CS 1571 Introduction to AI
Lecture 2

Applications of AI
Problem solving by searching

Milos Hauskrecht
milos@cs.pitt.edu
5329 Sennott Square

Course administrivia

Instructor: Milos Hauskrecht
5329 Sennott Square
milos@cs.pitt.edu

TA: Peter Djalaliev
6503 Sennott Square
peterdj@cs.pitt.edu

Course web page:
http://www.cs.pitt.edu/~milos/courses/cs1571/

AI applications: Software systems.

Diagnosis of: software, technical components

Adaptive systems

- **Adapt to the user**

- **Examples:**
  - **Intelligent interfaces**
  - **Intelligent helper applications**, intelligent tutoring systems
  - **Web applications:**
    - softbots, shopbots (see e.g. http://www.botspot.com/)

AI applications: information retrieval.

- **Web search engines**
  - Improve the quality of search
  - Rely on methods developed in AI
  - Add inferences

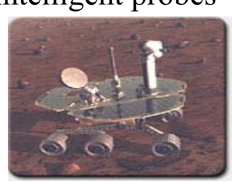
- **Semantic web (or web 2):**
  - From information to knowledge sharing
  - Ontology languages

AI applications: Speech recognition.

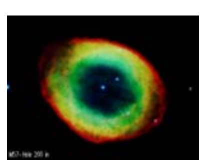
- **Speech recognition systems:**
  - Early systems based on the Hidden Markov models
- **Adaptive speech systems**
  - Adapt to the user (training)
  - continuous speech
  - commercially available software – (Nuance, IBM) http://www.nuance.com/naturallyspeaking/
- **Multi-user speech recognition systems**
  - Restricted (no training)
  - Voice command/voice activated devices
  - Customer support systems:
    - Airline schedules, baggage tracking;
    - Credit card companies

Applications: Space exploration

Autonomous rovers, intelligent probes



Analysis of data

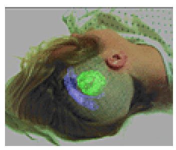


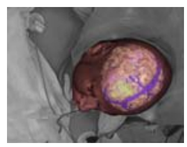
Telescope scheduling

AI applications: Medicine.

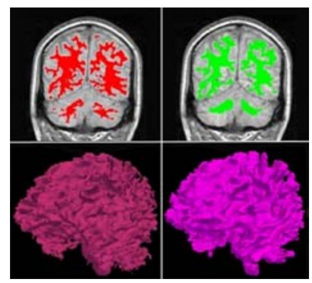
- **Medical diagnosis:** QMR system. Internal medicine.
- **Patient Monitoring and Alerting:** Cerner
- **Medical imaging**

  http://groups.csail.mit.edu/vision/medical-vision/index.html
  - Image guided surgery



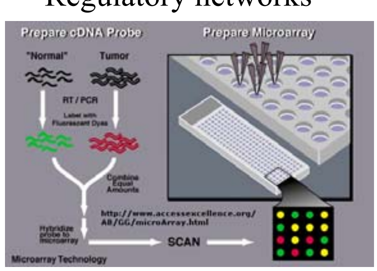


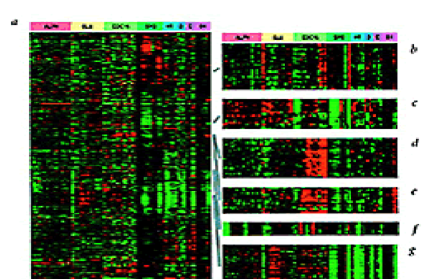
  - Classification of body structures and visualization



AI applications: Bioinformatics.

- **Genomics and Proteomics**
  - Sequence analysis
  - Prediction of gene regions on DNA
  - Analysis of DNA micro-array and proteomic MS profiles: find genes, proteins (peptides) that characterize a specific disease
  - Regulatory networks





Example of a microarray used in gene sequencing

AI applications: Transportation.

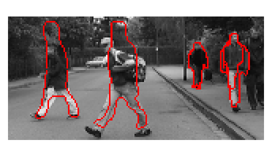
Autonomous vehicle control:

- ALVINN (CMU, Pomerleau 1993) .
  - Autonomous vehicle
  - Driving across US
- DARPA challenge (http://www.darpa.mil/grandchallenge/)
  - Drive across Mojave desert
  - 2004 – no vehicle finished the course
  - 2005 – 5 vehicles finished
    - Stanford team won
  - 2007 - DARPA Urban Challenge
    - 60 miles in urban area settings
    - 6 vehicles finished, CMU won

AI applications: Transportation.

