COMP250: Artificial Intelligence

7: Navigation

FALMOUTH
UNIVERSITY

Research journal

Research wiki check-in

Research journal submission

Research journal submission

- The deadline is rapidly approaching!

Research journal submission

- The deadline is rapidly approaching!
- **Everybody** must submit a copy of the wiki via LearningSpace

Research journal submission

- The deadline is rapidly approaching!
- **Everybody** must submit a copy of the wiki via LearningSpace
  - Clone the wiki using Git: `https://github.com/Falmouth-Games-Academy/comp250-wiki.wiki.git`

Research journal submission

- The deadline is rapidly approaching!
- **Everybody** must submit a copy of the wiki via LearningSpace
  - Clone the wiki using Git: `https://github.com/Falmouth-Games-Academy/comp250-wiki.wiki.git`
  - Make sure you are cloning the correct repo! It should have all of the wiki content in `.md` files

Research journal submission

- The deadline is rapidly approaching!
- **Everybody** must submit a copy of the wiki via LearningSpace
  - Clone the wiki using Git: `https://github.com/Falmouth-Games-Academy/comp250-wiki.wiki.git`
  - Make sure you are cloning the correct repo! It should have all of the wiki content in `.md` files
  - Zip your cloned repo and upload it

FALMOUTH UNIVERSITY

Pathfinding

The problem

- We have a **graph**

The problem

- We have a **graph**
  - **Nodes** (points)

The problem

- We have a **graph**
  - **Nodes** (points)
  - **Edges** (lines between points, each with a **length**)

The problem

- We have a **graph**
  - **Nodes** (points)
  - **Edges** (lines between points, each with a **length**)
- E.g. a road map

The problem

- We have a **graph**
  - **Nodes** (points)
  - **Edges** (lines between points, each with a **length**)
- E.g. a road map
  - Nodes = addresses

The problem

- We have a **graph**
  - **Nodes** (points)
  - **Edges** (lines between points, each with a **length**)
- E.g. a road map
  - Nodes = addresses
  - Edges = roads

The problem

- We have a **graph**
  - **Nodes** (points)
  - **Edges** (lines between points, each with a **length**)
- E.g. a road map
  - Nodes = addresses
  - Edges = roads
- E.g. a tile-based 2D game

The problem

- We have a **graph**
  - **Nodes** (points)
  - **Edges** (lines between points, each with a **length**)
- E.g. a road map
  - Nodes = addresses
  - Edges = roads
- E.g. a tile-based 2D game
  - Nodes = grid squares

The problem

- We have a **graph**
  - **Nodes** (points)
  - **Edges** (lines between points, each with a **length**)
- E.g. a road map
  - Nodes = addresses
  - Edges = roads
- E.g. a tile-based 2D game
  - Nodes = grid squares
  - Edges = connections between adjacent squares

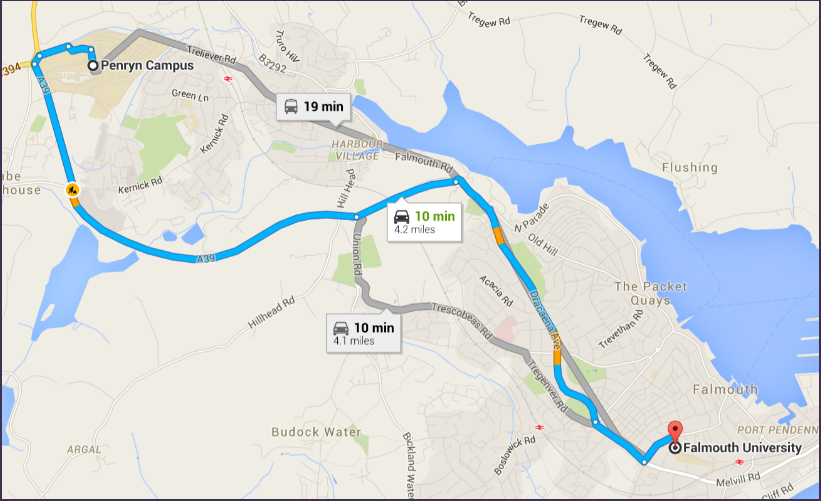
The problem

- We have a **graph**
  - **Nodes** (points)
  - **Edges** (lines between points, each with a **length**)
- E.g. a road map
  - Nodes = addresses
  - Edges = roads
- E.g. a tile-based 2D game
  - Nodes = grid squares
  - Edges = connections between adjacent squares
- Given two nodes A and B, find the **shortest path** from A to B

