02

Chapiter

Problem Solving by Searching

2I1AE1: Artificial Intelligence

Régis Clouard, ENSICAEN - GREYC

"The beginning of all sciences is the astonishment that things are what they are."
Aristote

In this chapter

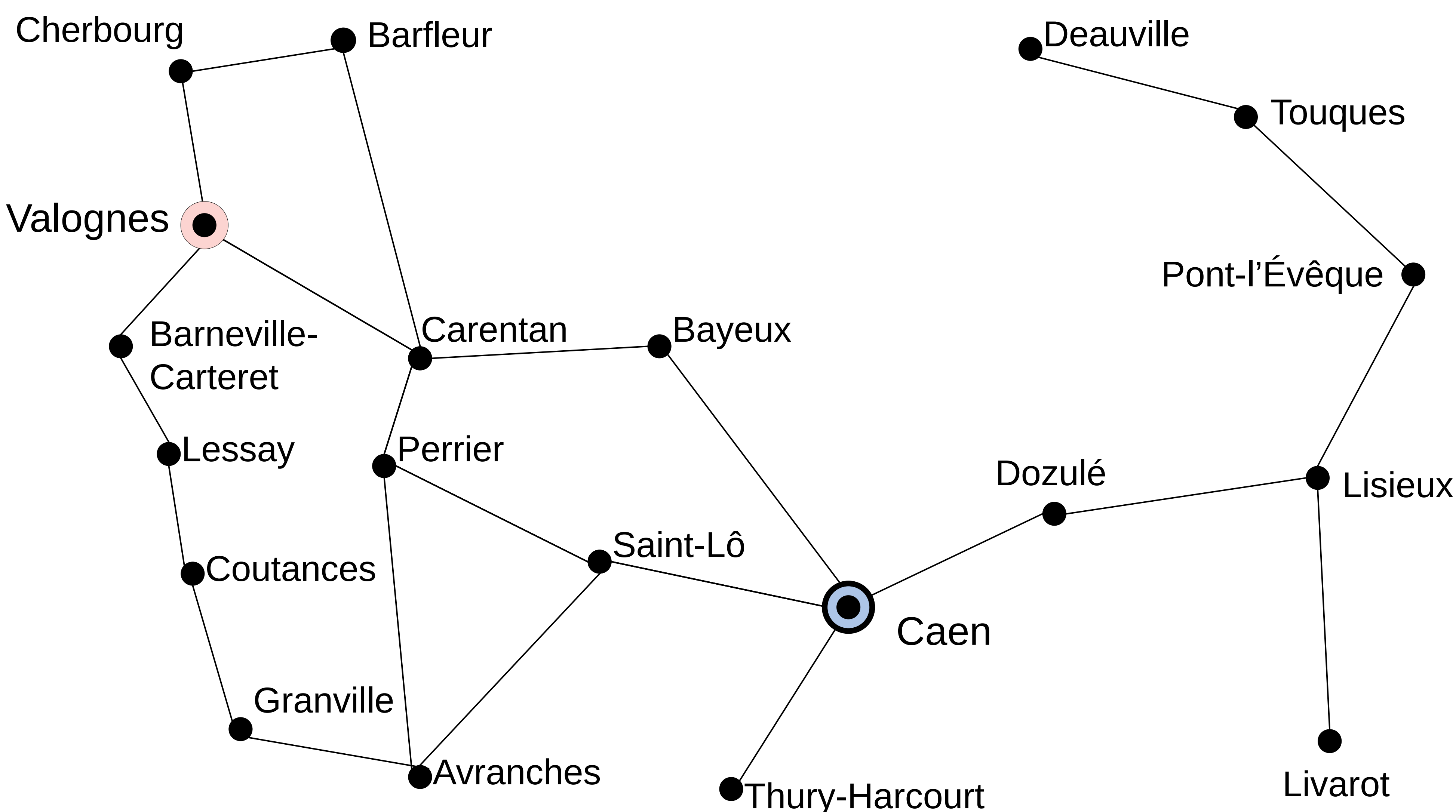
- Agents that plan ahead
  - State Space Search problems
  - Representing State Space with Graph
  - Searching a solution

1- State Space Search Problems

- What we need:
  - A set of states: S
  - A start state: $s_0 \in S$
  - A goal state or set of goal states or, equivalently, a goal test : a boolean function which tells us whether a given state $s_g \in S$ is a goal state.
  - A set of actions : $a \in A$
  - An action function : a mapping from a state and an action to a new state $f(s,a) \rightarrow s'$
- A **state** is a snapshot of the environment described with quantitative or qualitative data

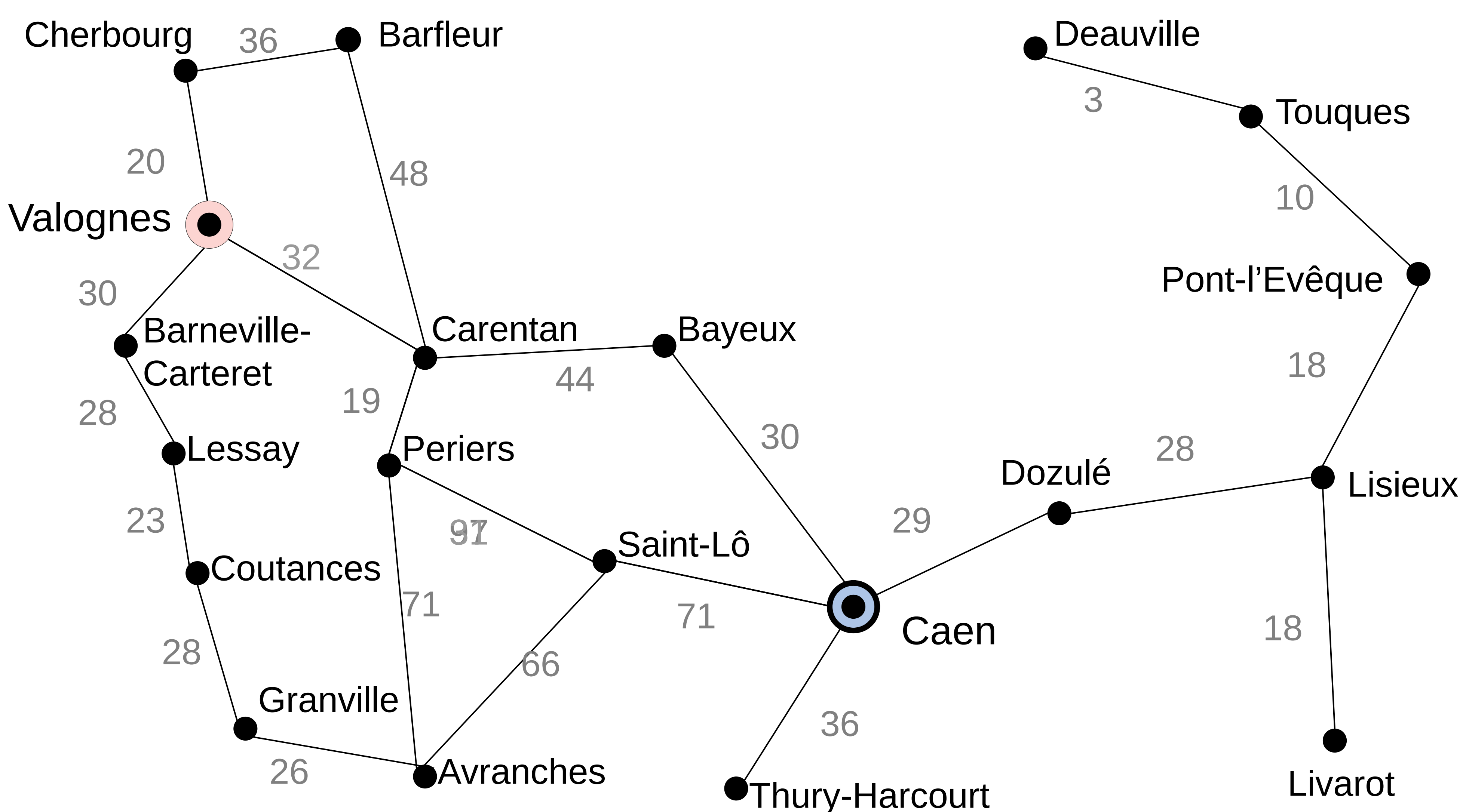
Example of State Space Search Problem: Traveler Problem

