UNIVERSITY OF BIRMINGHAM

Artificial Intelligence 1
Optimisation

Dr Leonardo Stella

School of Computer Science

2023/2024 - Week 9

Aims of the Session

This session aims to help you:

- Understand how to formulate an optimisation problem
- Explain the steps involved in Hill Climbing
- Compare the performance of search and optimisation algorithms

Outline

Outline

Optimisation

- Until now, we have addressed problems in fully observable, deterministic, static, known environments
- In this lecture, we will relax some of these constraints and look at optimisation problems
- **Optimisation** problems arise in many disciplines, including computer science, engineering, operations research, etc.
- In an optimisation problem, one seeks to find a solution that minimises or maximises an **objective function** (or more than one)
- Depending on the problem, there can be some **constraints** that must be satisfied for a solution to be **feasible**

Optimisation: An Example

Example. Let us consider the following problem: find the point on the line $y = 2x - 4$ that is closest to the origin.

- How to formulate this problem?

$$\min f(\bar{x}) = L^2([x, y]^\top, O) = \sqrt{(x - x_0)^2 + (y - y_0)^2}$$

$$= \sqrt{x^2 + (2x - 4)^2} = (5x^2 - 16x + 16)^{1/2}$$

- From calculus, we take the derivative of $f(x)$ with respect to x (chain rule) and set to zero:

$$\frac{\mathrm{d}f(\bar{x})}{\mathrm{d}\bar{x}} = \frac{1}{2}(5x^2 - 16x + 16)^{-1/2}(10x - 16)$$

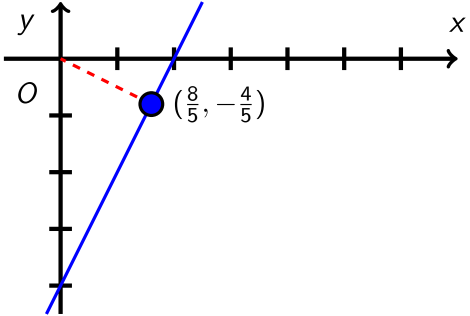
$$= \frac{5x - 8}{\sqrt{5x^2 - 8x + 16}} = 0$$

Optimisation: An Example

Example (continued). Let us consider the following problem: find the point on the line $y = 2x - 4$ that is closest to the origin.

- For the previous formula to be zero, it is sufficient that the numerator is equal to zero:

$$5x - 8 = 0 \Rightarrow x = \frac{8}{5}, \qquad y = 2x - 4 \Rightarrow y = -\frac{4}{5}$$



Formulating an Optimisation Problem

- The canonical representation of an optimisation problem is like in the following:

$$
\begin{aligned}
\min / \max \quad & f(x), \\
\text{s.t.} \quad & g_i(x) \leq 0, \quad i = 1, \ldots, m, \\
& h_j(x) = 0, \quad j = 1, \ldots, n,
\end{aligned}
$$

- where $f(x)$ the objective function, x is the design variable
- $g_i(x)$ and $h_j(x)$ are the constraints
- The **search space** of the problem is the space of all possible x values

Formulating an Optimisation Problem

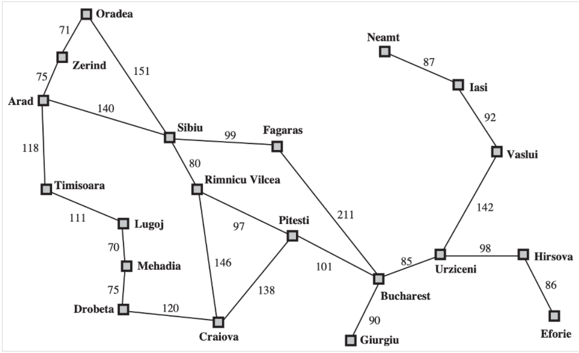
- The design variables represent a candidate solution
- Therefore, the **search space** of a candidate solution is defined by the design variables

- The objective function defines the cost (or quality) of a solution
- This function is the one to be optimised (through min or max)

- The solution must optionally satisfy a number of constraints
- They define the feasibility of the solution

Formulating an Optimisation Problem

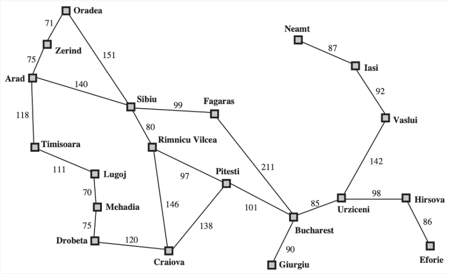
Activity. Consider the following problem: find a path from a city of origin (Arad) to a city of destination (Bucharest) that minimises the distance travelled, while ensuring that direct paths between non-neighbouring cities are not used. In pairs or small groups, formulate the corresponding optimisation problem.