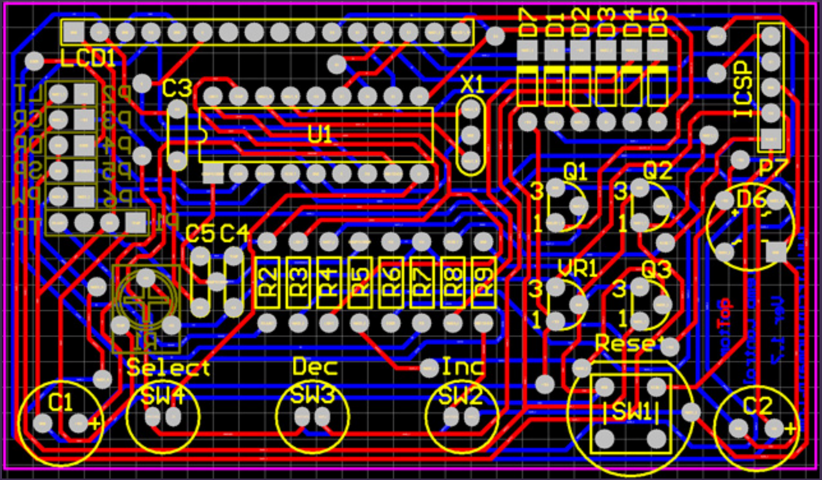
The problem

- We have a **graph**
  - **Nodes** (points)
  - **Edges** (lines between points, each with a **length**)
- E.g. a road map
  - Nodes = addresses
  - Edges = roads
- E.g. a tile-based 2D game
  - Nodes = grid squares
  - Edges = connections between adjacent squares
- Given two nodes A and B, find the **shortest path** from A to B
  - "Shortest" in terms of edge lengths — could be distance, time, fuel cost, ...

Applications of pathfinding



Applications of pathfinding



FALMOUTH UNIVERSITY

Applications of pathfinding

Many applications in game AI

Applications of pathfinding

Many applications in game AI

- Non-player character AI

Applications of pathfinding

Many applications in game AI

- Non-player character AI
- Mouse-based movement (e.g. strategy games)

Applications of pathfinding

Many applications in game AI

- Non-player character AI
- Mouse-based movement (e.g. strategy games)
- Maze navigation

Applications of pathfinding

Many applications in game AI

- Non-player character AI
- Mouse-based movement (e.g. strategy games)
- Maze navigation
- Puzzle solving

Pathfinding example

Pathfinding example

- https://github.com/falmouth-games-academy/comp250-live-coding

Pathfinding example

- `https://github.com/falmouth-games-academy/comp250-live-coding`
- Open `07_pathfinding` in PyCharm

Graph traversal

Graph traversal

- **Depth-first** or **breadth-first**

Graph traversal

- **Depth-first** or **breadth-first**
- Recall: can be implemented with a **stack** or a **queue** respectively

Graph traversal

- **Depth-first** or **breadth-first**
- Recall: can be implemented with a **stack** or a **queue** respectively
- For graphs (as opposed to trees), need to remember which nodes have been **visited** to avoid getting stuck in a loop

Graph traversal

- **Depth-first** or **breadth-first**
- Recall: can be implemented with a **stack** or a **queue** respectively
- For graphs (as opposed to trees), need to remember which nodes have been **visited** to avoid getting stuck in a loop
- Inefficient — generally has to explore the **entire map**

Graph traversal

- **Depth-first** or **breadth-first**
- Recall: can be implemented with a **stack** or a **queue** respectively
- For graphs (as opposed to trees), need to remember which nodes have been **visited** to avoid getting stuck in a loop
- Inefficient — generally has to explore the **entire map**
- Finds a path, but probably not the **shortest**

Graph traversal

- **Depth-first** or **breadth-first**
- Recall: can be implemented with a **stack** or a **queue** respectively
- For graphs (as opposed to trees), need to remember which nodes have been **visited** to avoid getting stuck in a loop
- Inefficient — generally has to explore the **entire map**
- Finds a path, but probably not the **shortest**
- Third type of traversal: **best-first**