- Traveler problem : Find a route from one city (*Valognes*) to the other (*Caen*) given the road map



Example of State Space Search Problem: Traveler Problem

- Another flavor: find the route with the **minimum length** between the two cities



Example of State Space Search Problem: Puzzle 8

- Find the sequence of the ‘empty tile’ moves from the initial game position to the designated target position

Initial position

5	4	
6	1	8
7	3	2

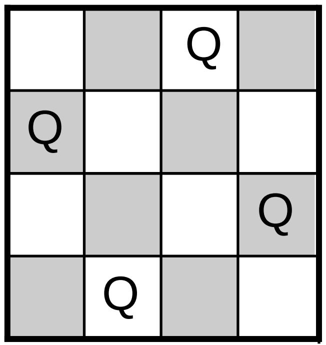
Goal position

1	2	3
4	5	6
7	8	

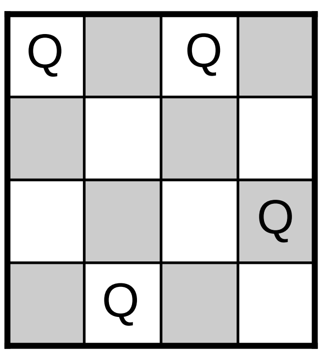
[http://www.tilepuzzles.com/default.asp?p=12]

Example of State Space Search Problem: N-Queens Problem (Carl Gauss- 1850)

- Find a configuration of N queens not attacking each other on an N$\times$N chessboard



A goal configuration



A bad configuration

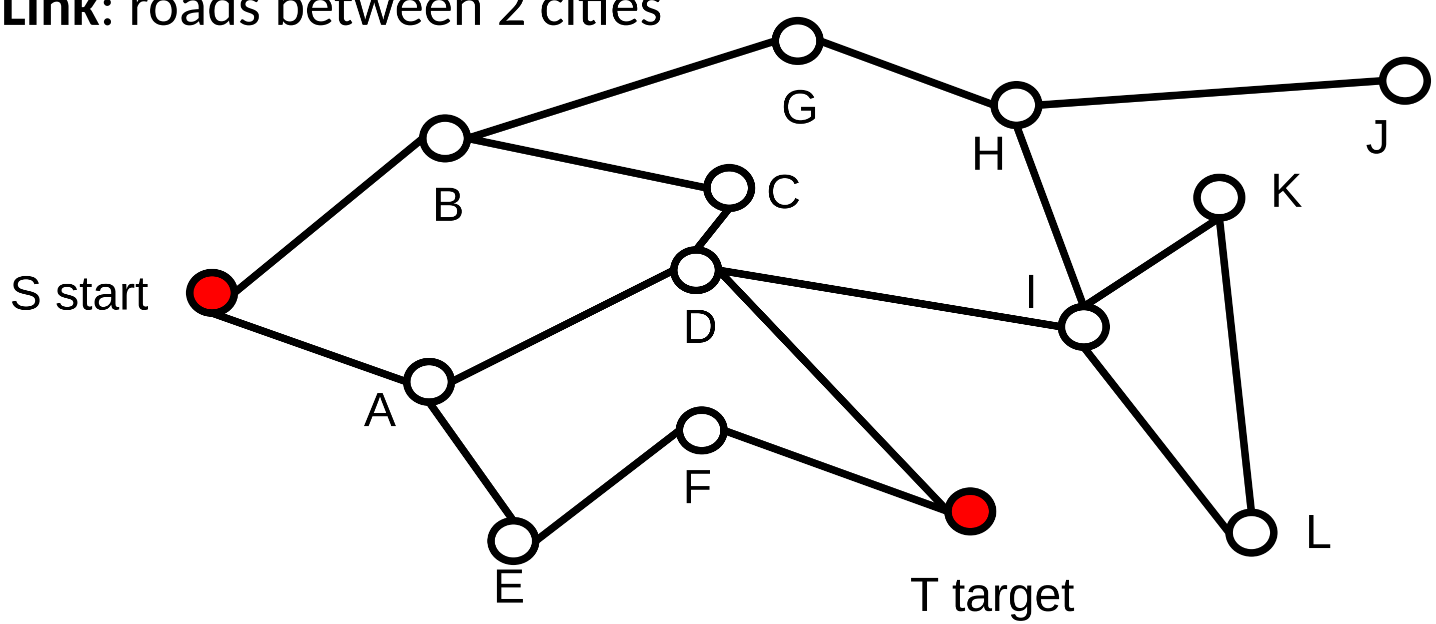
[http://eightqueen.becher-sundstroem.de/]

Search Problem

- A search problem is defined by:
  - A search space: The set of objects among which we search for the solution.
    - *Example of state: City, N-queen configuration*
  - An initial state
    - *Example: Valognes*
  - A goal test: What are the characteristics of the object we want to find in the search space?
    - Examples:
      - *Path between cities S and T.*
      - *Path between A and B with the smallest number of links.*
      - *Path between A and B with the shortest distance.*
      - *Path between A and B with the cheapest price (tolls…).*
  - Actions: How to go from one state to another.
    - *Example: use a road between two cities.*

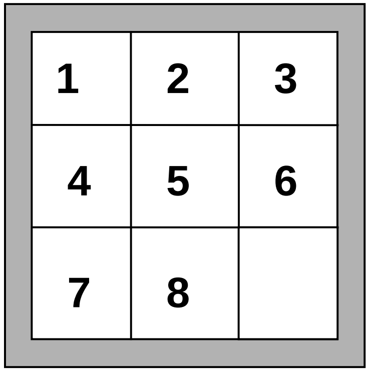
2- State space graph: A first class of Search Problem representation

