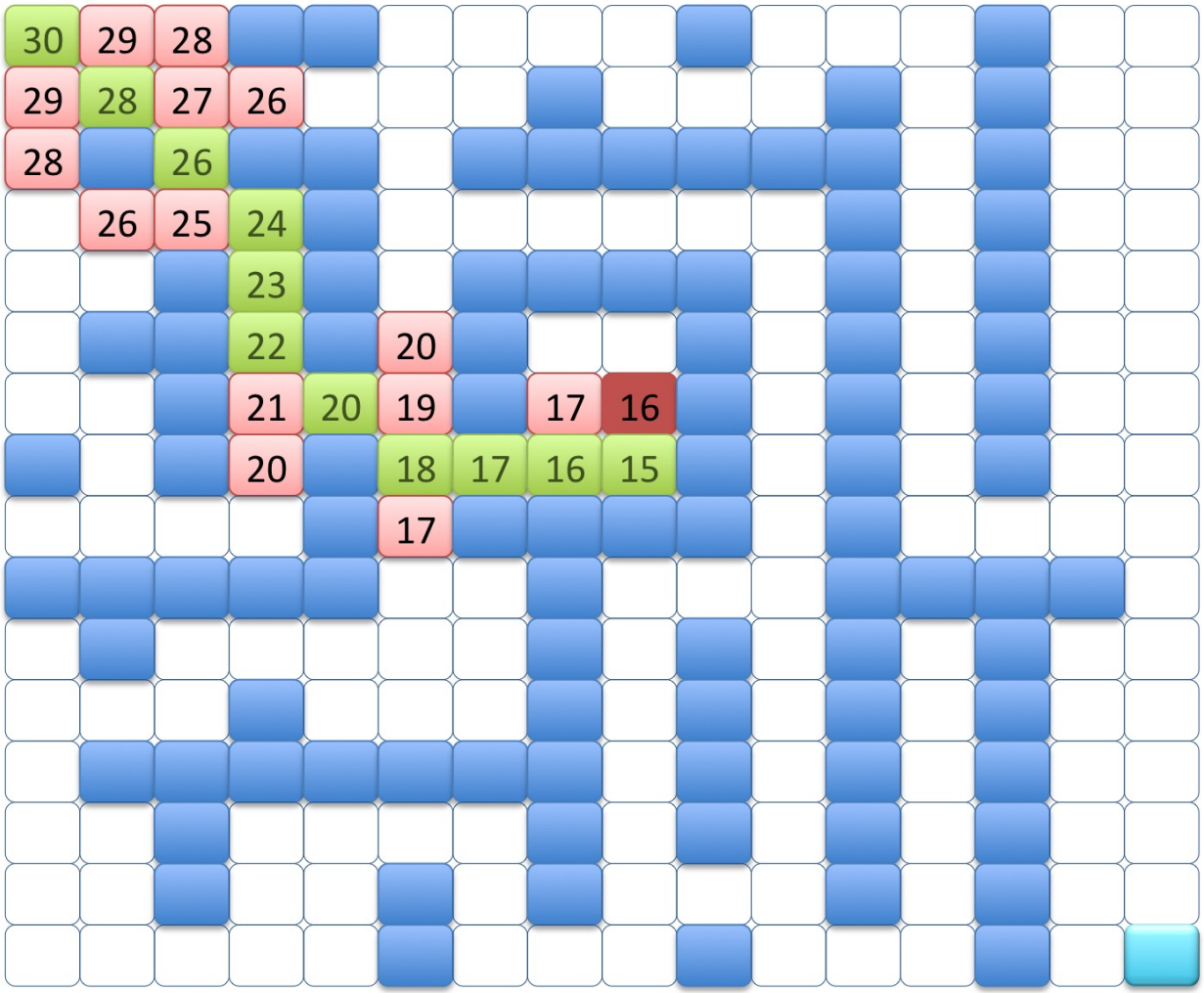
Labyrinth. $h(n = (x_i, y_i), m = (x_m, y_m)) = |x_i - x_m| + |y_i - y_m|$



Labyrinth. $h(n = (x_i, y_i), m = (x_m, y_m)) = |x_i - x_m| + |y_i - y_m|$



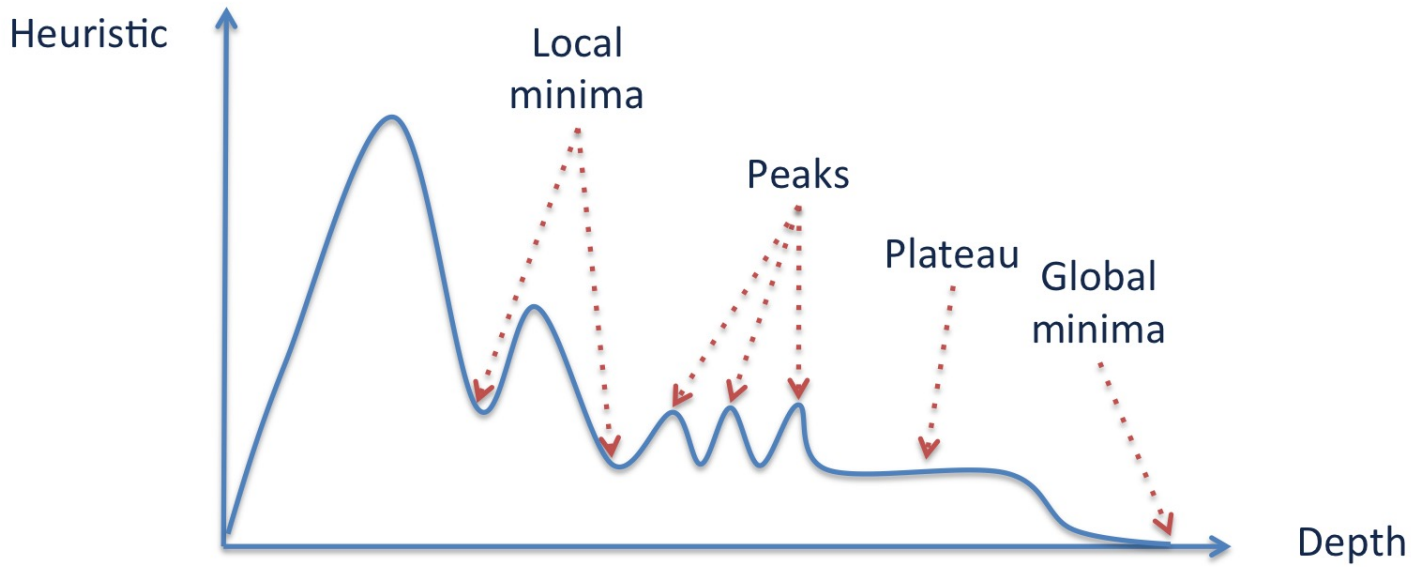
Labyrinth. $h(n = (x_i, y_i), m = (x_m, y_m)) = |x_i - x_m| + |y_i - y_m|$



Characteristics

Problems of *greedy* methods

- Local Maximum (or minimum)
- Plateaux
- Peaks



Characteristics

- Solutions
  - backtrack
  - perform more than one step
  - restart with random node selection

Characteristics

- Solutions
  - backtrack
  - perform more than one step
  - restart with random node selection
- Properties
  - completeness: no
  - admissibility: no
  - efficiency: fast and useful if the heuristic function is monotonically increasing/decreasing

Outline

Best-first search

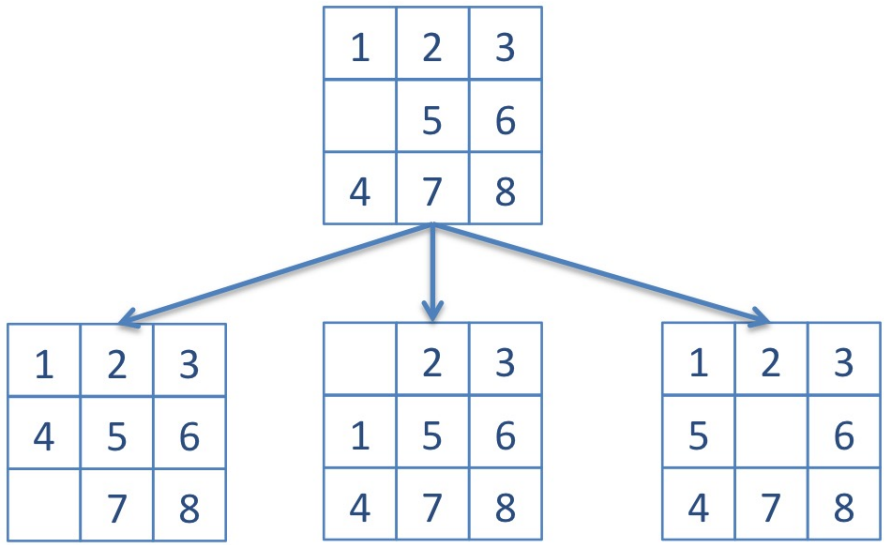
Procedure Best-first (Initial-state I, Goal function `Goal`)