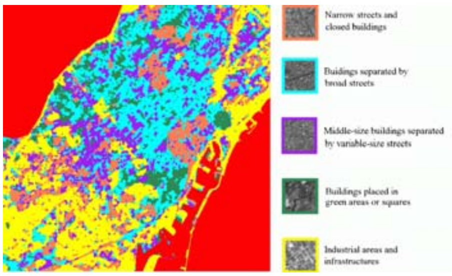
- **Vision systems:**
  - Automatic plate recognition

  - Pedestrian detection (Daimler-Benz)

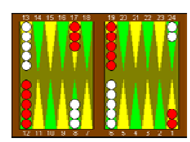


  - Traffic monitoring
- **Route optimizations**

Classification of images or its parts



AI applications: Game playing.



- **Backgammon**
  - TD-backgammon
    - a program that learned to play at the championship level (from scratch).
    - reinforcement learning
- **Chess**
  - Deep blue (IBM) program beats Kasparov.

- **Bridge**

- **Etc.**

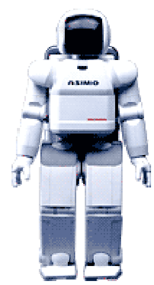
AI applications.

- **Robotic toys**
  - Sony's Aibo
  (http://www.us.aibo.com/)



- **Irobot's Roomba vacuum cleaners**

- **Humanoid robot**
  - Honda's ASIMO
  (http://world.honda.com/robot/)



Other application areas

- **Text classification, document sorting**:
  - Web pages, e-mails
  - Articles in the news
- **Video, image classification**
- **Music composition, picture drawing**
- **Entertainment** ☺

Topics

- **Problem solving and search.**
  - Formulating a search problem, Search methods, Combinatorial and Parametric Optimization.
- **Logic and knowledge representations.**
  - Logic, Inference
- **Planning.**
  - Situation calculus, STRIPS, Partial-order planners,
- **Uncertainty.**
  - Modeling uncertainty, Bayesian belief networks, Inference in BBNs, Decision making in the presence of uncertainty.
- **Machine Learning.**
  - A little

Problem solving by searching

Example

- Assume a problem of computing the roots of the quadratic equation

$$ax^2 + bx + c = 0$$

Do you consider it a challenging problem?

Hardly, we just apply the standard formula:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Solving problems by searching

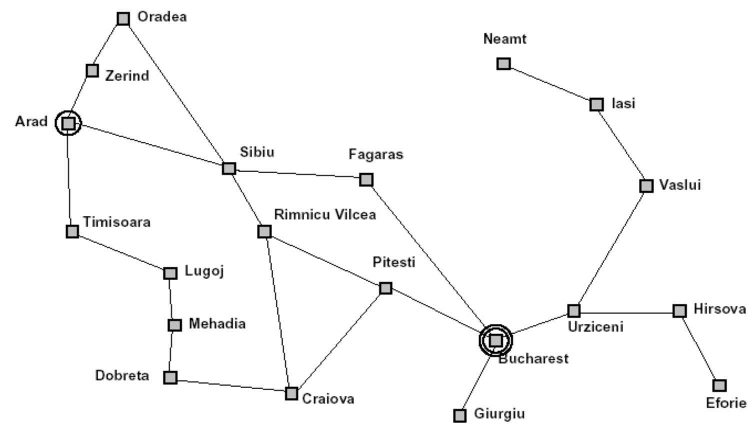
- Some problems have a straightforward solution
  - Just apply a known formula, or follow a standardized procedure

    Example: solution of the quadratic equation
  - Hardly a sign of intelligence

- More interesting problems require **search**:
  - more than one possible alternative needs to be explored before the problem is solved
  - the number of alternatives to search among can be very large, even infinite.