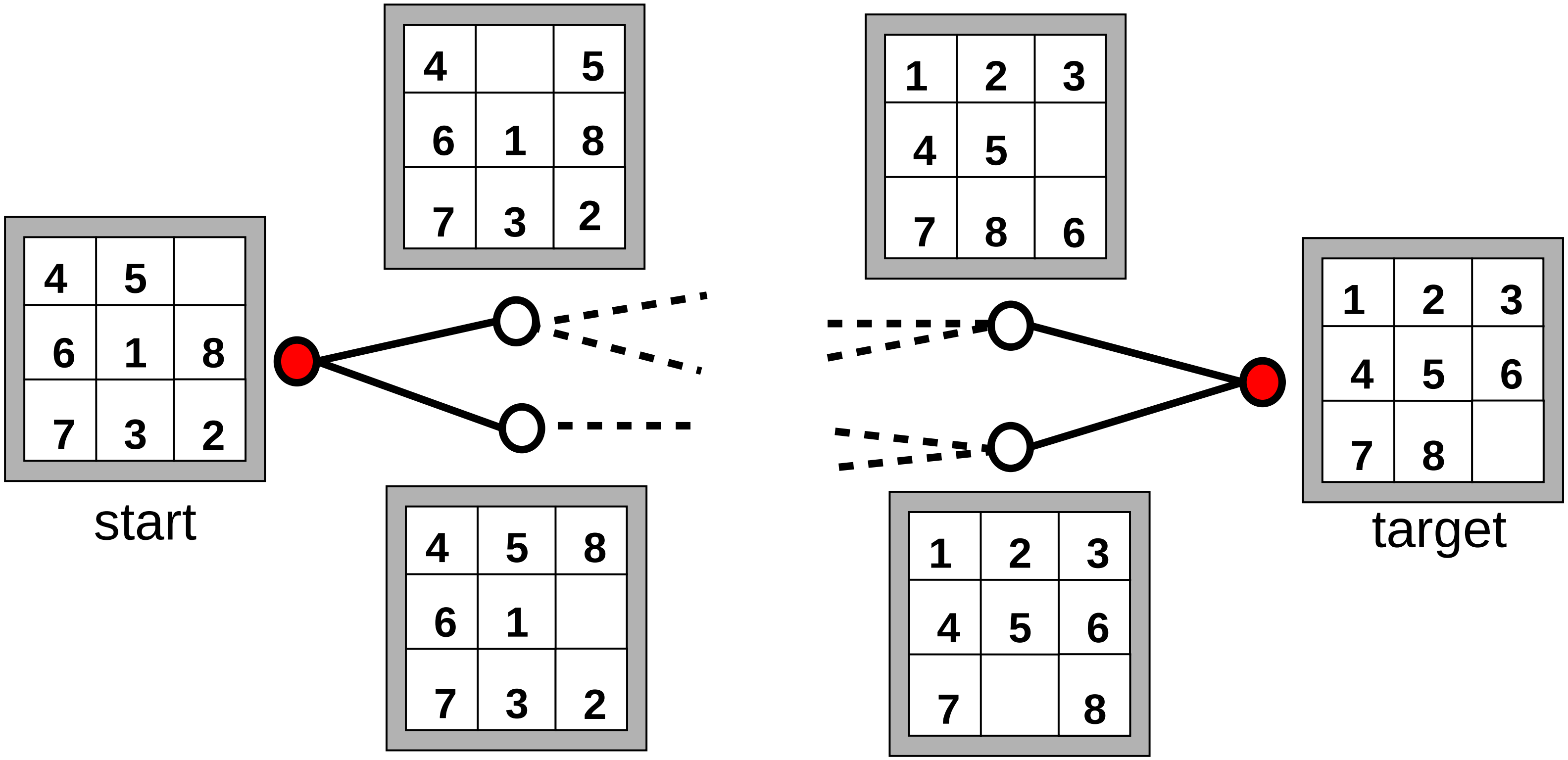
- A large class of search problems can be naturally represented as **graph search problems**.
  - **Node**: state
  - **Link**: action
- Typical example: Path finding
  - **Goal**: path between S and T
  - Graph
    - **Node**: city
    - **Link**: roads between 2 cities



Graph Search Problem

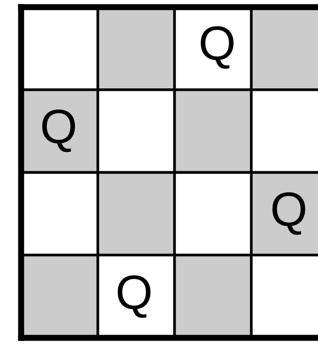
- Another example: Puzzle 8
  - **Goal**: find a sequence of moves from the initial configuration to the goal configuration.
  - Graph:
    - **Nodes (States)**: configuration of the game.
    - **Links**: valid moves made by the player.

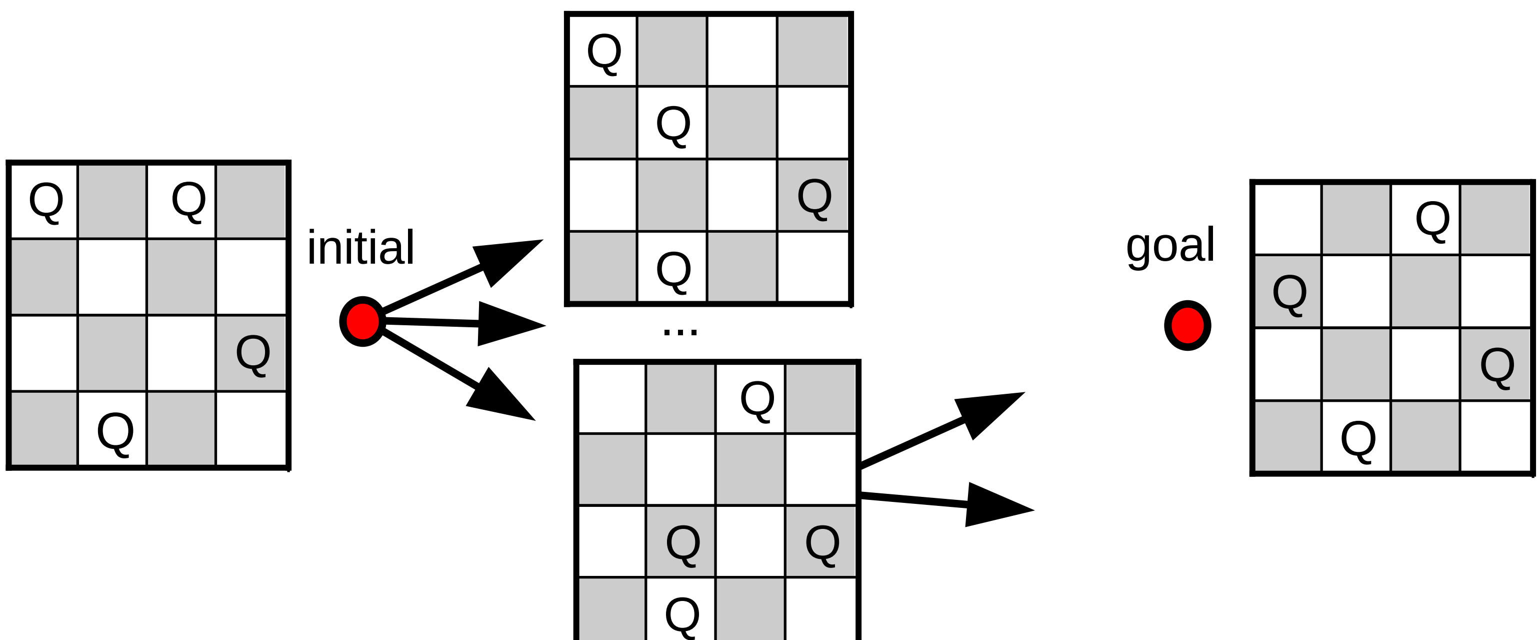




Graph Search Problem

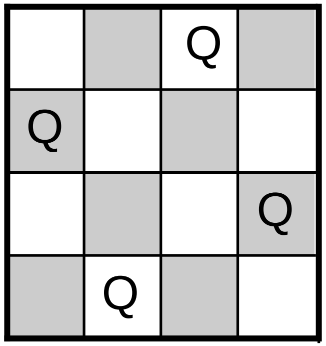
- Less obvious conversion example: N-queens problem
  - Can we convert it to a graph search problem?
  - **Goal**: the N queens in a configuration not attacking each other.
  - Graph:
    - **Nodes (States)**: N-queen configurations.
    - **Initial node**: an arbitrary N-queen configuration.
    - **Links**: change a position of one queen on a n empty square.