OPEN=$\{I\}$, CLOSED=$\emptyset$, SOLUTION=False
Until OPEN is empty or SOLUTION=True
    Remove the first (best) node in OPEN, n, and place it in CLOSED
    If `Goal`(n) =True then SOLUTION=True
    Else Expand n, generating the set S of successors of n,
                that are not predecessors of n
        For every $s \in S$
            If $s \notin$OPEN$\cup$CLOSED
            Then Add s to OPEN and establish a pointer to n
            Else If $s \in$OPEN
                Then $g(s) = \min\{$previous-$g()$, new-$g()\}$
                Else [$s \in$CLOSED]
                    If new-$g() <$ previous-$g()$
                    Then move s from CLOSED to OPEN [*reopen node*]
        Sort OPEN according to minimum $f(n)$
If SOLUTION=True
Then RETURN path from I to n using pointers
Else RETURN Failure

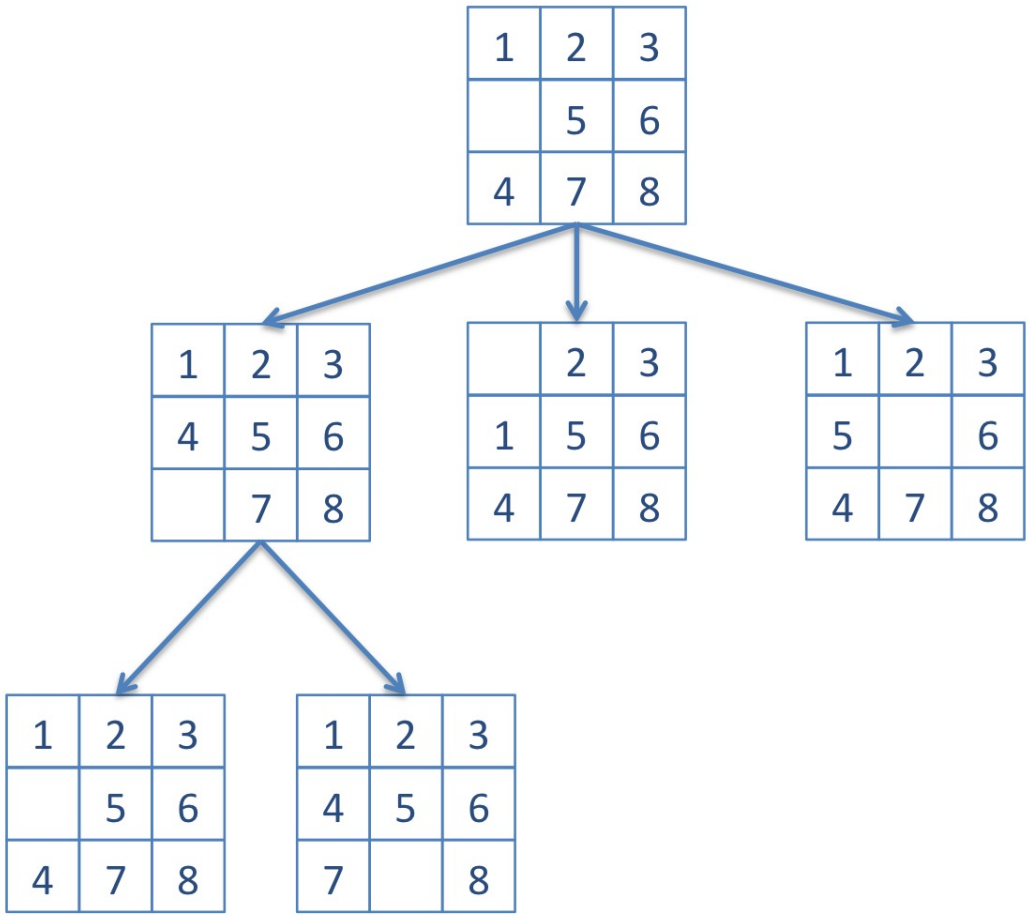
Search

1	2	3
	5	6
4	7	8

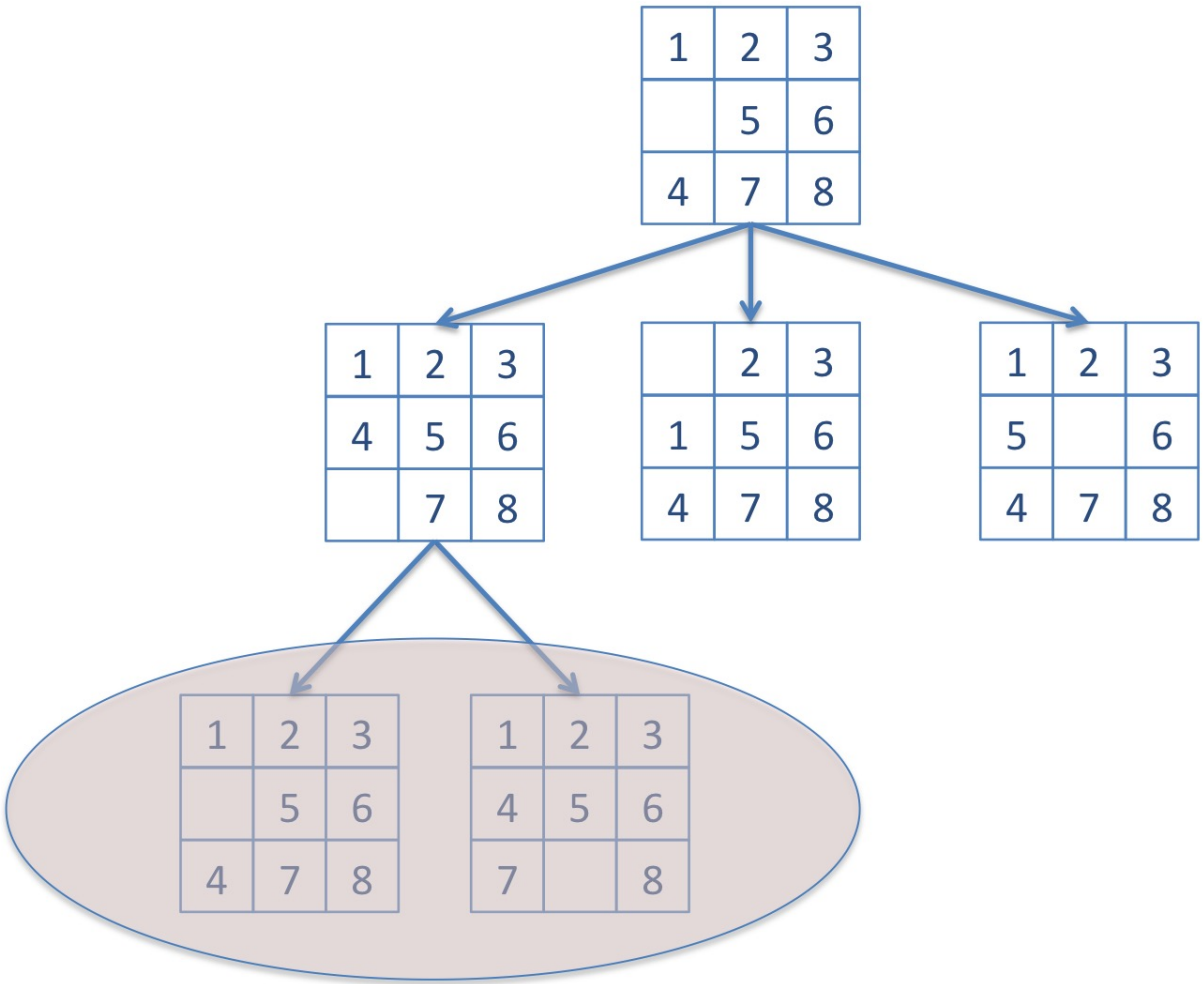
Search



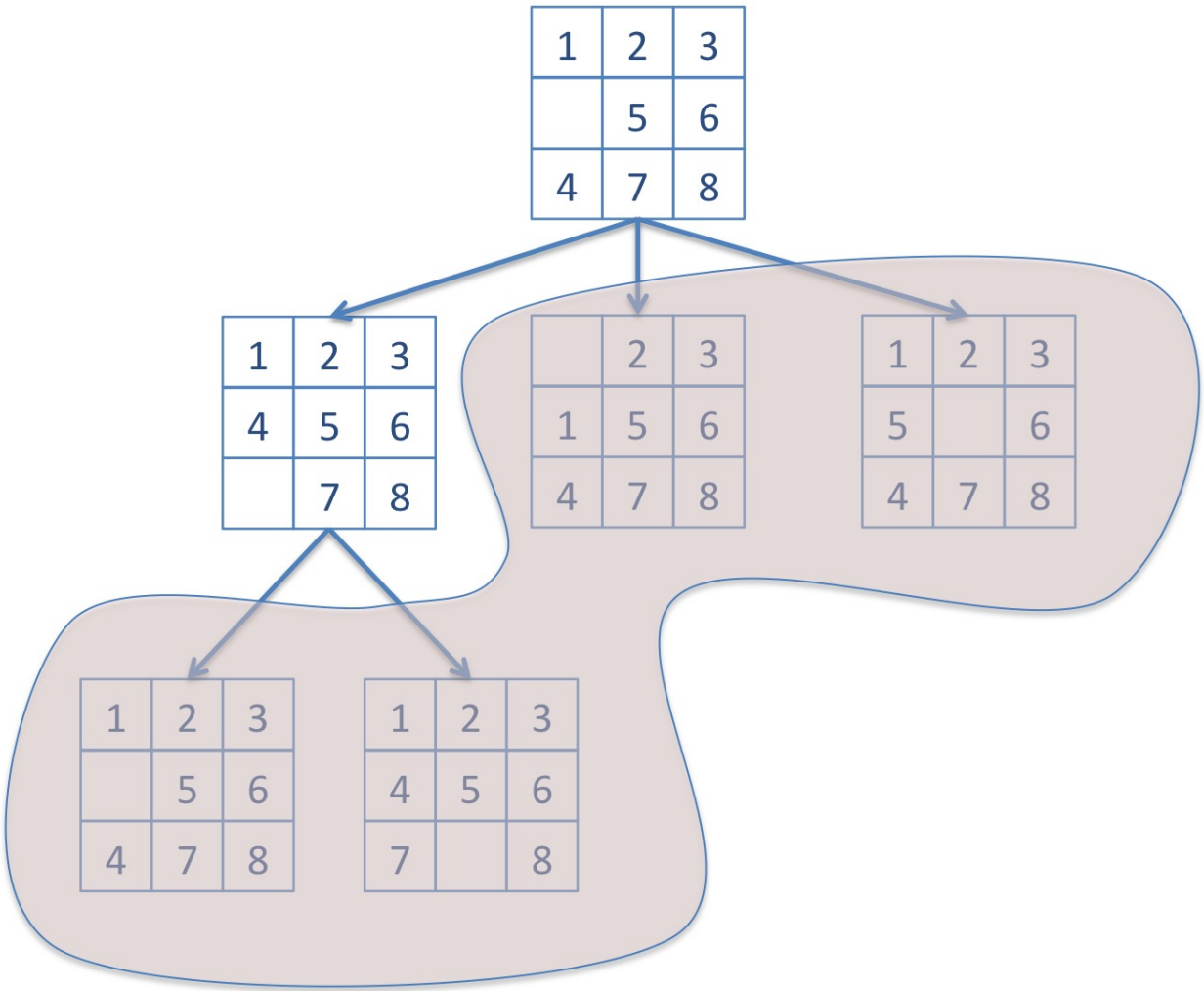
Search



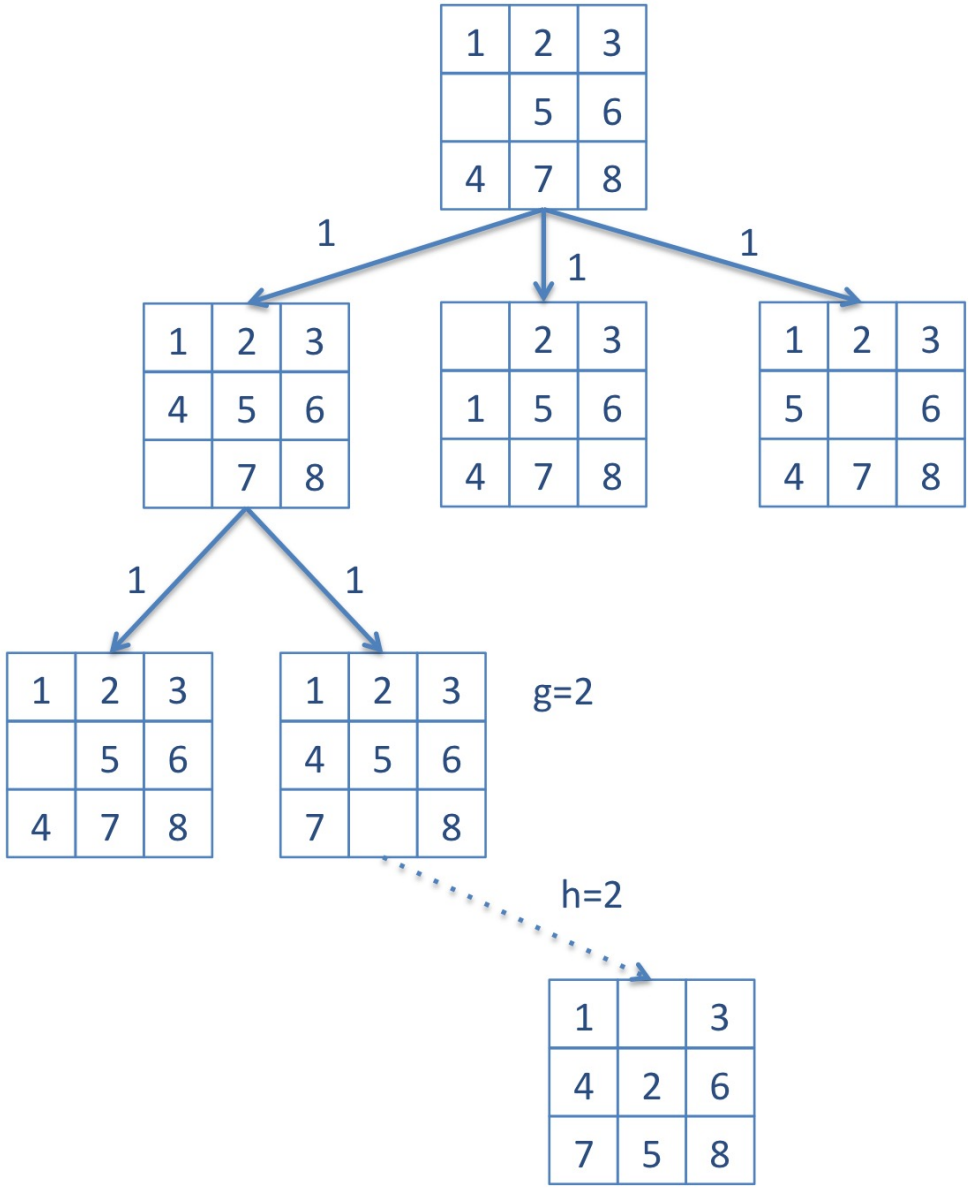
Hill-climbing



Best-first search



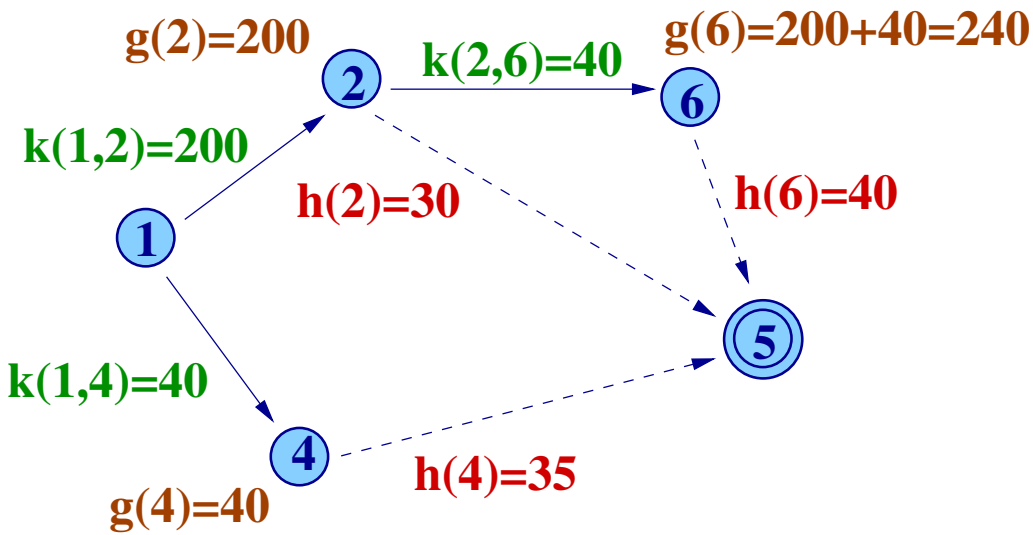
A*


1 2 3
5 6
4 7 8
1
1
1
1 2 3
4 5 6
7 8
2 3
1 5 6
4 7 8
1 2 3
5 6
4 7 8
1
1
1 2 3
5 6
4 7 8
1 2 3
4 5 6
7 8
g=2
h=2
1 3
4 2 6
7 5 8

A* (Hart, Nilsson and Raphael, 1968)