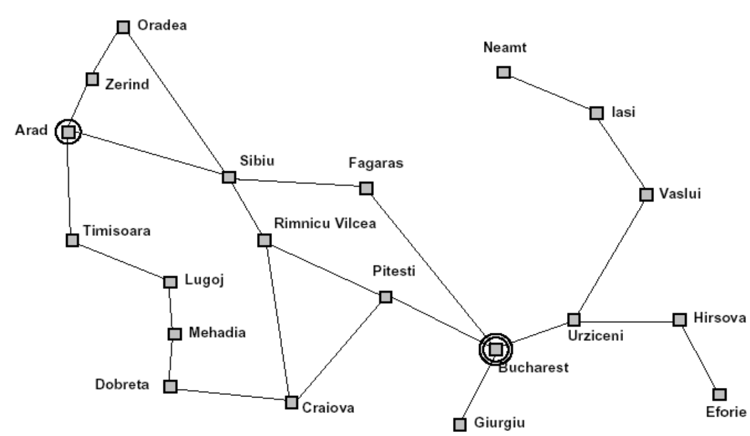
Search example: Traveler problem

- Find a route from one city (**Arad**) to the other (**Bucharest**)



Example. Traveler problem

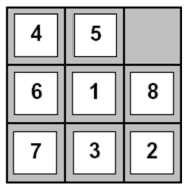
- Another flavor of the traveler problem:
  - find the route with **the minimum length** between S and T



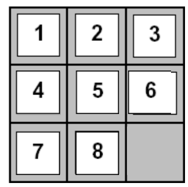
Example. Puzzle 8.

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

Initial position

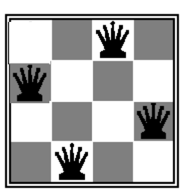


Goal position

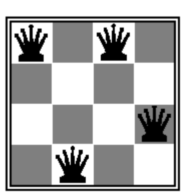


Example. N-queens problem.

Find a configuration of n queens not attacking each other



A goal configuration



A bad configuration

A search problem

is defined by:

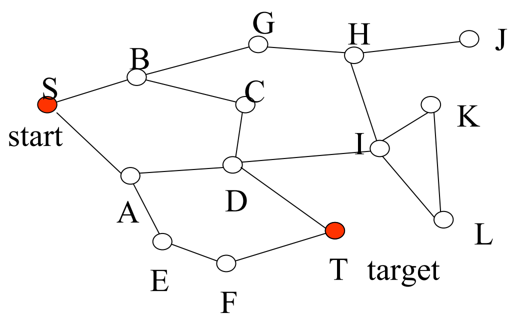
- **A search space**:
  - The set of objects among which we search for the solution Example: objects = routes between cities, or N-queen configurations
- **A goal condition**
  - What are the characteristics of the object we want to find in the search space?
  - Examples:
    - Path between cities A and B
    - Path between A and B with the smallest number of links
    - Path between A and B with the shortest distance
    - Non-attacking n-queen configuration

Search

- **Search (process)**
  - The process of exploration of the search space
- **The efficiency of the search depends on:**
  - The search space and its size
  - Method used to explore (traverse) the search space
  - Condition to test the satisfaction of the search objective (what it takes to determine I found the desired goal object)
- **Important to remember !!!**
  - You can choose the **search space** and the **exploration policy**
  - These choices can have a profound effect on the efficiency of the solution

Graph search

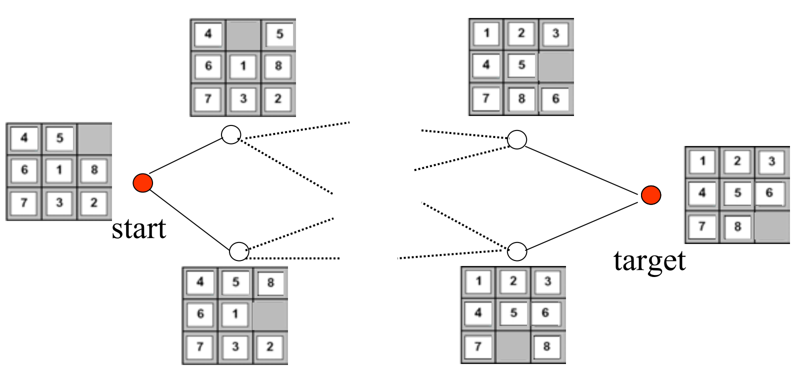
- **Many search problems can be naturally represented as graph search problems**
- **Typical example: Route finding**
  – Map corresponds to the graph, nodes to cities, links to available connections between cities
  – **Goal:** find a route (path) in the graph from S to T