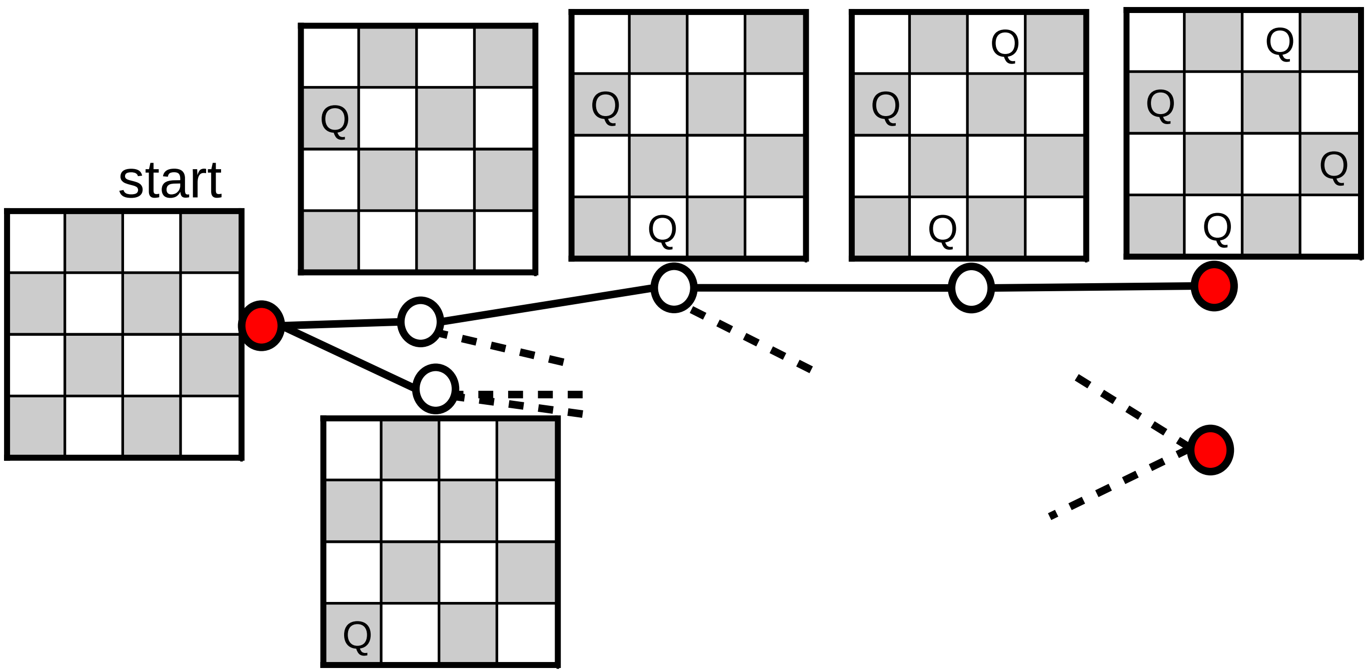




Graph Search Problem

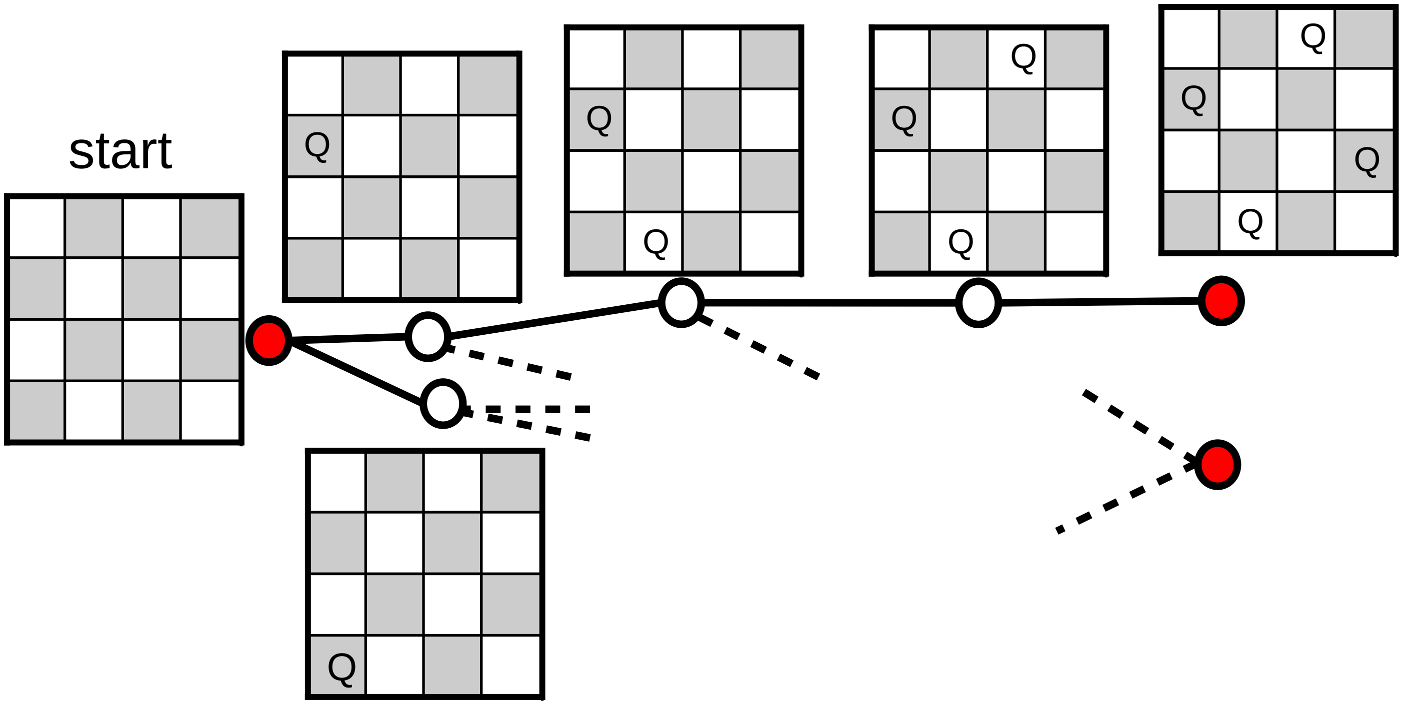
- Is there an alternative way to formulate the N-queens problem as a search problem?
  - **Goal**: the N queens in a configuration not attacking each other.
  - Graph:
    - **Nodes (States)**: configurations of 0, 1, 2,… N queens.
    - **Initial node**: no queens on the board.
    - **Links**: additions of a queen to the board on an empty square.