Graph traversal

- **Depth-first** or **breadth-first**
- Recall: can be implemented with a **stack** or a **queue** respectively
- For graphs (as opposed to trees), need to remember which nodes have been **visited** to avoid getting stuck in a loop
- Inefficient — generally has to explore the **entire map**
- Finds a path, but probably not the **shortest**
- Third type of traversal: **best-first**
  - "Best" according to some heuristic evaluation

Graph traversal

- **Depth-first** or **breadth-first**
- Recall: can be implemented with a **stack** or a **queue** respectively
- For graphs (as opposed to trees), need to remember which nodes have been **visited** to avoid getting stuck in a loop
- Inefficient — generally has to explore the **entire map**
- Finds a path, but probably not the **shortest**
- Third type of traversal: **best-first**
  - "Best" according to some heuristic evaluation
  - Often implemented with a **priority queue**

Greedy search

Greedy search

- Always try to move **closer** to the goal

Greedy search

- Always try to move **closer** to the goal
- Visit the node whose **distance to the goal** is **minimal**

Greedy search

- Always try to move **closer** to the goal
- Visit the node whose **distance to the goal** is **minimal**
- Doesn't handle **dead ends** well

Greedy search

- Always try to move **closer** to the goal
- Visit the node whose **distance to the goal** is **minimal**
- Doesn't handle **dead ends** well
- Not guaranteed to find the **shortest** path

Dijkstra's algorithm

Dijkstra's algorithm

- Let $g(x)$ be the distance of the path found from the start to x

Dijkstra's algorithm

- Let $g(x)$ be the distance of the path found from the start to x
- Choose a node that minimises $g(x)$

Dijkstra's algorithm

- Let $g(x)$ be the distance of the path found from the start to x
- Choose a node that minimises $g(x)$
- Needs to handle cases where a shorter path to a node is discovered later in the search

Dijkstra's algorithm

- Let $g(x)$ be the distance of the path found from the start to x
- Choose a node that minimises $g(x)$
- Needs to handle cases where a shorter path to a node is discovered later in the search
- **Is** guaranteed to find the shortest path

Dijkstra's algorithm

- Let $g(x)$ be the distance of the path found from the start to x
- Choose a node that minimises $g(x)$
- Needs to handle cases where a shorter path to a node is discovered later in the search
- **Is** guaranteed to find the shortest path
- ... but is not the most efficient algorithm for doing so

A^* search

A* search

- Let $h(x)$ be an estimate of the distance from x to the goal (as in greedy search)

A^* search

- Let $h(x)$ be an estimate of the distance from x to the goal (as in greedy search)
- Let $g(x)$ be the distance of the path found from the start to x (as in Dijkstra's algorithm)

A^* search

- Let $h(x)$ be an estimate of the distance from x to the goal (as in greedy search)
- Let $g(x)$ be the distance of the path found from the start to x (as in Dijkstra's algorithm)
- Choose a node that minimises $g(x) + h(x)$

Properties of A* search

- A* is **guaranteed** to find the shortest path if the distance estimate $h(x)$ is **admissible**

Properties of A* search

- A* is **guaranteed** to find the shortest path if the distance estimate $h(x)$ is **admissible**
- Essentially, **admissible** means it must be an **underestimate**

Properties of A* search

- A* is **guaranteed** to find the shortest path if the distance estimate $h(x)$ is **admissible**
- Essentially, **admissible** means it must be an **underestimate**
  - E.g. straight line Euclidean distance is clearly an underestimate for actual travel distance

Properties of A^* search

- A^* is **guaranteed** to find the shortest path if the distance estimate $h(x)$ is **admissible**
- Essentially, **admissible** means it must be an **underestimate**
  - E.g. straight line Euclidean distance is clearly an underestimate for actual travel distance
- The more accurate $h(x)$ is, the more efficient the search

Properties of A^* search

- A^* is **guaranteed** to find the shortest path if the distance estimate $h(x)$ is **admissible**
- Essentially, **admissible** means it must be an **underestimate**
    - E.g. straight line Euclidean distance is clearly an underestimate for actual travel distance
- The more accurate $h(x)$ is, the more efficient the search
    - E.g. $h(x) = 0$ is admissible (and gives Dijkstra's algorithm), but not very helpful