Graph search

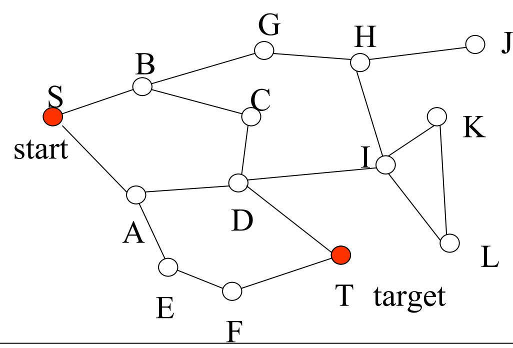
- **Less obvious conversion:**

  Puzzle 8. Find a sequence of moves from the initial configuration to the goal configuration.
  – nodes corresponds to states of the game,
  – links to valid moves made by the player



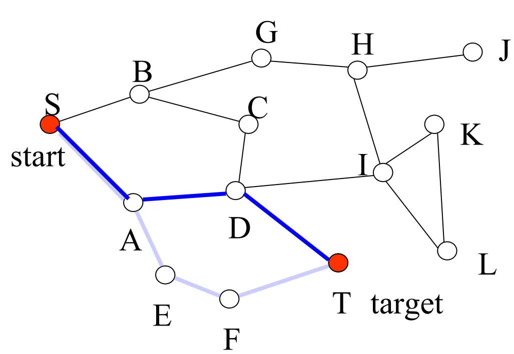
Graph search problem

- **States** - game positions, or locations in the map that are represented by nodes in the graph
- **Operators -** connections between cities, valid moves
- **Initial state** – start position, start city
- **Goal state** – target position (positions), target city (cities)



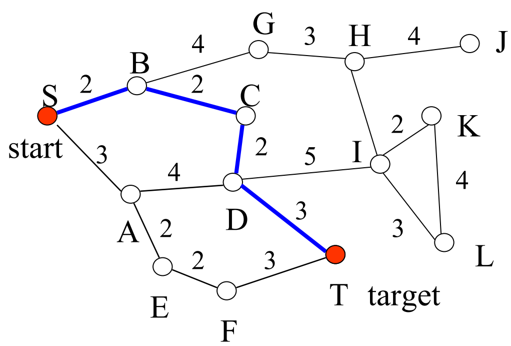
Graph search

- **More complex versions of the graph search problems:**
  – Find a minimal length path
    (= a route with the smallest number of connections, the shortest sequence of moves that solves Puzzle 8)



Graph search

- **More complex versions of the graph search problems:**
  - Find a minimum cost path
    (= a route with the shortest distance)

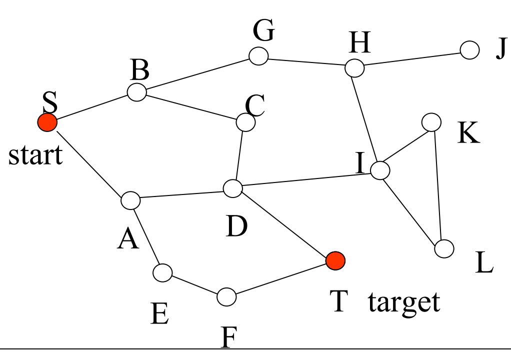


Graph search

How to find the path in between S and T ?

A strawman solution:

1. **Generate systematically all sequences of 1, 2, 3, … edges**
2. Check if the sequence yields a path between S and T.



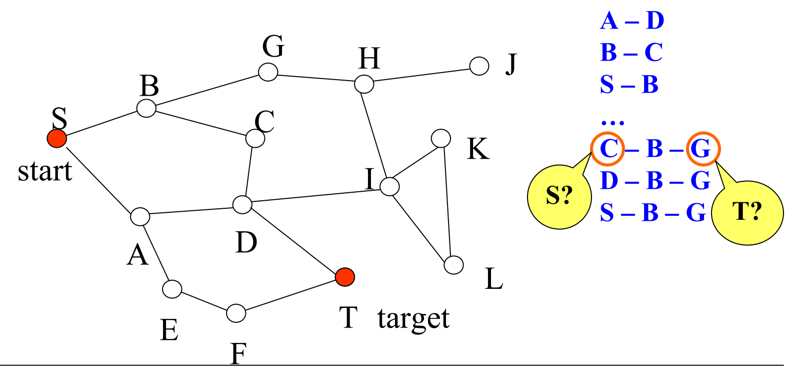
A – D
B – C
S – B
…
C – B – G
D – B – G
S – B – G
…

Graph search

How to find the path in between S and T ?