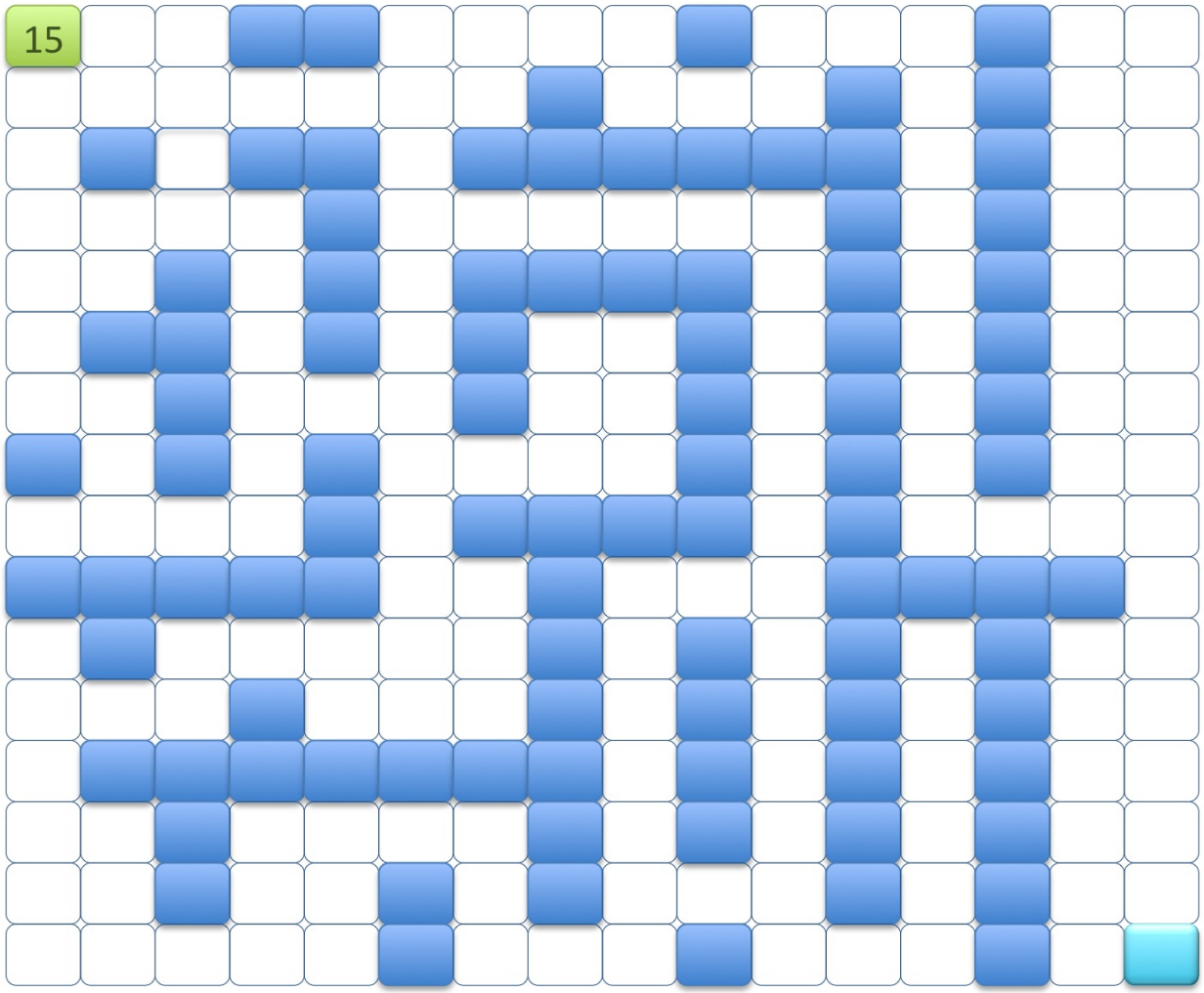
- Evaluation function: $f(n) = g(n) + h(n)$
  - $f(n)$: evaluation function
  - $g(n)$: cost of the path to the n from the initial node
  - $h(n)$: heuristic function, estimated cost from n to any goal node
- $g(n)$ is computed as the sum of the costs of the edges in path, $k(n_i, n_j)$
- The actual optimal values can only be known at the end of the search
  - $f^*(n)$: is the optimal cost of going from the initial node to a goal node through n
  - $g^*(n)$: actual cost going from the initial node to node n
  - $h^*(n)$: actual cost going from node n to a goal node

Definitions



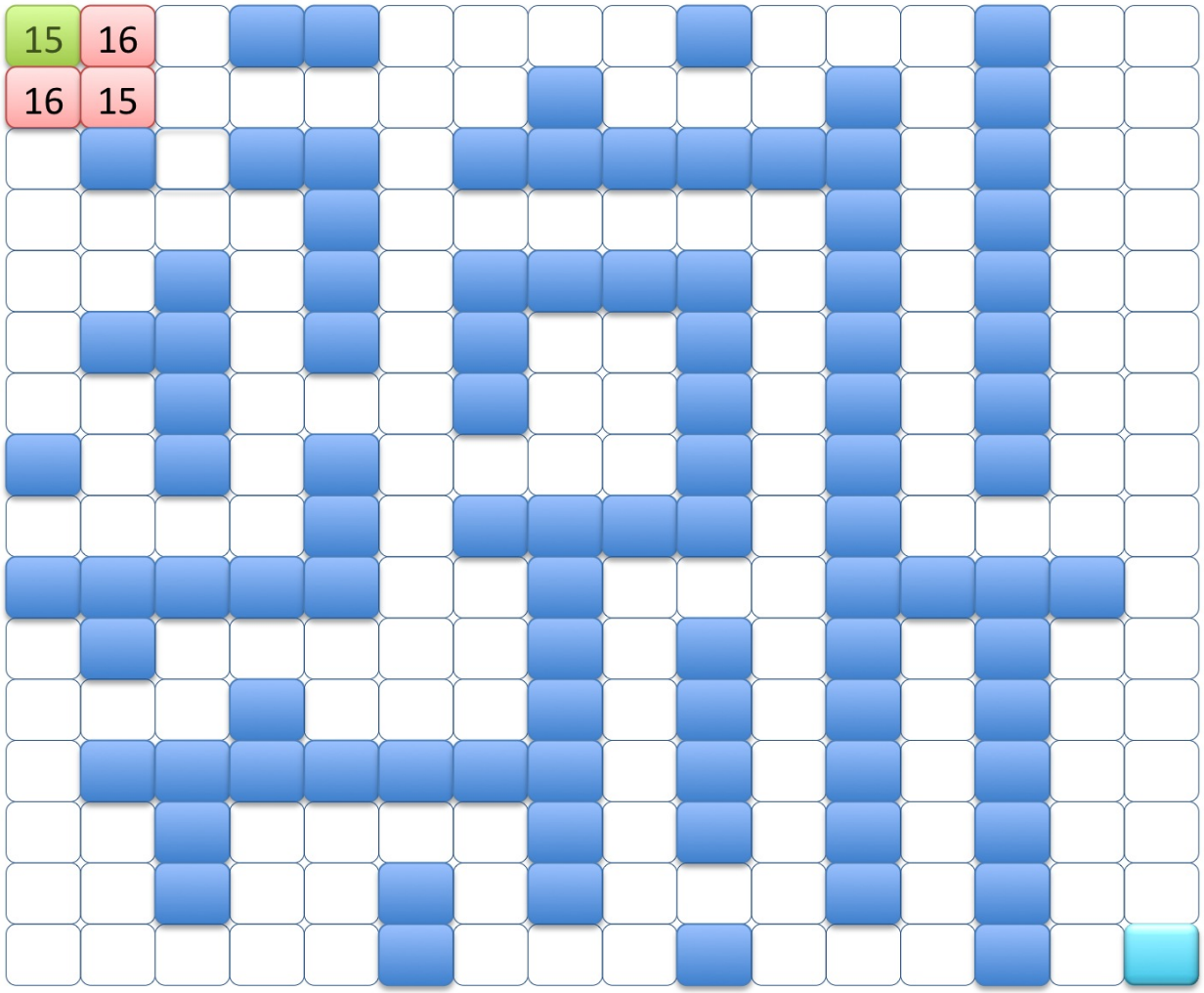
Labyrinth. $h(n = (x_i, y_i), m = (x_m, y_m)) = \max(|x_i - x_m|, |y_i - y_m|)$

f(0)=g(0)+h(0)=0+15=15

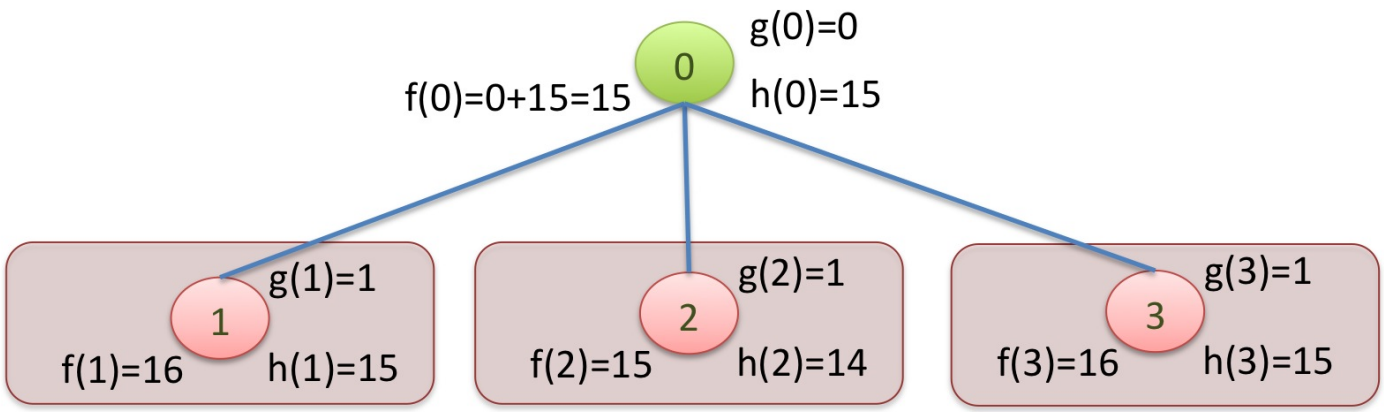


Labyrinth. $h(n = (x_i, y_i), m = (x_m, y_m)) = \max(|x_i - x_m|, |y_i - y_m|)$

f(0)=g(0)+h(0)=0+15=15
f(1)=g(1)+h(1)=1+15=16
f(2)=g(2)+h(2)=1+14=15
f(3)=g(3)+h(3)=1+15=16

