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

1. Optimisation
2. Hill Climbing

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**

Formulating an Optimisation Problem

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

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

- 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

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)

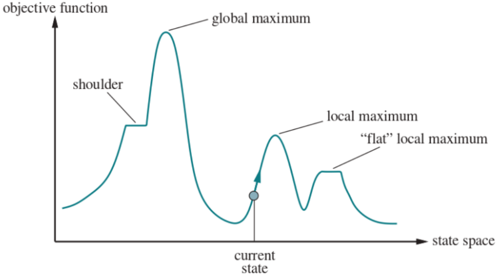
Local Search

- **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

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!