A strawman solution:

1. Generate systematically all sequences of 1, 2, 3, … edges
2. **Check if the sequence yields a path between S and T.**

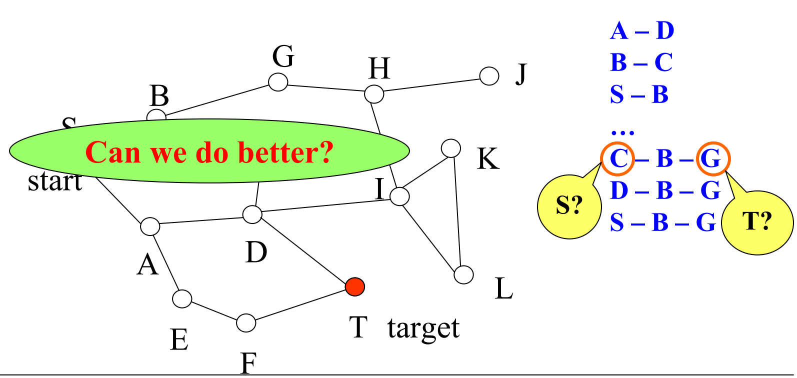


Graph search

How to find the path in between S and T ?

A strawman solution:

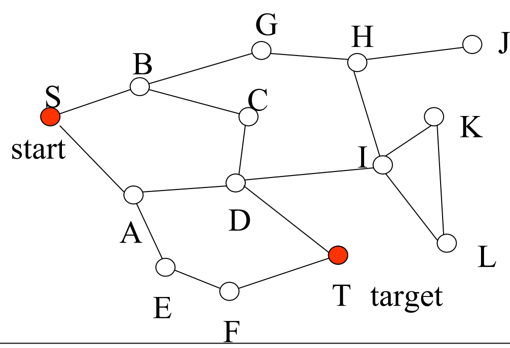
1. Generate systematically all sequences of 1, 2, 3, … edges
2. Check if the sequence yields a path between S and T.



Graph search

Can we do better?

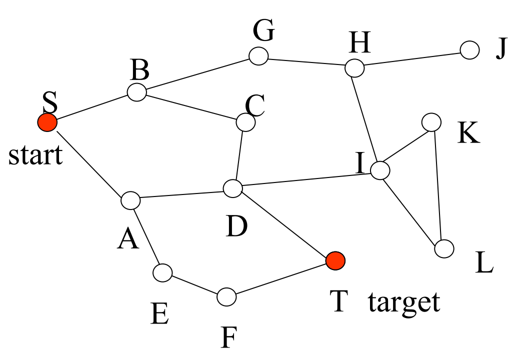
- We are not interested in sequences that do not start in S and that are not valid paths
- **Solution:**



S – A
S – B
S – B – C
S – B – G
S – A – D
….

Graph search

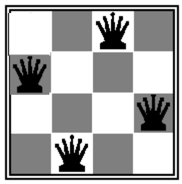
- Being smarter about the space we search for the solution pays off in terms of the search process efficiency**.**



N-queens

Some problems can be converted to the graph search problems

- **But some problems are harder and less intuitive**
  - Take e.g. N-queens problem.



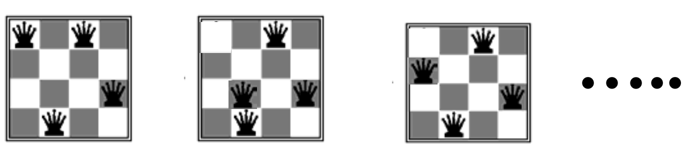
Goal configuration

- **Problem:**
  - **We look for a configuration, not a sequence of moves**
  - **No distinguished initial state, no operators (moves)**

N-queens

How to choose the search space for N-queens?

- Ideas? **Search space:**
  - all configurations of N queens on the board



- **Can we convert it to a graph search problem?**
- We need states, operators, initial state and goal condition.

N-queens

Search space:

– all configurations of N queens on the board

- **Can we convert it to a graph search problem?**
- We need states, operators, initial state and goal state.