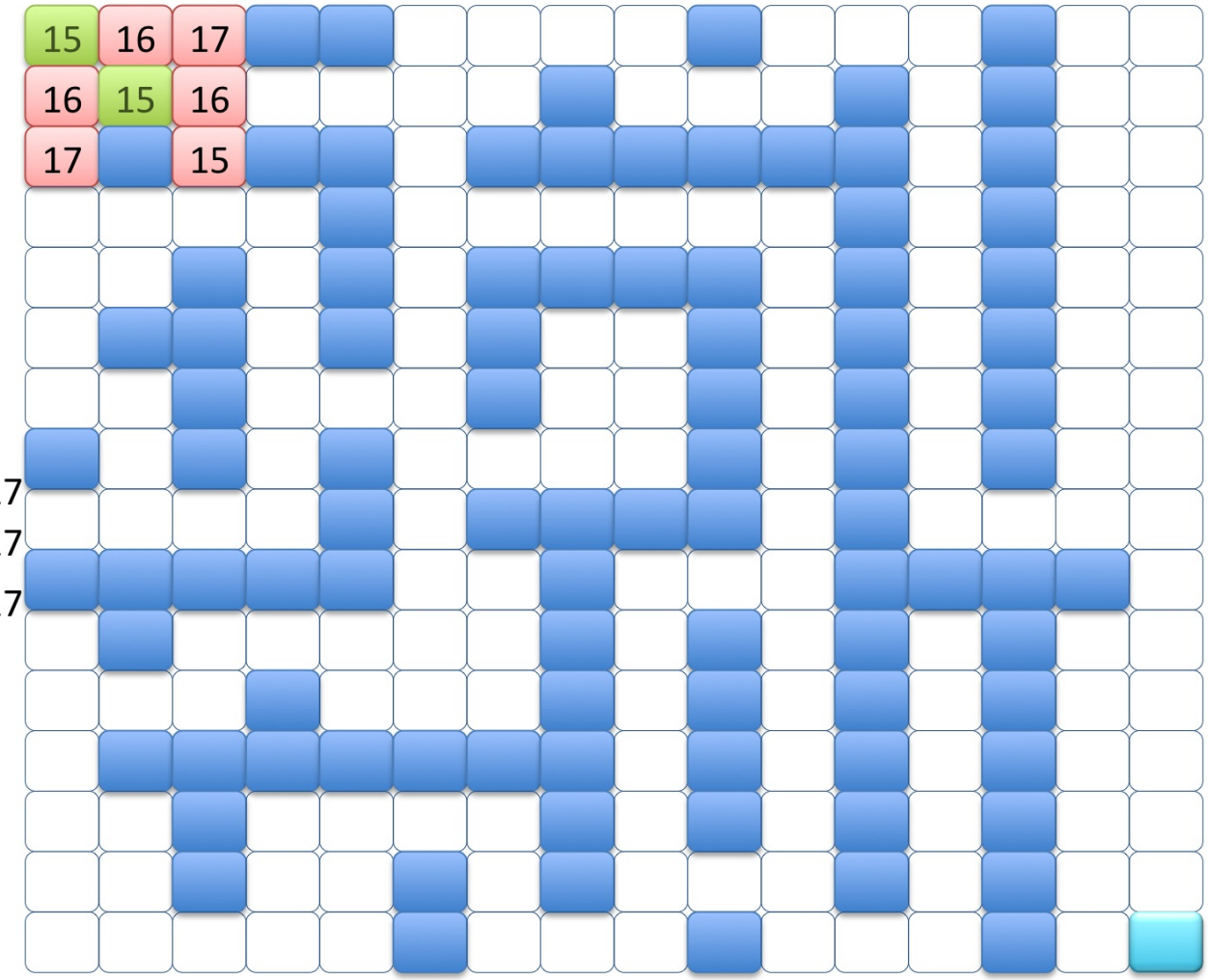


Labyrinth. $h(n = (x_i, y_i), m = (x_m, y_m)) = \max(|x_i - x_m|, |y_i - y_m|)$

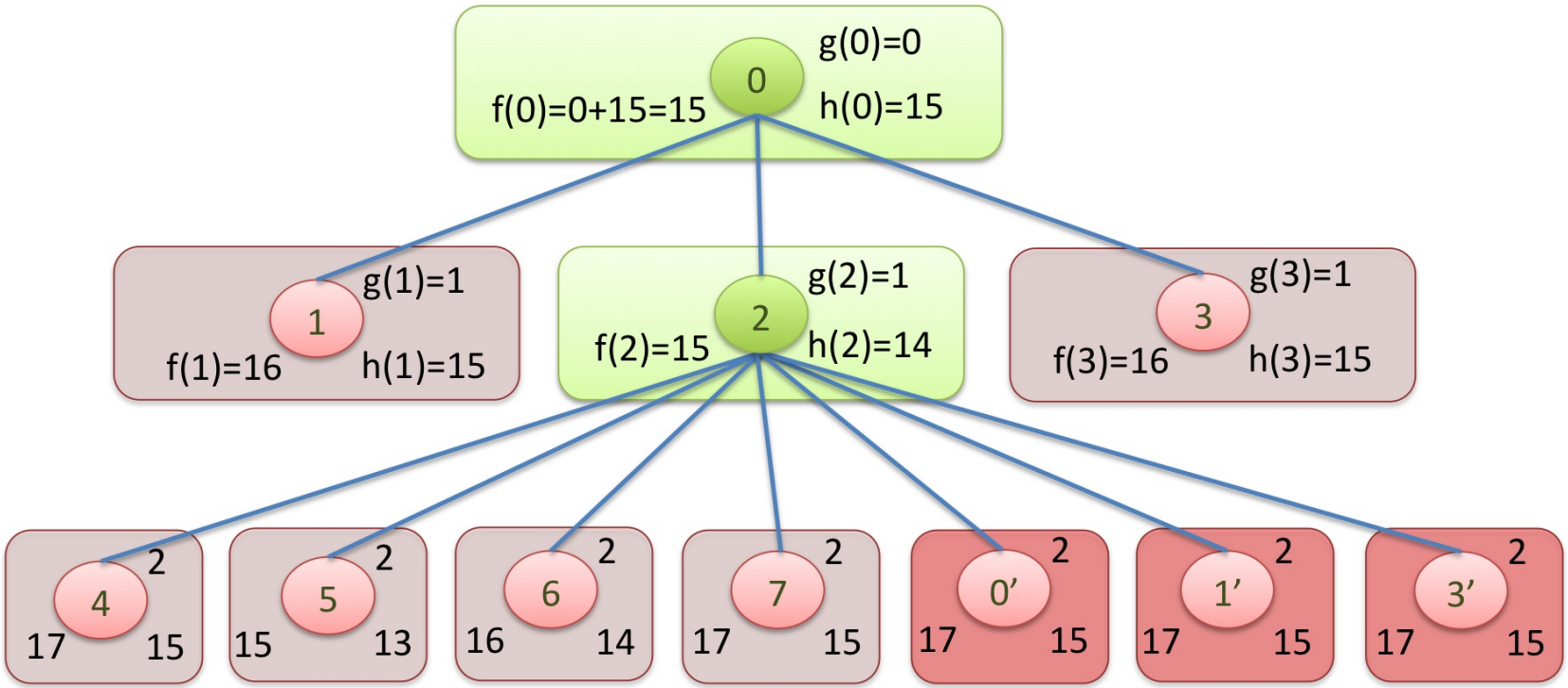


Labyrinth. $h(n = (x_i, y_i), m = (x_m, y_m)) = \max(|x_i - x_m|, |y_i - y_m|)$

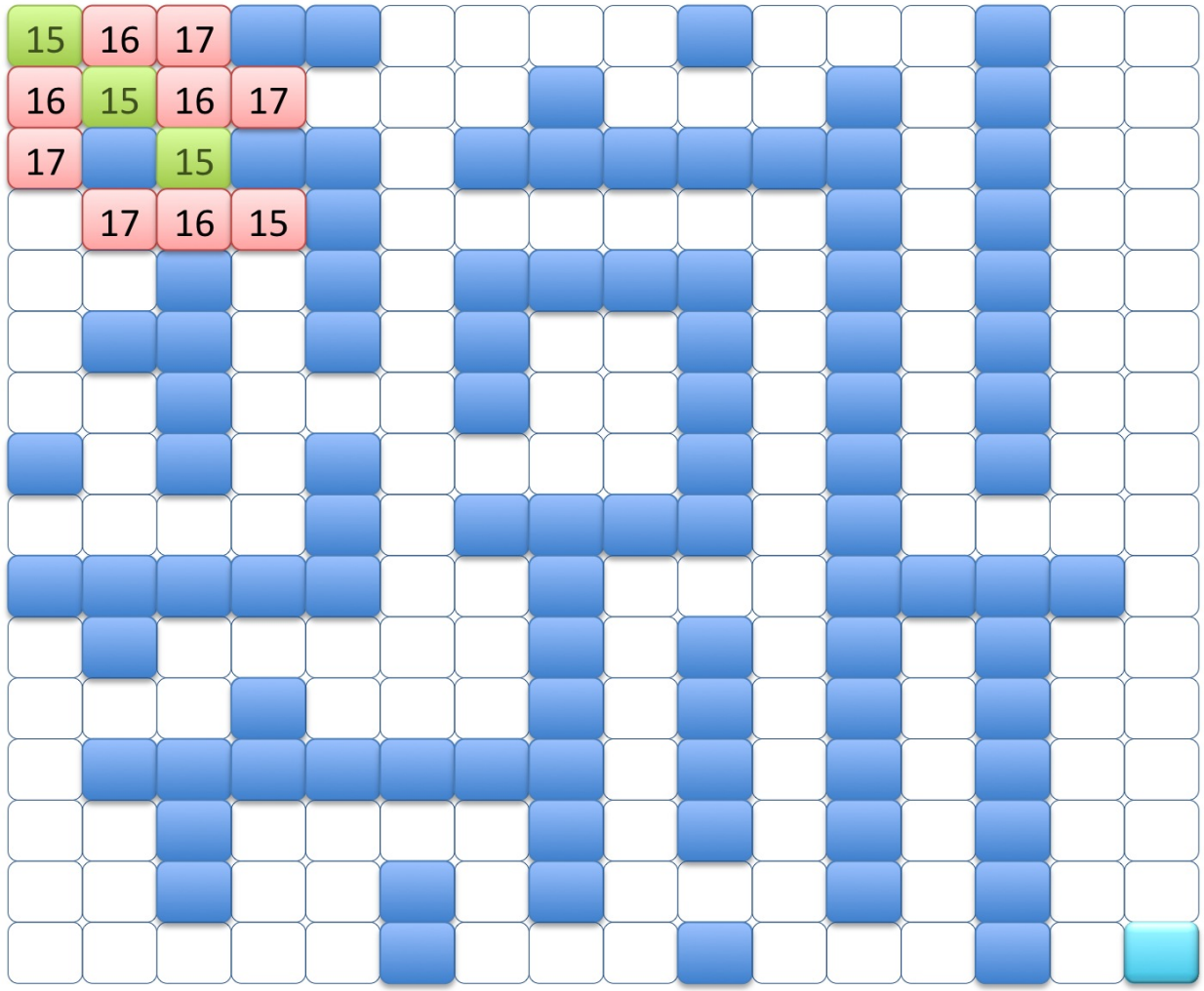
f(0)=g(0)+h(0)=0+15=15
f(1)=g(1)+h(1)=1+15=16