Graph Search Problem

- N-queens problem is a different graph search problem when compared to Puzzle 8 or Route planning.
  - **We want to find only the target configuration, not a path.**



Two Types of Graph Search Problems

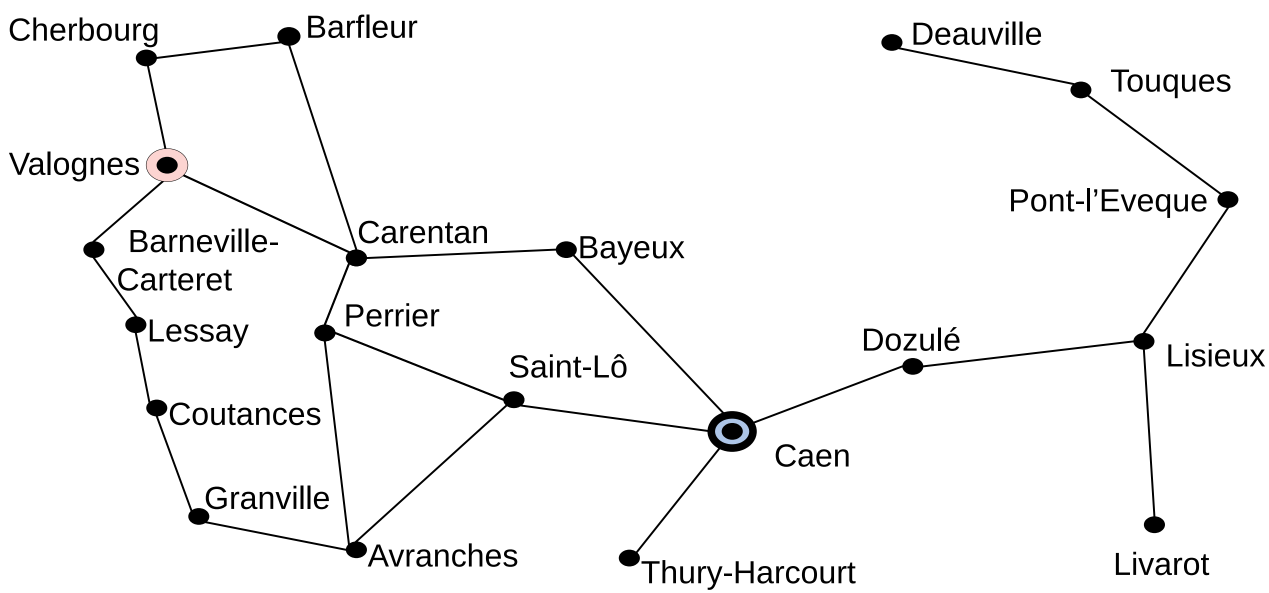
1. **Path search**
   - Find a path between states s_0 and s_t
   - Examples: Traveler problem, Puzzle 8
   - **Additional goal criterion**: minimum length/cost path
   - Solution: sequence of actions

2. **Configuration search**
   - Find a state satisfying the goal condition
   - Examples: N-queens problem
   - **Additional goal criterion**: none
   - Solution: a configuration

Well-Defined Search Problem

- Search problems should be defined formally in terms of:
  - **States** (*Graph nodes*)
    - The set of objects we search for the solution.
  - **Initial state** *(Graph root node)*
    - State we start to search from.
  - **Actions** (*Graph* links)
    - Transform one state to another.
  - **Goal test**
    - A test that decides whether the target state is reached.
  - **Solution cost function**
    - Assigns a numeric cost to each solution.

Example: Traveler Problem



- Search problem formulation:
  - **States**: cities
  - **Initial state**: city of Valognes
  - **Actions**: moves to neighbor cities using road
  - **Goal test**: city of Caen
  - **Type of the problem**: path search
  - **Possible solution cost**: path length

Example: Puzzle-8 Problem

5	4	
6	1	8
7	3	2

Initial position

1	2	3
4	5	6
7	8	

Goal position

- Search problem formulation:
  - **States**: tile configurations
  - **Initial state**: initial configuration
  - **Actions**: moves of the empty tile
  - **Goal test**: reach the winning configuration
  - **Type of the problem**: path search
  - **Possible solution cost**: a number of moves

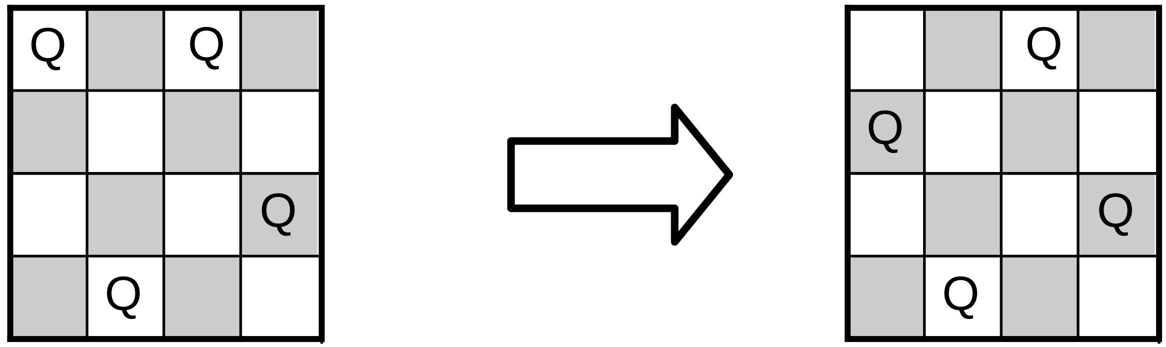
Example: Knuth Conjecture

- Donald Knuth's conjecture (1964)
  - One can start at 3 and reach any integer by iterating factorial, sqrt, and floor.
    - Example: from 3 reach 5 (one solution $\lfloor\sqrt{\sqrt{(3!)!}}\rfloor = 5$)

- Search problem formulation:
  - **States**: set of natural numbers $\mathbb{N}$
  - **Initial state**: number 3
  - **Actions**: Floor, square root, and factorial operations
  - **Goal test**: any given natural number
  - **Type of the problem**: path search
  - **Possible solution cost**: each action has a cost of 1

- Note: the search space is infinite.

Example: N-Queens Problem

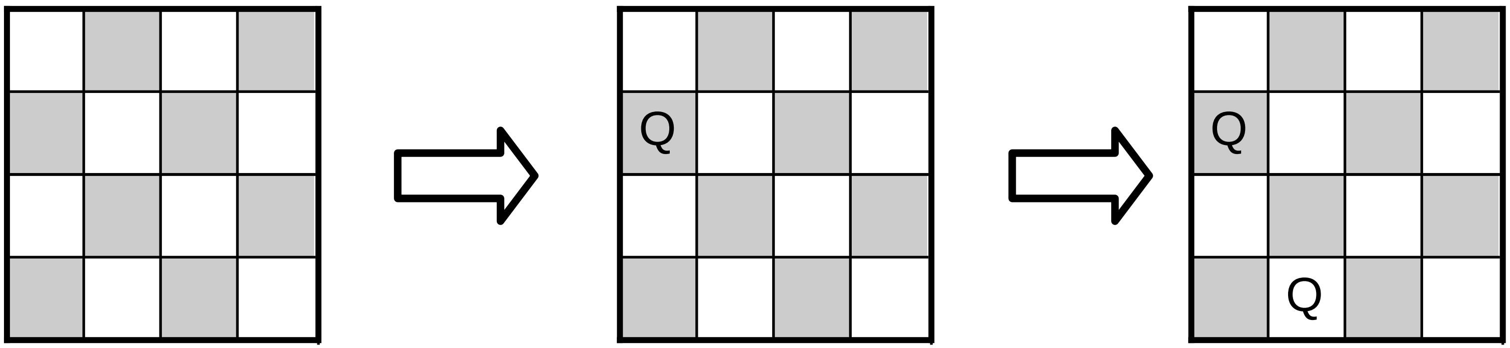


Bad goal configuration **Valid goal configuration**

- Search problem formulation:
  - **States**: configurations of 4 queens on the board
  - **Initial state**: an arbitrary configuration of 4 queens
  - **Actions**: move one queen to a different unoccupied position
  - **Goal test**: a configuration with non-attacking queens
  - **Type of the problem**: configuration search
  - **Possible solution cost**: no obvious cost