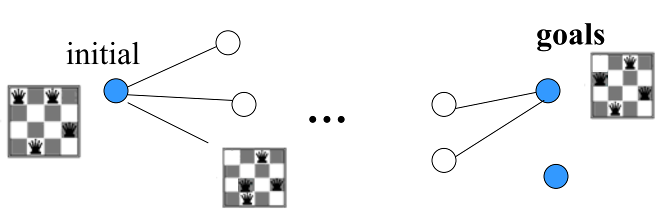
States are: N-queen configurations
Initial state: ?
Operators (moves)?

N-queens

Search space:

– all configurations of N queens on the board

- **Can we convert it to a graph search problem?**
- We need states, operators, initial state and goal condition.

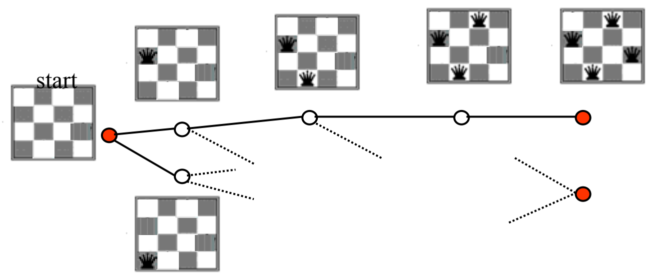


Initial state: an arbitrary N-queen configuration
Operators (moves): change a position of one queen

N-queens

Is there an alternative way to formulate the N-queens problem as a search problem?

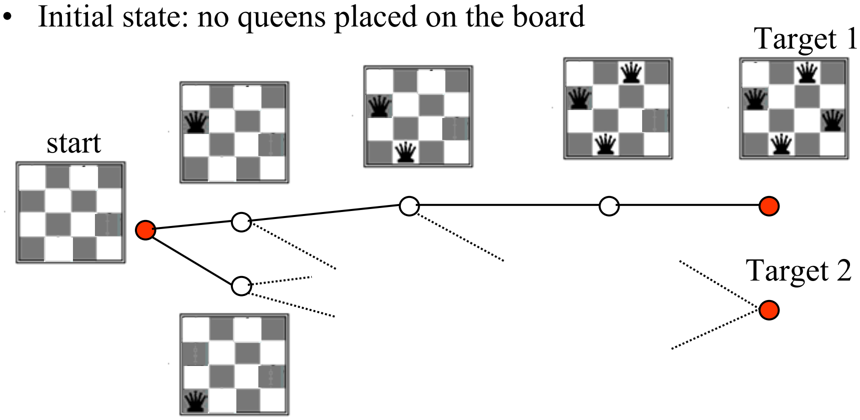
- **Search space:** configurations of 0,1,2, … N queens
- Graph search:
  - – States configurations of 0,1,2,…N queens
  - – Operators: additions of a queen to the board
  - – Initial state: 0 queens on the board



Graph search

A trick: generate a configuration step by step (one queen per step)

- States (nodes) correspond to configurations of 0,1,2,3,4 queens
- Links (operators) correspond to the addition of a queen
- Initial state: no queens placed on the board

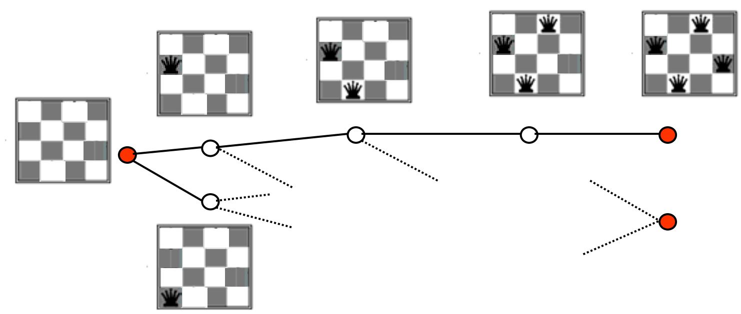


Graph search

N-queens problems

- This 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

- **Path search**
  - Find a path between states S and T
  - **Example:** traveler problem, Puzzle 8
  - **Additional goal criterion:** minimum length (cost) path

- **Configuration search (constraint satisfaction search)**
  - Find a state (configuration) satisfying the goal condition
  - **Example:** n-queens problem, design of a device with a predefined functionality
  - **Additional goal criterion:** "soft" preferences for configurations, e.g. minimum cost design