f(2)=g(2)+h(2)=1+14=15
f(3)=g(3)+h(3)=1+15=16
f(4)=g(4)+h(4)=2+15=17
f(5)=g(5)+h(5)=2+13=15
f(6)=g(6)+h(6)=2+14=16
f(7)=g(7)+h(7)=2+15=17
f(0')=g(0')+h(0')=2+15=17
f(1')=g(1')+h(1')=2+15=17
f(3')=g(3')+h(3')=2+15=17



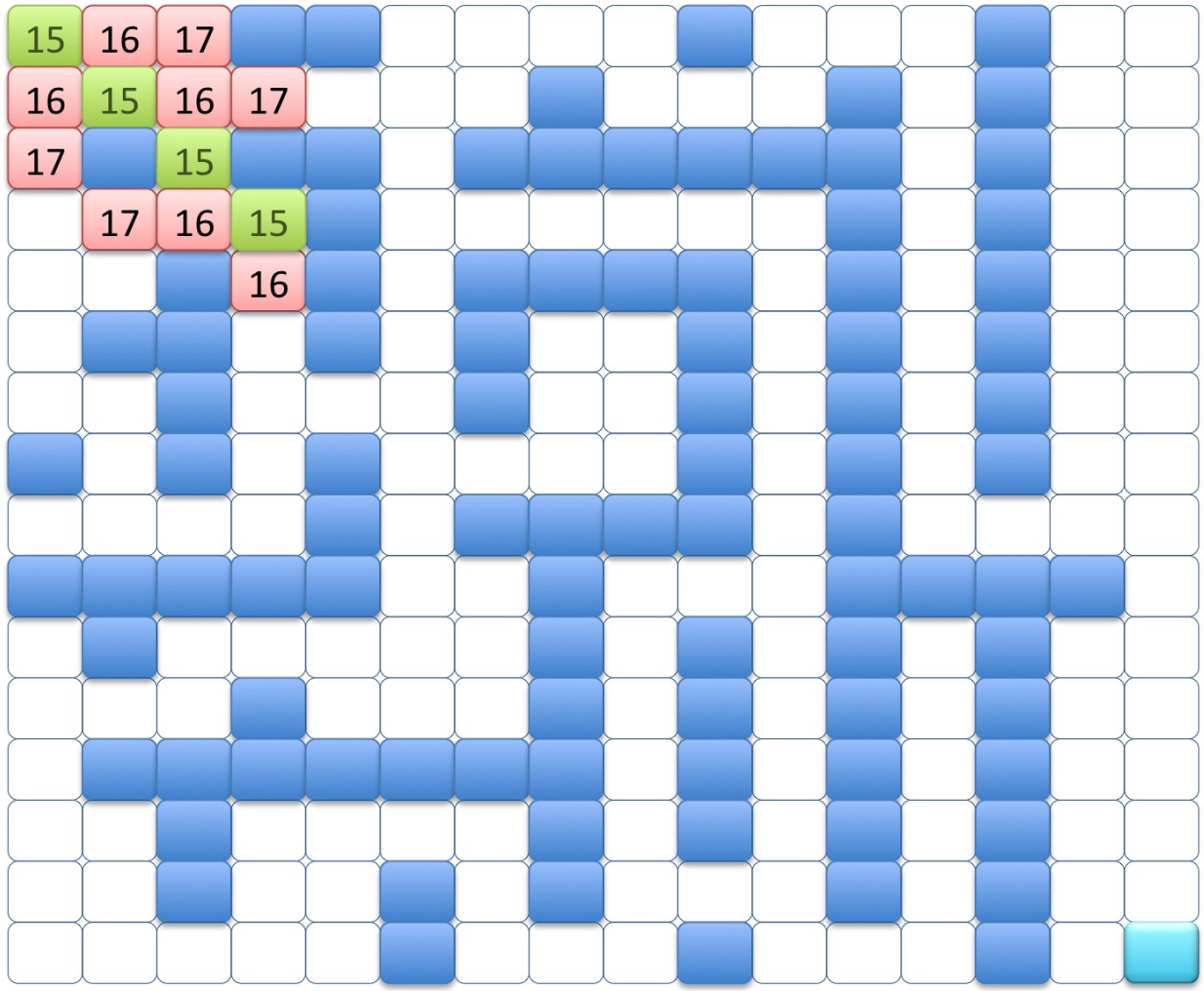
Labyrinth. $h(n = (x_i, y_i), m = (x_m, y_m)) = \max(|x_i - x_m|, |y_i - y_m|)$