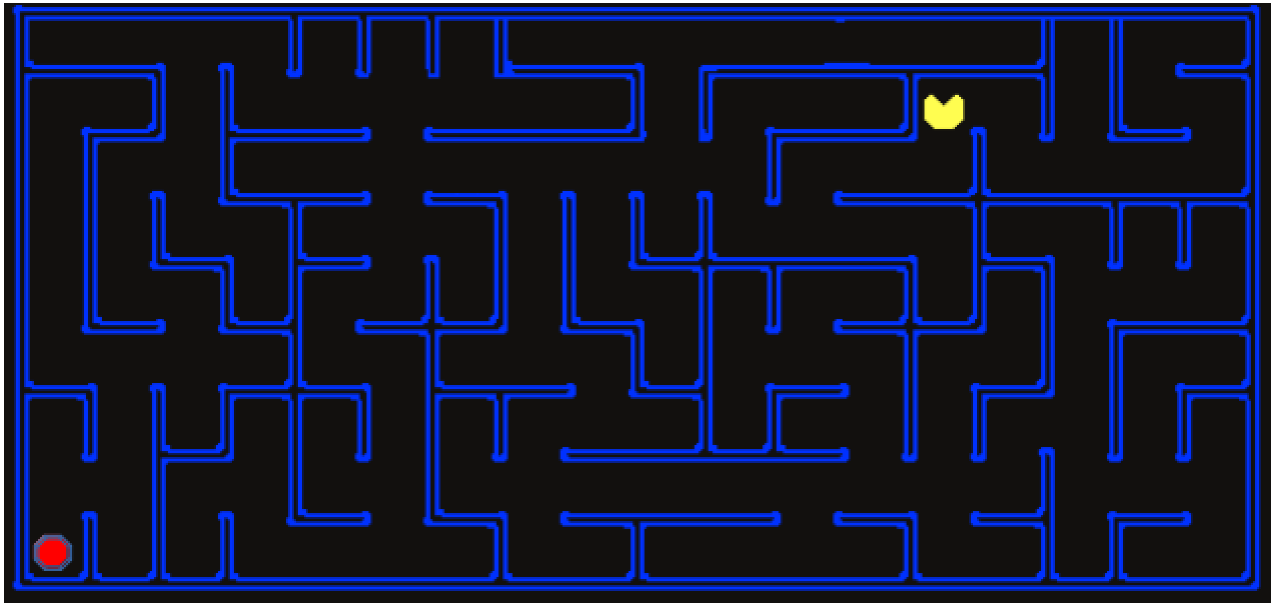
Example: N-Queens Problem

- Alternative formulation of N-queens problem.



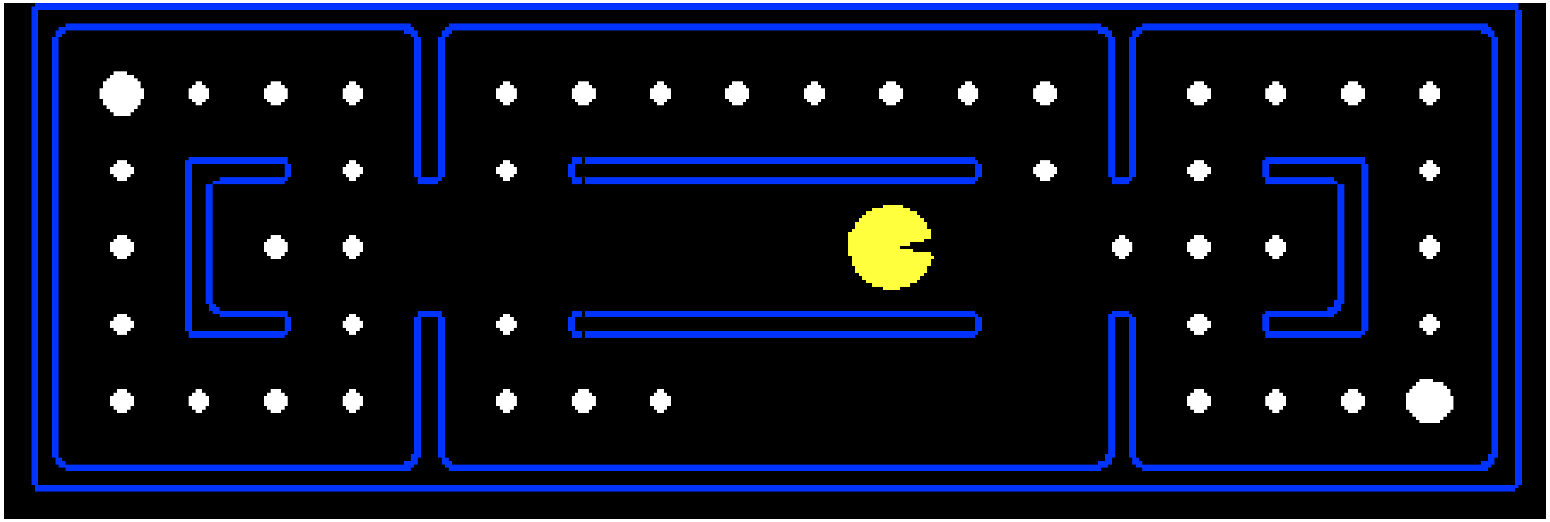
- Search problem formulation:
  - **States**: configurations of 0 to 4 queens on the board
  - **Initial state**: no-queen configuration
  - **Actions**: add a queen to the leftmost unoccupied column
  - **Goal test**: a configuration with 4 non-attacking queens
  - **Type of the problem**: configuration search
  - **Possible solution cost**: no obvious cost

State Representation (Graph Node)



- A search state retains only the necessary details (abstraction).
  - It is a snapshot of the environment.
- Example: pathing
  - **States**: Pacman coordinates (x, y) [size: 2 integers]
  - **Actions**: N, S, E, W.
  - **Goal test :** Pacman reaches end coordinates (x, y).

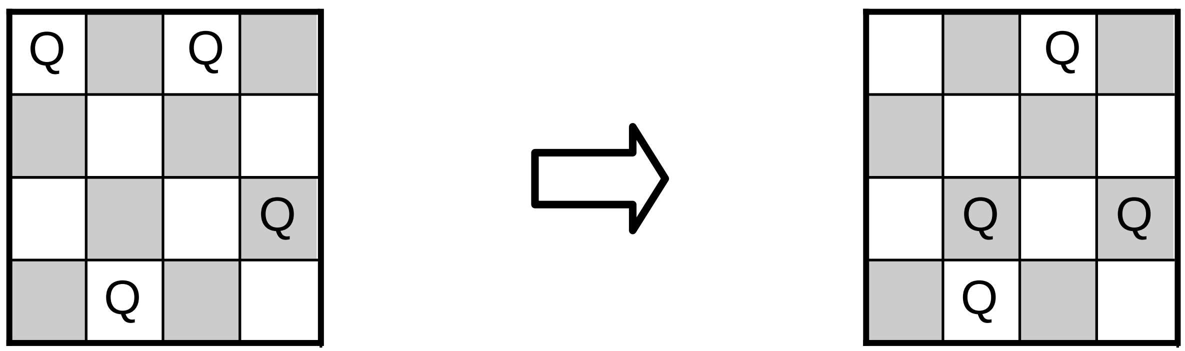
State Representation (Graph Node)



- A search state retains only the necessary details (abstraction).
  - It is a snapshot of the environment.
- Example: eat all dots.
  - **States**: {(xp, yp), dots grid} [size: n*n integers]
  - **Actions**: N, S, E, W.
  - **Goal test :** dots all eaten.