Properties of A* search

- A* is **guaranteed** to find the shortest path if the distance estimate $h(x)$ is **admissible**
- Essentially, **admissible** means it must be an **underestimate**
  - E.g. straight line Euclidean distance is clearly an underestimate for actual travel distance
- The more accurate $h(x)$ is, the more efficient the search
  - E.g. $h(x) = 0$ is admissible (and gives Dijkstra's algorithm), but not very helpful
- $h(x)$ is a **heuristic**

Properties of A^* search

- A^* is **guaranteed** to find the shortest path if the distance estimate $h(x)$ is **admissible**
- Essentially, **admissible** means it must be an **underestimate**
  - E.g. straight line Euclidean distance is clearly an underestimate for actual travel distance
- The more accurate $h(x)$ is, the more efficient the search
  - E.g. $h(x) = 0$ is admissible (and gives Dijkstra's algorithm), but not very helpful
- $h(x)$ is a **heuristic**
  - In AI, a heuristic is an estimate based on human intuition

Properties of A* search

- A* is **guaranteed** to find the shortest path if the distance estimate $h(x)$ is **admissible**
- Essentially, **admissible** means it must be an **underestimate**
  - E.g. straight line Euclidean distance is clearly an underestimate for actual travel distance
- The more accurate $h(x)$ is, the more efficient the search
  - E.g. $h(x) = 0$ is admissible (and gives Dijkstra's algorithm), but not very helpful
- $h(x)$ is a **heuristic**
  - In AI, a heuristic is an estimate based on human intuition
  - Heuristics are often used to prioritise search, i.e. explore the most promising options first

Tweaking A^*

Tweaking A^*

- Can change how $g(x)$ is calculated

Tweaking A^*

- Can change how $g(x)$ is calculated
  - Increased movement cost for rough terrain, water, lava...

Tweaking A*

- Can change how $g(x)$ is calculated
  - Increased movement cost for rough terrain, water, lava...
  - Penalty for changing direction

Tweaking A*

- Can change how $g(x)$ is calculated
  - Increased movement cost for rough terrain, water, lava...
  - Penalty for changing direction
- Different $h(x)$ can lead to different paths (if there are multiple "shortest" paths)

String pulling

String pulling

- Paths restricted to edges can look unnatural

String pulling

- Paths restricted to edges can look unnatural
- Intuition: visualise the path as a string, then pull both ends to make it taut

String pulling

- Paths restricted to edges can look unnatural
- Intuition: visualise the path as a string, then pull both ends to make it taut
- Simple algorithm:

String pulling

- Paths restricted to edges can look unnatural
- Intuition: visualise the path as a string, then pull both ends to make it taut
- Simple algorithm:
  - Found path is $p[0], p[1], \ldots, p[n]$

String pulling

- Paths restricted to edges can look unnatural
- Intuition: visualise the path as a string, then pull both ends to make it taut
- Simple algorithm:
  - Found path is $p[0], p[1], \ldots, p[n]$
  - If the line from $p[i]$ to $p[i+2]$ is unobstructed, remove point $p[i+1]$

String pulling

- Paths restricted to edges can look unnatural
- Intuition: visualise the path as a string, then pull both ends to make it taut
- Simple algorithm:
  - Found path is $p[0], p[1], \ldots, p[n]$
  - If the line from $p[i]$ to $p[i+2]$ is unobstructed, remove point $p[i+1]$
  - Repeat until there are no more points that can be removed

Navigation meshes

Pathfinding in videogames

Pathfinding in videogames

- A* works on any **graph**

Pathfinding in videogames

- A* works on any **graph**
- But what if the game world is not a graph? E.g. complex 3D environments

Waypoint navigation

Waypoint navigation

- Manually place graph nodes in the world

Waypoint navigation

- Manually place graph nodes in the world
- Place them at key points, e.g. in doorways, around obstacles

Waypoint navigation

- Manually place graph nodes in the world
- Place them at key points, e.g. in doorways, around obstacles
- Works, but...

