UNIVERSITY OF BIRMINGHAM

Artificial Intelligence 1
Optimisation and Constraints Handling

Dr Leonardo Stella

School of Computer Science

2023/2024 - Week 10

Aims of the Session

This session aims to help you:

- Understand the weaknesses in Hill Climbing and how to mitigate them
- Explain the steps involved in Simulated Annealing
- Deal with constraints in optimisation problems

Outline

1. Recap: Informed Search and Optimisation
2. Hill Climbing: Variants
3. Simulated Annealing
4. Constraint Handling

Outline

1. Recap: Informed Search and Optimisation
2. Hill Climbing: Variants
3. Simulated Annealing
4. Constraint Handling

Recap: Quiz

Q1. Which evaluation function $f(n)$ does $\mathbf{A}^\star$ use?

- ○ $f(n) = c(n)$, where $c(n)$ is the cost to get to n
- ○ $f(n) = h(n)$, where $h(n)$ is the heuristic
- ○ $f(n) = h(n) - c(n)$, where $h(n)$ is the heuristic and $c(n)$ is the cost to get to n
- ○ $f(n) = h(n)^2$, where $h(n)$ is the heuristic
- ○ None of the above

Recap: Quiz

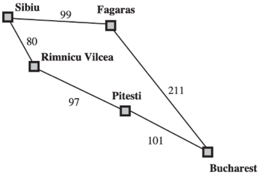
Q1. Which evaluation function $f(n)$ does **A*** use?

- ○ $f(n) = c(n)$, where $c(n)$ is the cost to get to n
- ○ $f(n) = h(n)$, where $h(n)$ is the heuristic
- ○ $f(n) = h(n) - c(n)$, where $h(n)$ is the heuristic and $c(n)$ is the cost to get to n
- ○ $f(n) = h(n)^2$, where $h(n)$ is the heuristic
- ● None of the above

Recap: Quiz

Q2. Given the heuristic in the table below and the cost to reach each city on the map (initial state: Sibiu), which one is the correct value of $f(\text{Pitesti})$ for **greedy best-first search**?

- ○ $f(\text{Pitesti}) = 100$
- ○ $f(\text{Pitesti}) = 197$
- ○ $f(\text{Pitesti}) = 177$
- ○ $f(\text{Pitesti}) = 277$
- ○ None of the above

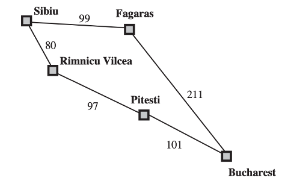


Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Recap: Quiz

Q2. Given the heuristic in the table below and the cost to reach each city on the map (initial state: Sibiu), which one is the correct value of $f(\text{Pitesti})$ for **greedy best-first search**?

- ● $f(\text{Pitesti}) = 100$
- ○ $f(\text{Pitesti}) = 197$
- ○ $f(\text{Pitesti}) = 177$
- ○ $f(\text{Pitesti}) = 277$
- ○ None of the above

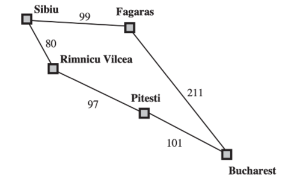


Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Recap: Quiz

Q3. Given the heuristic in the table below and the cost to reach each city on the map (initial state: Sibiu), which one is the correct value of $f(\text{Pitesti})$ for **A***?

- ○ $f(\text{Pitesti}) = 100$
- ○ $f(\text{Pitesti}) = 197$
- ○ $f(\text{Pitesti}) = 177$
- ○ $f(\text{Pitesti}) = 277$
- ○ None of the above

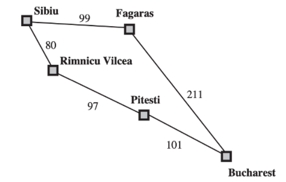


Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Recap: Quiz

Q3. Given the heuristic in the table below and the cost to reach each city on the map (initial state: Sibiu), which one is the correct value of $f(\text{Pitesti})$ for **A***?

- ○ $f(\text{Pitesti}) = 100$
- ○ $f(\text{Pitesti}) = 197$
- ○ $f(\text{Pitesti}) = 177$
- ● $f(\text{Pitesti}) = 277$
- ○ None of the above



Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Recap: Quiz

Q4. What are the components of an optimisation problem formulation?