Comparison of Problem Formulation

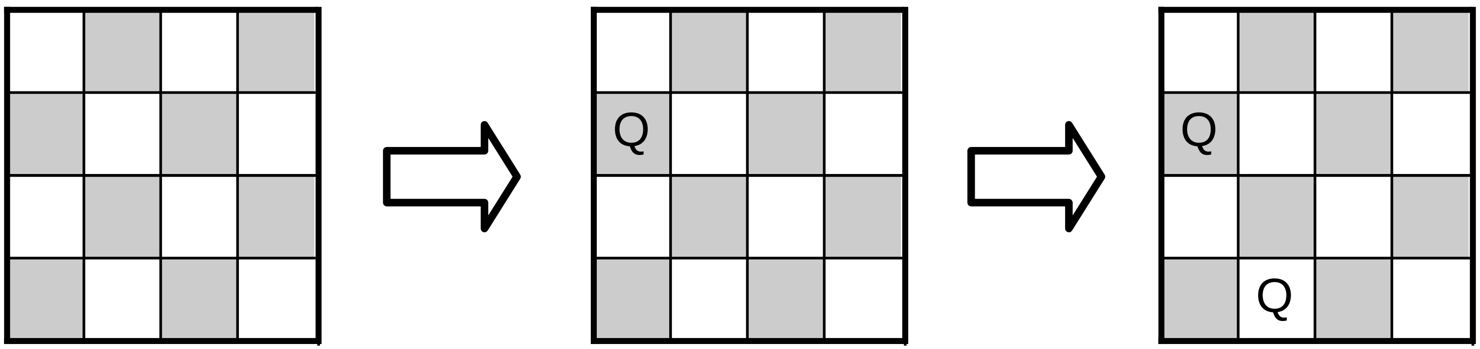
- Consider the N-queens problem.
- **Solution 1**:
  - **States**: board configurations with all queens in the board.
  - **Actions**: switch one of the queens.



  - $\binom{16}{4} = \frac{16!}{4! * 12!} = 1820$ configurations.
  - French checkers (10x10) ≈ 10^{13} (If we consider 10^6 moves/second → 3 years)

Comparison of Problem Formulation

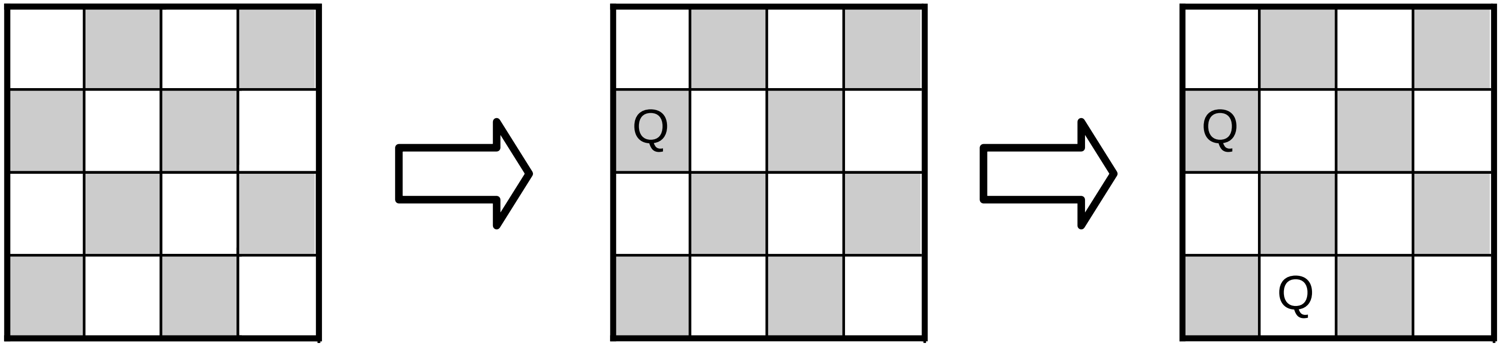
- **Solution 2:**
  - **States**: board configuration of 0 to 4 queens on the board.
  - **Actions**: add a queen to the leftmost unoccupied column.



- $1 + 4 + 4^2 + 4^3 + 4^4 = 341$ configurations altogether.
- French checkers (10x10) ≈ 10^{10} (If we consider 10^6 moves/second → 1 day)

Comparison of Problem Formulation

- **Solution 3:**
  - **States**: board configurations of 0 to 4 queens on the board.
  - **Actions**: add a queen to the leftmost unoccupied column such that it does not attack already placed queens.



  - < 1 + 4 + 4*3 + 4*3*2 + 4*3*2*1 = 65 configurations altogether.
  - French checkers (10x10) ≈ 10^6 (If we consider 10^6 moves/second → 1 s)

3- Searching a solution

- **Search (process)**
  - The process of exploration of the search space.
- **The efficiency of the search depends on:**
  - Size of search space.
  - Method used to explore the search space.

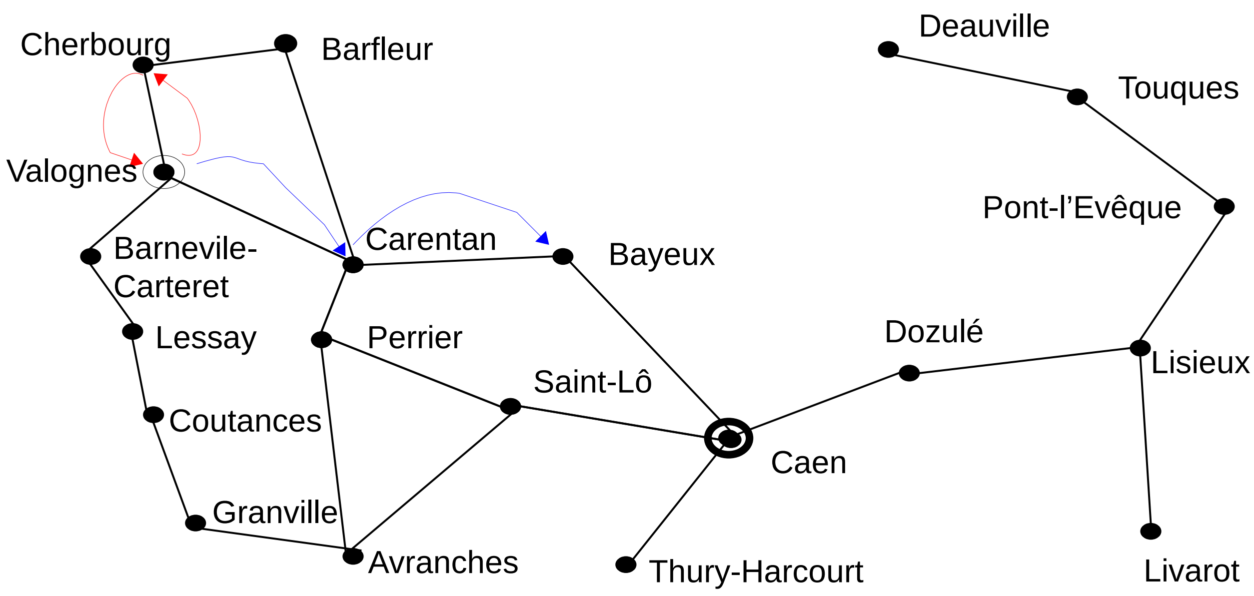
- **Think twice before solving the problem by search:**
  - Choose the **search space** and the **exploration method**.