Search problem

Search problems that can be represented or converted into a graph search problems can be defined in terms of:

- **Initial state**
  - State (configuration) we start to search from (e.g. start city, initial game position)
- **Operators**:
  - Transform one state to another (e.g. valid connections between cities, valid moves in Puzzle 8)
- **Goal condition:**
  - Defines the target state (destination, winning position)
- **Search space (**the set of objects we search for the solution) **:**
  - is now defined indirectly through**:**

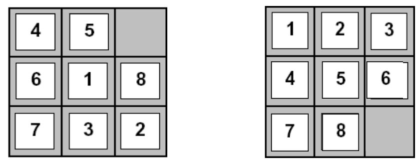
    the initial state + operators

Traveler problem.

Traveler problem formulation:

- **States:** different cities
- **Initial state:** city Arad
- **Operators:** moves to cities in the neighborhood
- **Goal condition: city** Bucharest
- **Type of the problem:** path search
- **Possible solution cost:** path length

Puzzle 8 example

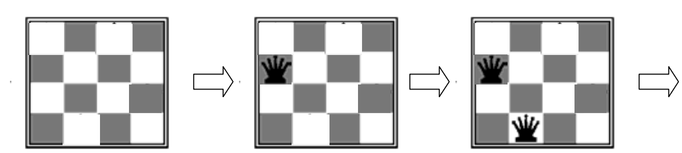


Initial state **Goal state**

Search problem formulation:

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

N-queens problem



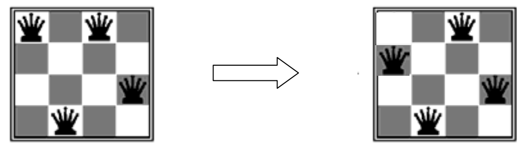
Initial configuration

Problem formulation:

- **States:** configurations of 0 to 4 queens on the board
- **Initial state:** no-queen configuration
- **Operators:** add a queen to the leftmost unoccupied column
- **Goal:** a configuration with 4 non-attacking queens
- **Type of the problem**: configuration search

N-queens problem

Alternative formulation of N-queens problem



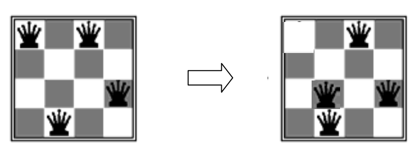
Bad goal configuration   Valid goal configuration

Problem formulation:

- **States:** different configurations of 4 queens on the board
- **Initial state:** an arbitrary configuration of 4 queens
- **Operators:** move one queen to a different unoccupied position
- **Goal:** a configuration with non-attacking queens
- **Type of the problem:** configuration search

Comparison of two problem formulations

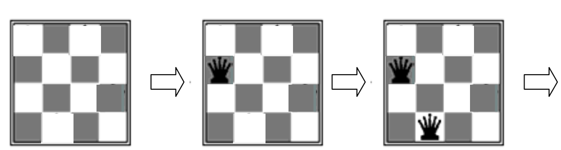
Solution 1:



Operators: switch one of the queens

$\binom{16}{4}$ - all configurations

Solution 2:

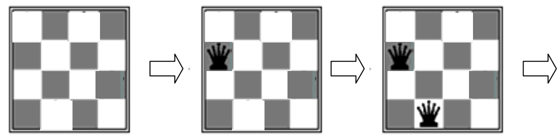


Operators: add a queen to the leftmost unoccupied column

$1 + 4 + 4^2 + 4^3 + 4^4 < 4^5$ - configurations altogether

Even better solution to the N-queens

Solution 2:



Operators: add a queen to the leftmost unoccupied column

$< 4^5$ - configurations altogether

Improved solution with a smaller search space

Operators: add a queen to the leftmost unoccupied column such that it does not attack already placed queens

$$\leq 1 + 4 + 4 * 3 + 4 * 3 * 2 + 4 * 3 * 2 * 1 = 65$$

- configurations altogether

Formulating a search problem

- **Search (process)**
  - The process of exploration of the search space
- **The efficiency of the search depends on:**
  - The search space and its size
  - Method used to explore (traverse) the search space
  - Condition to test the satisfaction of the search objective (what it takes to determine I found the desired goal object)
- **Think twice before solving the problem by search:**
  - Choose the **search space** and the **exploration policy**