Formulating an Optimisation Problem

Let us define the design variable and search space:

- Recall that design variables represent candidate solutions
- **1-d array** x **of any size** containing the sequence of cities
- Excluding the city of origin and city of destination
- The search space consists of **all possible sequences of cities**

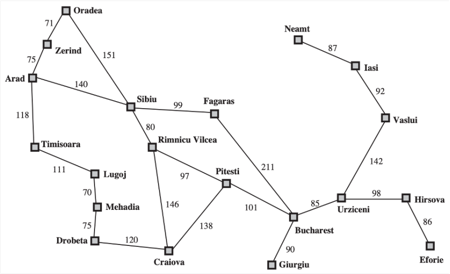


Formulating an Optimisation Problem

Let us define the objective function:

- Recall that objective functions describe the quality of a solution
- **Minimise the sum of the distances** between consecutive cities in the solution

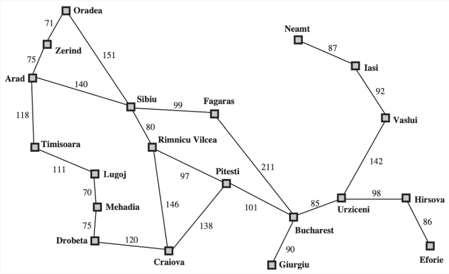
$$\text{Arad} \quad \underbrace{\begin{bmatrix} x_1: & \text{Sibiu} \\ x_2: & \text{Fagaras} \end{bmatrix}}_{x} \quad \text{Bucharest}$$



Formulating an Optimisation Problem

Which solutions are feasible and infeasible?

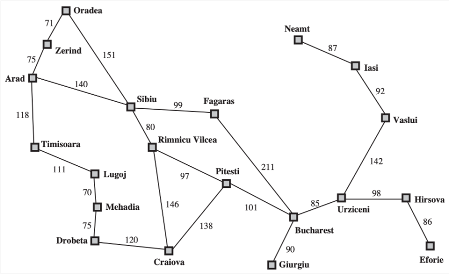
- **Infeasible**: Arad [Rimnicu Vilcea, Mehadia]$^\top$ Bucharest (moves to non-neighbouring cities)
- **Feasible** (non-optimal): Arad [Sibiu, Fagaras]$^\top$ Bucharest (moves to neighbouring cities, not minimal cost)
- **Feasible** (optimal): Arad [Sibiu, Rimnicu Vilcea, Pitesti]$^\top$ Bucharest (minimal cost)



Formulating an Optimisation Problem

Let us now define the constraints, which determine feasibility of a solution:

- **Explicit constraint** (non-existing path): direct paths between non-neighbouring cities must not be used
- **Implicit constraint**: the city of origin must be Arad and city of destination must be Bucharest
- **Implicit constraint**: only cities from the map can be used



Formulating an Optimisation Problem

We can now formalise each component of the problem:

- Set of cities from the map: $C = \{\text{Oradea}, \ldots, \text{Eforie}\}$
- City of origin: $a \in C$; city of destination $b \in C$
- Design variable: vector x, $|x| > 0$; $\forall i \in \{1, \ldots, |x|\}$, $x_i \in C \setminus \{a, b\}$
- Matrix D of distances; $D_{ij} = d_{ij}$ if $\{i, j\} \in \mathcal{E}$; $D_{ij} = -1$, otherwise
- Objective function:

$$f(x) = D_{a,x_1} + \sum_{i=1}^{|x|-1} D_{x_i,x_{i+1}} + D_{x_{|x|},b}$$

- Constraints:

$$h(x) = \begin{cases} 1, & \text{if } D_{a,x_1} = -1 \text{ or } D_{x_{|x|},b} = -1, \\ 1, & \text{if } \exists i \in \{1, \ldots, |x| - 1\} : D_{x_i,x_{i+1}} = -1, \\ 0, & \text{otherwise.} \end{cases}$$

Formulating an Optimisation Problem

Solution. The optimisation problem is:

$$\begin{aligned} \min \quad & f(x) = D_{a,x_1} + \sum_{i=1}^{|x|-1} D_{x_i,x_{i+1}} + D_{x_{|x|},b}, \\ \text{s.t.} \quad & h(x) = 0. \end{aligned}$$

where $|x| > 0$, $\forall i \in \{1, \dots, |x|\}$, $x_i \in C \setminus \{a, b\}$ and:

- $C = \{\text{Oradea}, \dots, \text{Eforie}\}$ is the set of cities from the map
- $a, b \in C$ are the city of origin and destination, respectively
- $D \in \mathbb{R}^{n \times n}$ is the matrix of distances; $D_{ij} = d_{ij}$ if $\{i, j\} \in \mathcal{E}$, d_{ij} being the actual distance between i and j; $D_{ij} = -1$, otherwise
- and the constraint is defined as

$$h(x) = \begin{cases} 1, & \text{if } D_{a,x_1} = -1 \text{ or } D_{x_{|x|},b} = -1, \\ 1, & \text{if } \exists i \in \{1, \dots, |x| - 1\} : D_{x_i,x_{i+1}} = -1, \\ 0, & \text{otherwise.} \end{cases}$$