Search Process

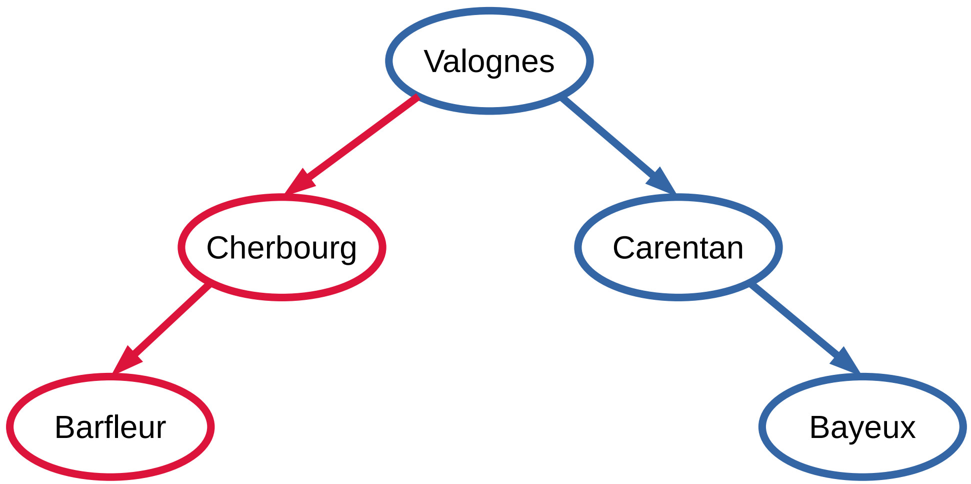
- **Exploration of the state space** through successive application of actions from the initial state.
- **A search tree** = a kind of (search) exploration trace:
  - **Branches** corresponding to explored paths in the graph.
  - **Leaf** nodes corresponding to the exploration **fringe**, built on-line during the search process.

Graph State and Search Tree

- Graph



- Path in the graph: a branch in the search tree

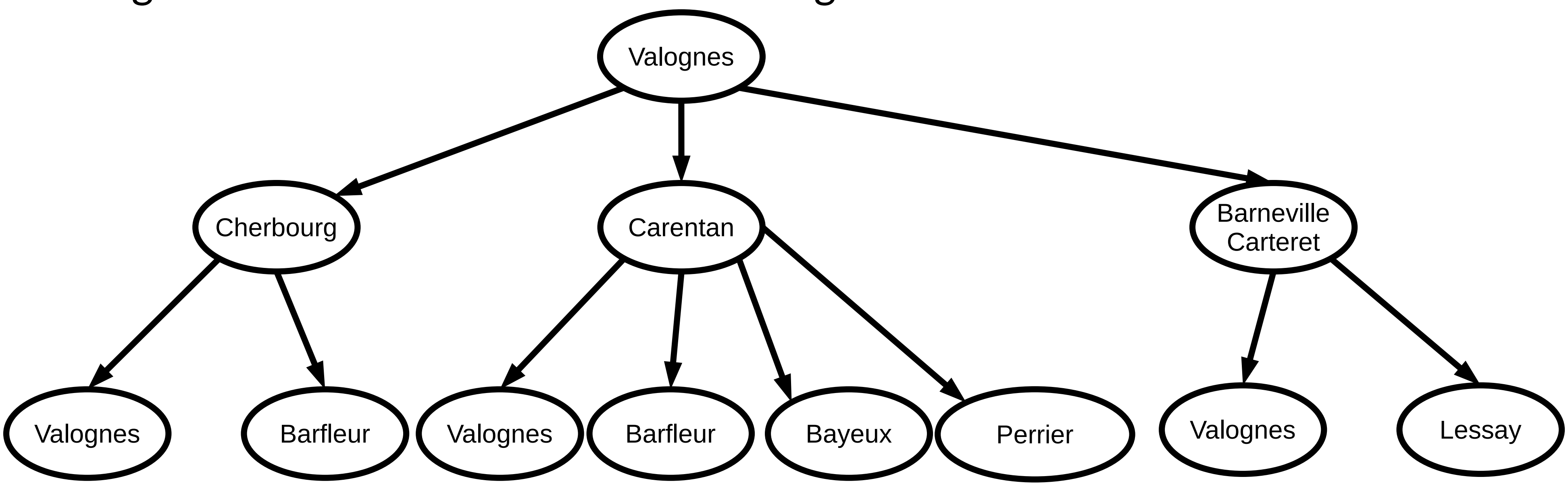


General Search Algorithm

- Based on a **list** nodes to be examined : **open list** (the fringe).

```
function GENERAL-SEARCH(problem) returns solution
  var open-list ← MAKE-LIST(MAKE-NODE(INITIAL-STATE[problem]))
  LOOP
     IF EMPTY(open-list) THEN return failure
     node ← REMOVE-FRONT-LIST(open-list)
     IF IS-GOAL(problem, STATE[node]) THEN return the related solution
     open-list ← ADD-IN-LIST(GET-SUCCESSORS(node, problem), open-list)
end
```

Note: The goal test is NOT done when node is generated but when node is examined.



Search Algorithms

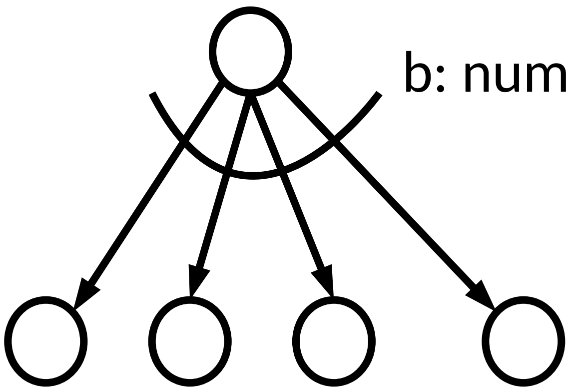
- Various search **algorithms** has been defined from this general algorithm.
  - They only differ on the way they maintain the **open-list** of candidates nodes for expansion:
    - **ADD-IN-LIST(node, open-list)**
- Difference between search path and configuration
  - **Path**: The **open-list** stores the list of successive states from start state to current state
  - **Configuration**: The **open-list** stores only the current state
- Implementation for a **specific problem** needs to provide:
  - **GET-SUCCESSORS(node, problem)**
    - Generate all successor nodes of a given node.
  - **IS-GOAL(problem, state)**
    - Test if state satisfies all goal conditions.

Measuring Problem-Solving Performance

- Search algorithm performance is evaluated in 4 ways:
  - **Completeness**
    - Does the method find the solution if it exists?
  - **Optimality**
    - Is the solution returned by the algorithm optimal?
  - **Time complexity**
    - How much time it takes to find the solution?
  - **Space complexity**
    - How much memory is needed to do this?

Parameters to Measure Complexity

- Space and time complexities
  - Complexity is measured in terms of parameters:
    - ***b*** – maximum branching factor



    - ***d*** – depth of the optimal solution
    - ***m*** – maximum depth of the state space