- ☐ Design variables
- ☐ State space
- ☐ Objective functions
- ☐ Constraints
- ☐ None of the above

Recap: Quiz

Q4. What are the components of an optimisation problem formulation?

- ■ Design variables
- ☐ State space
- ■ Objective functions
- ■ Constraints
- ☐ None of the above

Recap: Quiz

Q5. Local search algorithms **do not** keep track of the paths nor the states that have been reached/visited.

- ○ True
- ○ False

Recap: Quiz

Q5. Local search algorithms **do not** keep track of the paths nor the states that have been reached/visited.

- ● True
- ○ False

Recap: Quiz

Q6. Hill climbing is a greedy algorithm.

- ○ True
- ○ False

Recap: Quiz

Q6. Hill climbing is a greedy algorithm.

- ● True
- ○ False

Outline

1 Recap: Informed Search and Optimisation

2 Hill Climbing: Variants

3 Simulated Annealing

4 Constraint Handling

Weaknesses of Hill Climbing

- Last week, we discussed pros and cons of hill climbing
- Can we mitigate some of its weaknesses?
  - W: it can run for ever if the problem is not properly formulated
  - S: we run the algorithm for a maximum number of iterations m
  - Indeed, we used m to determine its time complexity: $\mathcal{O}(mnp)$

  - W: not guaranteed to be complete, nor optimal
  - S: as long as we are satisfied with a good solution, it can find it quickly

  - W: it can get stuck in local maxima and plateaus
  - S: there are variants of the algorithm to deal with this

Hill Climbing: Variants

- Let us consider an n-dimensional problem
- The gradient determines the uphill moves along the n dimensions
- **Stochastic Hill Climbing**
  - This variant chooses at random among the **uphill moves**
  - The probability of picking a specific move can depend on the **steepness**
  - Converges more slowly than steepest ascent
  - Can find higher quality solutions
- **First-Choice Hill Climbing**
  - Implements stochastic hill climbing by randomly generating successors until a better successor than the current state is generated
  - It performs well when a state has many successors
- **Random-Restart Hill Climbing**
  - Generates a series of hill-climbing searches from random initial states
  - It stops when a goal is found
  - It is complete with probability 1 as it will eventually generate a goal state as initial state

Outline

1. Recap: Informed Search and Optimisation
2. Hill Climbing: Variants
3. Simulated Annealing
4. Constraint Handling

Simulated Annealing

- A hill climbing algorithm that **never makes downhill moves** can get stuck in local maxima (or minima)
- A purely random walk will eventually find the global maximum, but will be **extremely inefficient**
- **Simulated Annealing** combines these two approaches:
  - Generate initial candidate solution at random
    - Generate neighbour solutions, pick one at random (instead of best)
    - If the move improves the current solution, it is always accepted
    - Otherwise, the algorithm accepts a worse move with a probability less than 1 that decreases over time
    - Terminate when a min temperature is reached or solution does not change in value
- Differently from hill climbing, simulated annealing allows bad moves

Simulated Annealing: Accepting Bad Moves

- How should we set the probability of accepting bad moves?
  - **High** probability in the beginning, similar to random search (explore)
  - **Lower** as time passes, similar to hill climbing (exploit)
- In metallurgy, **annealing** is the process used to temper or harden metals by heating them to a **high** temperature
- Then, **gradually** cooling them until they reach a low-energy state
- Simulated annealing uses a similar principle to accept worse moves
  - The algorithm accepts worse moves with some probability less than 1
  - The probability **decreases exponentially** the worse the move is
  - The probability decreases as the **temperature goes down**

Probability Function: Quality

- We model the probability using a thermodynamics-inspired function:

$$e^{\Delta E / T}$$

- 'The probability **decreases exponentially** the worse the move is'. This is determined by the numerator in the exponential:

$$\Delta E = E(x_{\text{new}}) - E(x_{\text{current}}) \leq 0,$$

  where $E(x_{\text{new}})$ is the energy (quality) of the new random state and $E(x_{\text{current}})$ is the energy (quality) of the current state

- **Case 1: Very bad move.** If $E(x_{\text{new}}) \ll E(x_{\text{current}})$, then $e^{\Delta E / T} \approx 0$, i.e., a lower probability to accept the move

- **Case 2: Not too bad move.** If $|E(x_{\text{new}}) - E(x_{\text{current}})| \approx 0$, then $e^{\Delta E / T} > 0$, i.e., a higher probability to accept the move