Outline

Local Search

- We can now formalise an optimisation problem
- In a search problem, we want to find paths through the search space
- For example, finding the shortest path between Arad and Bucharest
- In the previous slides, we formalised the corresponding optimisation problem (or one possible way)
- However, sometimes we are interested only in the final state (e.g., the 8-queen problem discussed in this week's tutorial session)

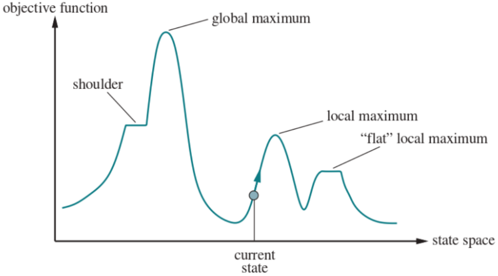
Local Search

- **Local search algorithms** (also referred to as **optimisation algorithms**) operate by searching from an initial (randomised) state to neighbouring states
- These algorithms **do not** keep track of the paths nor the states that have been reached/visited
- This means they are not systematic, but they have advantages:
  - they use very little memory
  - they can often find reasonable solutions in large or infinite state spaces
- Important applications include: integrated-circuit design, factory floor layout, job shop scheduling, portfolio management, etc.

Local Search

Example. Consider the problem of maximising an objective function whose value is defined as the elevation on the y-axis in the figure below (against the state space on the x-axis). The aim is to find the highest peak, i.e., a global maximum. This process is called **hill climbing**.

Example. Consider the opposite problem. The objective function now represents a cost that we want to minimise. The aim is to find the lowest valley, i.e., a global minimum. This process is called **gradient descent**.



Hill Climbing

- **Hill Climbing** is one of the most popular optimisation algorithms:
  - Generate initial candidate solution at random
    - Generate neighbour solutions and move to the one with highest value
    - If no neighbour has a higher value of the current solution, terminate
    - Otherwise, repeat with a new best neighbour solution
- Hill climbing does not look beyond the immediate neighbours of the current state, making it a **greedy algorithm**
- This is equivalent "to trying to find the top of Mount Everest in a thick fog while suffering from amnesia"

Hill Climbing

```
function HILL-CLIMBING(problem) returns a state that is a local maximum
  current ← problem.INITIAL
  while true do
    neighbor ← a highest-valued successor state of current
    if VALUE(neighbor) ≤ VALUE(current) then return current
    current ← neighbor
```

Hill Climbing

- To be able to apply hill climbing, we need to design the following components of the algorithm:
  - **Representation**, how to store the design variable, e.g., boolean, integer, float, array
    A good representation should facilitate the application of the initialisation procedure and neighbourhood operator
  - **Initialisation procedure**, how to pick an initial solution
    Usually, this involves selecting one at random
  - **Neighbourhood operator**, how to generate neighbour solutions

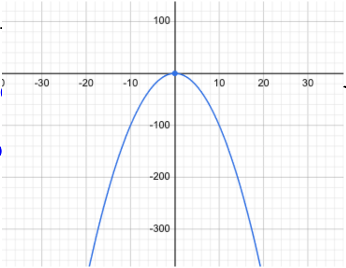
Hill Climbing

Activity. Consider the following optimisation problem:

- **Design variable**: $x \in \mathbb{Z}$
- **State space**: $\mathbb{Z}$
- **Constraints**: none
- **Objective function**: $\max f(x) = -x^2$

In pairs or small groups, define the components of hill climbing.

- Representation: in
- Initialisation proce n integer value
- Neighbourhood op



Performance of Hill Climbing

- Let us evaluate the performance of hill climbing.
  - **Completeness**: hill climbing is not complete, as it depends on the problem formulation and design of the algorithm
  - **Optimality**: hill climbing is not optimal (can get stuck in a local maximum/optimum)
  - **Time complexity**: $\mathcal{O}(mnp)$, where m is the maximum number of iterations, n is the maximum number of neighbours, each of which takes $\mathcal{O}(p)$ to generate
  - **Space complexity**: $\mathcal{O}(nq + r) = \mathcal{O}(nq)$, where the variable takes $\mathcal{O}(q)$ and r represents the space to generate the neighbours sequentially (which is negligible compared to n and q)

Hill Climbing

- Pros:
  - Can rapidly find a good solution by improving over a bad initial state (greedy)
  - Low time and space complexity compared to search algorithms
  - Does not require problem-specific heuristics
  - Start from a candidate solution, instead of building it step-by-step
- Cons:
  - Not guaranteed to be complete, nor optimal
  - Can get stuck in local maxima and plateaus

References

Russell, A. S., and Norvig, P., *Artificial Intelligence A Modern Approach*, 4^{th} Edition. Prentice Hall.

- Chapter 4 – "Search in Complex Environments", Section 4.1 "Local Search and Optimization Problems" up to and including Section 4.1.1 "Hill Climbing Search".

Aims of the Session

You should now be able to:

- Understand how to formulate an optimisation problem
- Explain the steps involved in Hill Climbing
- Compare the performance of search and optimisation algorithms

Thank you!