Waypoint navigation

- Manually place graph nodes in the world
- Place them at key points, e.g. in doorways, around obstacles
- Works, but...
  - More work for level designers

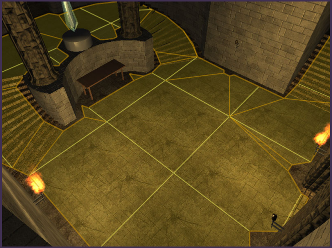
Waypoint navigation

- Manually place graph nodes in the world
- Place them at key points, e.g. in doorways, around obstacles
- Works, but...
  - More work for level designers
  - Requires lots of testing and tweaking to get natural-looking results

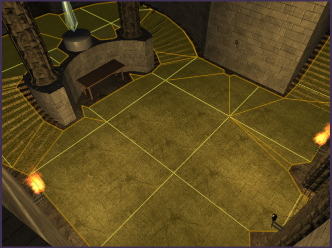
Waypoint navigation

- Manually place graph nodes in the world
- Place them at key points, e.g. in doorways, around obstacles
- Works, but...
  - More work for level designers
  - Requires lots of testing and tweaking to get natural-looking results
  - No good for dynamic environments

Navigation meshes

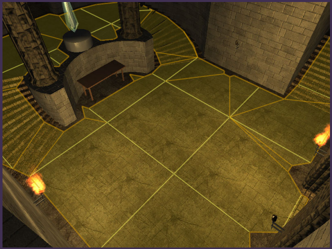


Navigation meshes



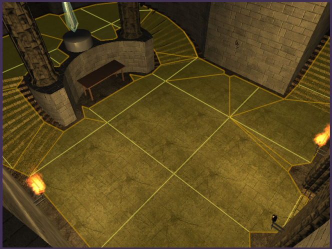
- Automatically generate navigation graph from level geometry

Navigation meshes



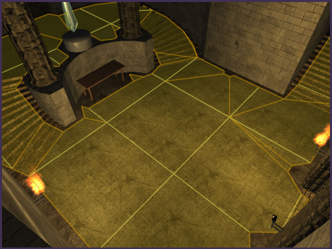
- Automatically generate navigation graph from level geometry
- Basic idea:

Navigation meshes



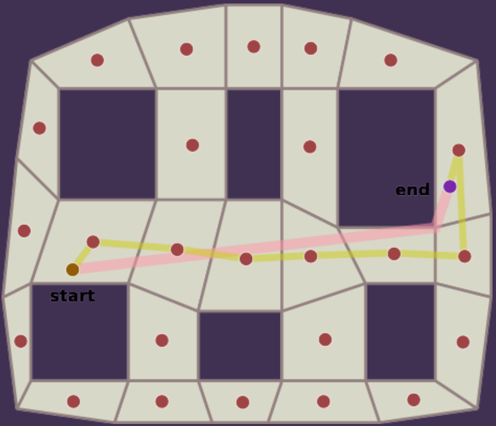
- Automatically generate navigation graph from level geometry
- Basic idea:
  - Filter level geometry to those polygons which are **passable** (i.e. floors, not walls/ceilings/obstacles)

Navigation meshes



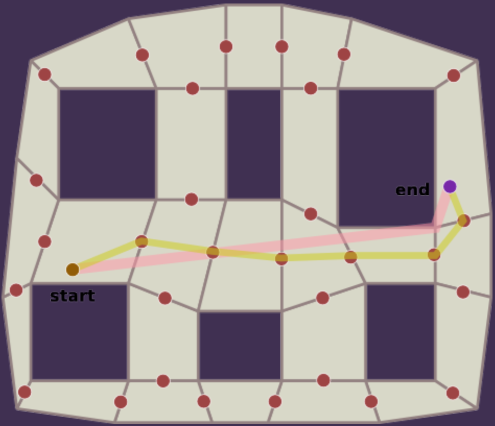
- Automatically generate navigation graph from level geometry
- Basic idea:
  - Filter level geometry to those polygons which are **passable** (i.e. floors, not walls/ceilings/obstacles)
  - Generate graph from polygons