Labyrinth. $h(n = (x_i, y_i), m = (x_m, y_m)) = \max(|x_i - x_m|, |y_i - y_m|)$

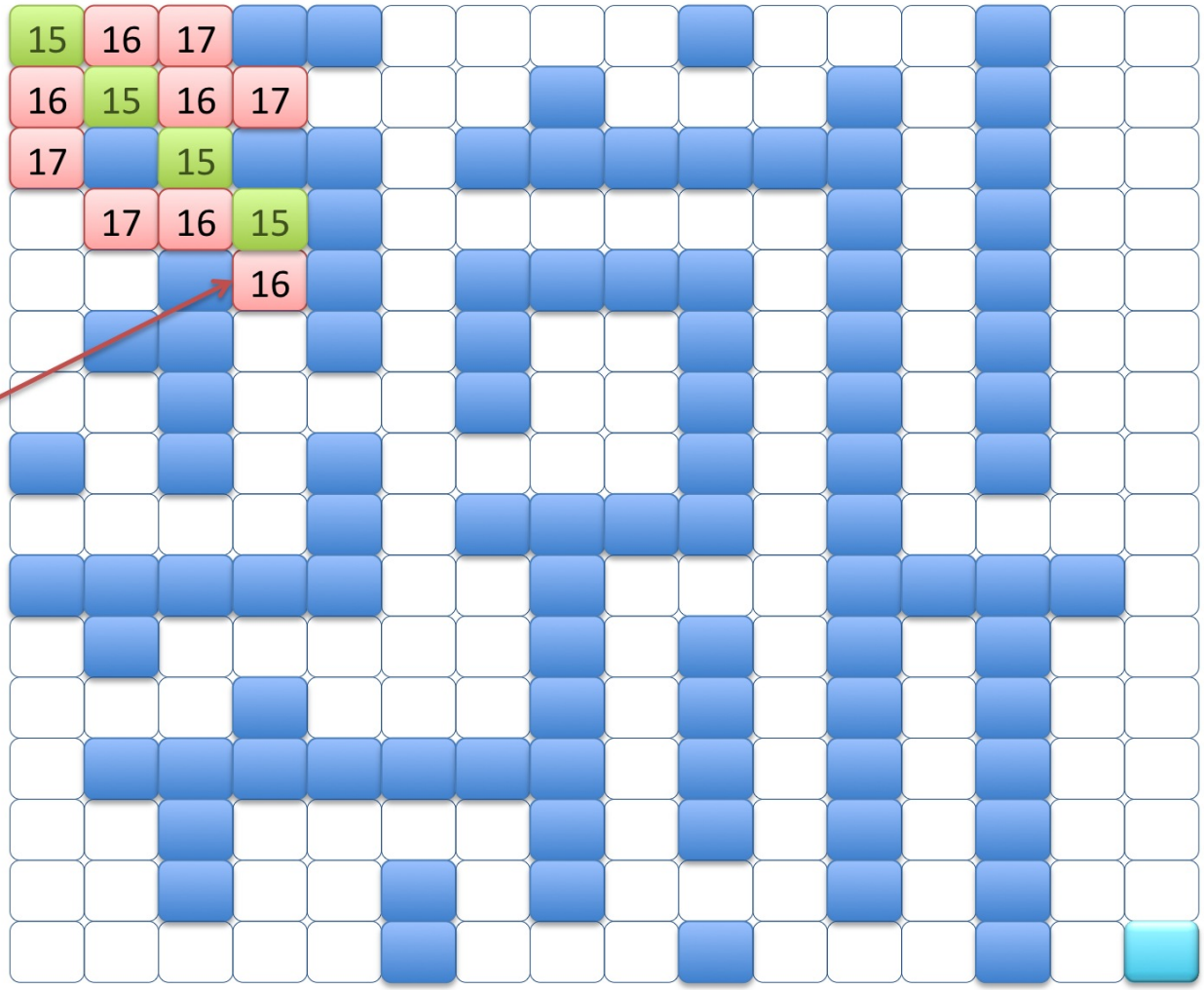


Labyrinth. $h(n = (x_i, y_i), m = (x_m, y_m)) = \max(|x_i - x_m|, |y_i - y_m|)$

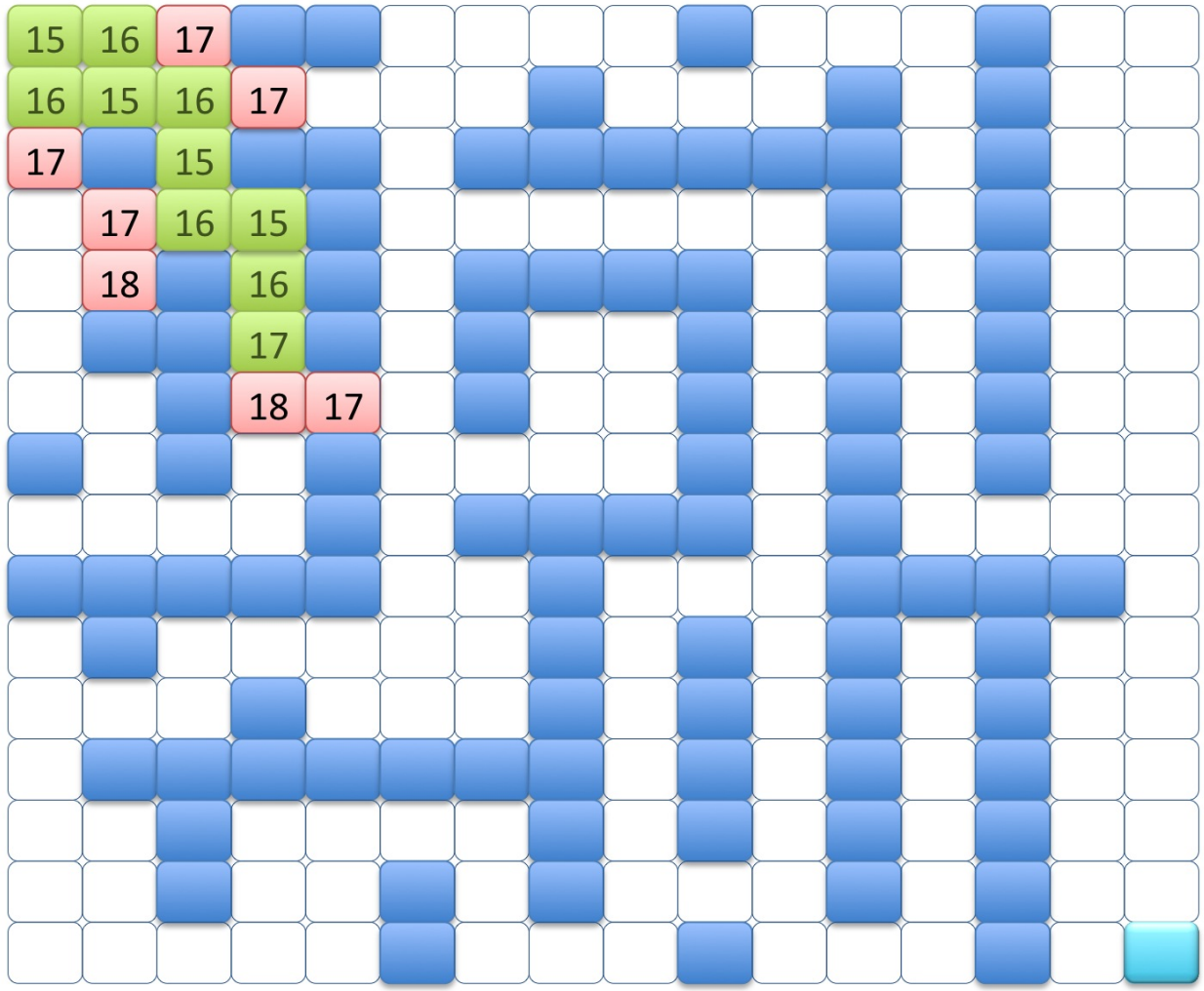


Labyrinth. $h(n = (x_i, y_i), m = (x_m, y_m)) = \max(|x_i - x_m|, |y_i - y_m|)$

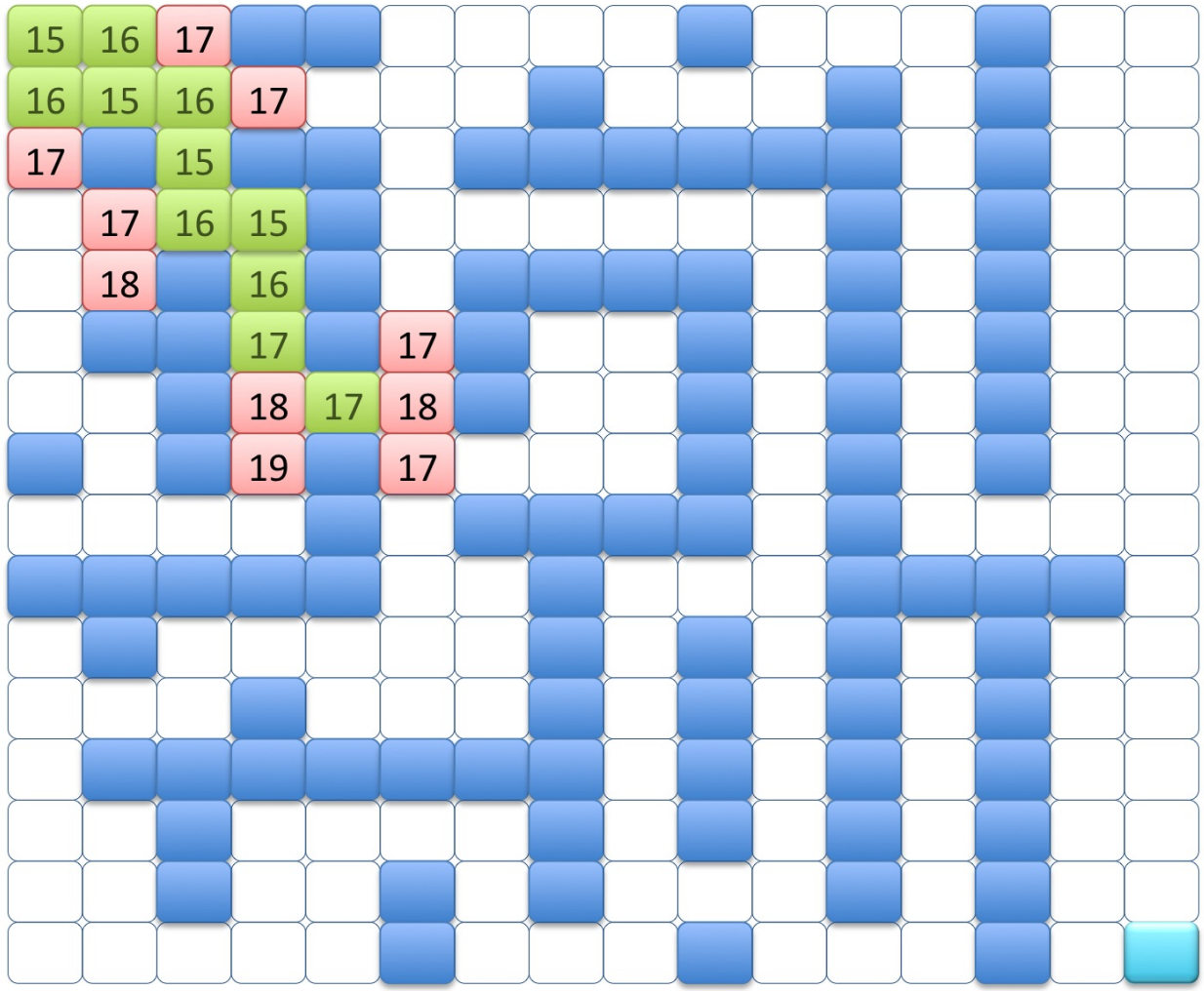
Ties are broken selecting node with smaller h(n)



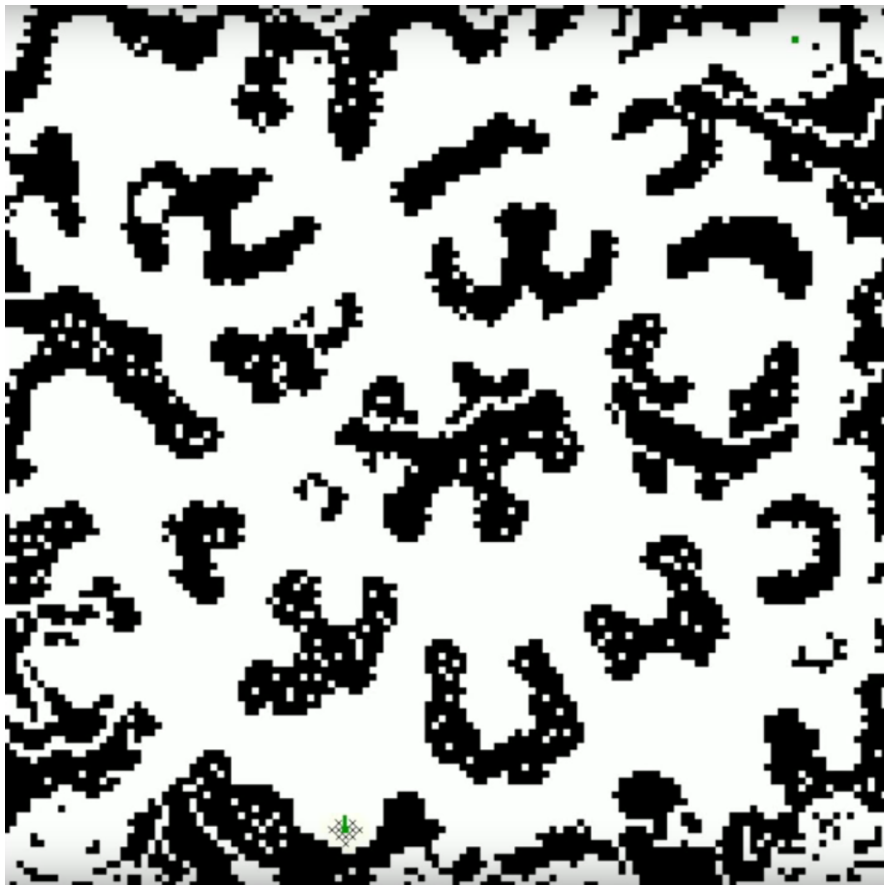
Labyrinth. $h(n = (x_i, y_i), m = (x_m, y_m)) = \max(|x_i - x_m|, |y_i - y_m|)$



Labyrinth. $h(n = (x_i, y_i), m = (x_m, y_m)) = \max(|x_i - x_m|, |y_i - y_m|)$



Example in a game map



https://www.youtube.com/watch?v=huJEgJ82360

Further information

- Admissible heuristic: $h(n) \leq h^*(n)\ \forall n$
- Consistent heuristic: $h(n) \leq k(n, n') + h(n')\ \forall n, n'$
  (n' successor of n)
- Goal-aware heuristic: $h(n) = 0\ \forall n \in G$
  (G is the set of goal nodes)
- If $h(n)$ is consistent, it will never need to restudy a node
- Property: every consistent and goal-aware heuristic is admissible

Further information

- Admissible heuristic: $h(n) \leq h^*(n)\ \forall n$
- Consistent heuristic: $h(n) \leq k(n, n') + h(n')\ \forall n, n'$
  (n' successor of n)
- Goal-aware heuristic: $h(n) = 0\ \forall n \in G$
  (G is the set of goal nodes)
- If $h(n)$ is consistent, it will never need to restudy a node
- Property: every consistent and goal-aware heuristic is admissible
- Completeness: yes
- Admissibility: yes, if
  - number of successors for every node is finite,
  - $k(n_i, n_j) \geq \epsilon > 0$ for each edge, and
  - heuristic function $h(\cdot)$ is admissible

Further information

- Admissible heuristic: $h(n) \leq h^*(n)\ \forall n$
- Consistent heuristic: $h(n) \leq k(n, n') + h(n')\ \forall n, n'$
  (n' successor of n)
- Goal-aware heuristic: $h(n) = 0\ \forall n \in G$
  (G is the set of goal nodes)
- If $h(n)$ is consistent, it will never need to restudy a node
- Property: every consistent and goal-aware heuristic is admissible