Meshes to graphs



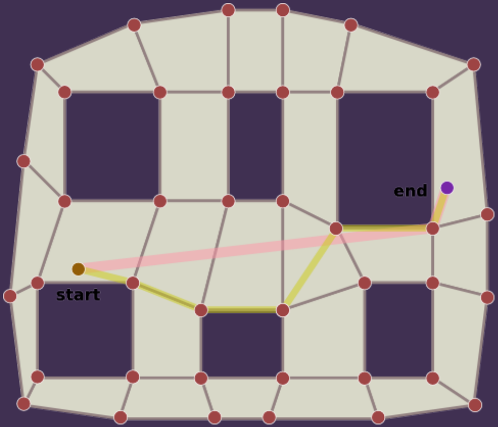
Centres of polygons

Meshes to graphs



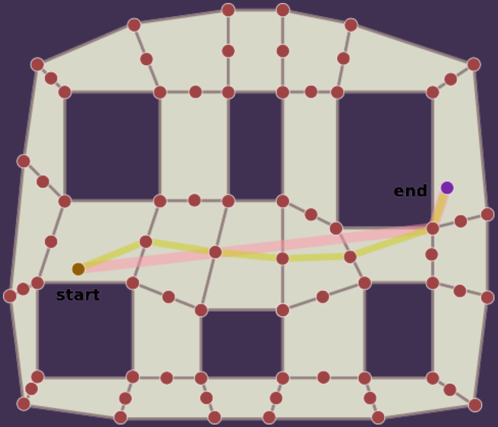
Centres of edges

Meshes to graphs



Vertices of polygons

Meshes to graphs



Hybrid approach: edges and vertices

Following the path

Following the path

- **Funnelling**: like string pulling but for navigation meshes

Following the path

- **Funnelling**: like string pulling but for navigation meshes
  - `http://digestingduck.blogspot.co.uk/2010/03/simple-stupid-funnel-algorithm.html`
  - `http://jceipek.com/Olin-Coding-Tutorials/pathing.html`

Following the path

- **Funnelling**: like string pulling but for navigation meshes
  - `http://digestingduck.blogspot.co.uk/2010/03/simple-stupid-funnel-algorithm.html`
  - `http://jceipek.com/Olin-Coding-Tutorials/pathing.html`
- **Steering**: don't have your AI agent follow the path exactly, but instead try to stay close to it

Following the path

- **Funnelling**: like string pulling but for navigation meshes
  - http://digestingduck.blogspot.co.uk/2010/03/simple-stupid-funnel-algorithm.html
  - http://jceipek.com/Olin-Coding-Tutorials/pathing.html
- **Steering**: don't have your AI agent follow the path exactly, but instead try to stay close to it
- **Dynamic environments**: may need to re-run pathfinder if environment changes (e.g. movable obstacles, destructible terrain)

The travelling salesman problem

The travelling salesman problem (TSP)

The travelling salesman problem (TSP)

- Classic problem in Computer Science

The travelling salesman problem (TSP)

- Classic problem in Computer Science
- We have a **graph**

The travelling salesman problem (TSP)

- Classic problem in Computer Science
- We have a **graph**
- From starting node S, find the **shortest possible path** that visits every node **exactly once** and returns to S

The travelling salesman problem (TSP)

- Classic problem in Computer Science
- We have a **graph**
- From starting node S, find the **shortest possible path** that visits every node **exactly once** and returns to S
- Many real-world applications

The travelling salesman problem (TSP)

- Classic problem in Computer Science
- We have a **graph**
- From starting node S, find the **shortest possible path** that visits every node **exactly once** and returns to S
- Many real-world applications
  - Transport and logistics

The travelling salesman problem (TSP)

- Classic problem in Computer Science
- We have a **graph**
- From starting node S, find the **shortest possible path** that visits every node **exactly once** and returns to S
- Many real-world applications
  - Transport and logistics
  - Manufacturing

The travelling salesman problem (TSP)

- Classic problem in Computer Science
- We have a **graph**
- From starting node S, find the **shortest possible path** that visits every node **exactly once** and returns to S
- Many real-world applications
    - Transport and logistics
    - Manufacturing
    - Playing Pac-Man

The travelling salesman problem (TSP)

- Classic problem in Computer Science
- We have a **graph**
- From starting node S, find the **shortest possible path** that visits every node **exactly once** and returns to S
- Many real-world applications
  - Transport and logistics
  - Manufacturing
  - Playing Pac-Man
  - Pub crawls (`http://www.math.uwaterloo.ca/tsp/pubs/`)

Solving TSP

Solving TSP

- TSP is **NP-complete**

Solving TSP

- TSP is **NP-complete**
  - If $P \neq NP$, then there is **no** polynomial-time algorithm for solving it

Solving TSP

- TSP is **NP-complete**
  - If $P \neq NP$, then there is **no** polynomial-time algorithm for solving it
- Entire research field devoted to finding efficient **search algorithms** and **heuristics**