- Completeness: yes
- Admissibility: yes, if
  - number of successors for every node is finite,
  - $k(n_i, n_j) \geq \epsilon > 0$ for each edge, and
  - heuristic function $h(\cdot)$ is admissible
- If $\forall n \quad h_1(n) <= h_2(n)$,
  - $h_2(n)$ is more informative than $h_1(n)$
  - $h_2(n)$ expands fewer (or equal) nodes
  - Example: Manhattan distance is more informative (less relaxed) than the number of misplaced cells

Characteristics

- Extremes
  - $h(n) = 0$ for each node: there is no information (Dijkstra)
  - $h(n) = h^*(n)$ for each node: there is perfect information
- Efficiency
  - it makes no sense to devote more computational resources to compute a good $h(n)$ for an equivalent search
  - balance between
    - time to compute
    - saved number of nodes

Summary of some best-first techniques

- Uninformed
  - Breadth-first search: $f(n) = \text{depth}(n)$
  - Dijkstra: $f(n) = g(n)$
- Informed (heuristics)
  - A* [Hart *et al.*, 1968]: $f(n) = g(n) + h(n)$
  - Greedy best-first search: $f(n) = h(n)$
  - IDA* [Korf, 1985]: $f(n) = g(n) + h(n)$ (iterative, bounded)
  - Weighted A* [Pohl, 1970]: $f(n) = g(n) + \omega h(n), \omega > 1$
    - complete, but not admissible
    - optimal solution cost: $f(g) \leq \omega h^*(i)$
      $h^*(i)$: optimal solution
      $f(g)$: cost of found solution

Summary

- Relaxation as a source of heuristic functions
- Heuristic functions: computation time vs. informedness
- Strategy: greedy (HC) or systematic (A*)
- Objective: satisficing or optimisation
- Optimisation metrics: (minimum/maximum) solution cost, length, etc.

References


Peter Hart, Nils Nilsson, and Bertram Raphael.
Formal basis for the heuristic determination of minimum cost paths.
IEEE Transactions System Science and Cybernetics, 4(2):100–107, 1968.


Richard E. Korf.
Depth-first iterative deepening: An admissible tree search.
Artificial Intelligence, 27(1):97–109, 1985.


Ira Pohl.
Heuristic search viewed as path finding in a graph.
Artificial Intelligence, 1(3-4):193–204, 1970.