Probability Function: Temperature

- ‘The probability decreases as the **temperature goes down**’. This is determined by the denominator $T > 0$
- At start, we set the temperature to a high value $T(0) = \mathcal{T} \gg 0$
- An update rule (**schedule**) is used to reduce it, for instance:

$$T(t+1) = \alpha T(t),$$

  where $T(t+1)$ and $T(t)$ represent the values of the temperature at the next iteration and at this iteration, respectively, and $\alpha \approx 1$
- If the temperature decreases slowly enough, simulated annealing will **find the global maximum with probability 1**
- This is a property of the Boltzman distribution $e^{\Delta E/T}$, i.e., the probability is **concentrated on the global maxima**

Probability Function: Temperature

- We model the probability using a thermodynamics-inspired function:

$$e^{\Delta E/T}$$

- How does the temperature affect the probability?
- **Case 1: High temperature.** If $T \gg 0$, then $e^{\Delta E/T} \approx 1$, i.e., a higher probability to accept the move
- **Case 2: Low temperature.** If $T \approx 0$, then $e^{\Delta E/T} \approx 0$, i.e., a lower probability to accept the move

Simulated Annealing

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state
  current ← *problem*.INITIAL
  for $t = 1$ **to** ∞ **do**
    $T \leftarrow$ *schedule*(t)
    if $T = 0$ **then return** *current*
    next ← a randomly selected successor of *current*
    $\Delta E \leftarrow$ VALUE(*current*) – VALUE(*next*)
    if $\Delta E > 0$ **then** *current* ← *next*
    else *current* ← *next* only with probability $e^{\Delta E/T}$

Performance of Simulated Annealing

- Let us evaluate the performance of simulated annealing
  - **Completeness**: simulated annealing is not complete, as it depends on the problem formulation and design of the algorithm
  - **Optimality**: simulated annealing is not optimal, as it depends on the termination criteria and the schedule
  - **Time complexity**: it depends on the schedule and the termination criterion
  - **Space complexity**: depends on how the design variable is represented in the algorithm

Simulated Annealing

- Pros:
  - Can usually find a good (near optimal) solution in a reasonable amount of time
  - It avoids getting stuck in poor local maxima and plateau by combining exploration and exploitation
- Cons:
  - Not guaranteed to be complete, nor optimal
  - Time/space complexity is problem- and representation-dependent

Outline

1. Recap: Informed Search and Optimisation
2. Hill Climbing: Variants
3. Simulated Annealing
4. Constraint Handling

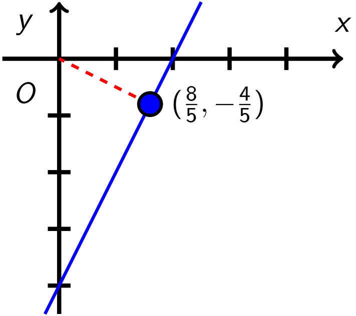
Applying Optimisation Algorithms

- To apply hill climbing and simulated annealing we need to specify:
- **Optimisation problem formulation**
  - Design variable and search space
  - Objective function
  - Constraints
- **Algorithm operators**
  - Representation
  - Initialisation procedure
  - Neighbourhood operator
- **Strategy to deal with constraints**
  1. By carefully designing algorithm operators
  2. By modifying the objective function

Optimisation: An Example

Activity. Find the closest point to the origin that lies on the line $y = 2x - 4$. In pairs or small groups, formulate the optimisation problem.

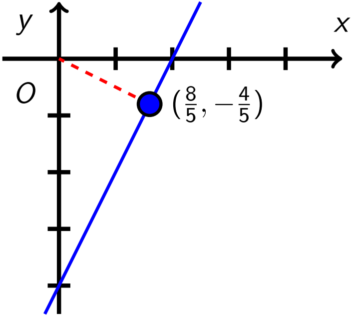
- **Design variable**
  $\bar{x} \in \mathbb{R}^2$, $\bar{x} = [x, y]^\top$
- **Search space**
  $\mathbb{R}^2$
- **Objective function**
  $\min f(\bar{x}) = \sqrt{(x - x_0)^2 + (y - y_0)^2}$
- **Constraints**
  $h(\bar{x}) = 0$, where
  $$h(\bar{x}) = \begin{cases} 0, & \text{if } y = 2x - 4, \\ 1, & \text{otherwise.} \end{cases}$$



Optimisation: An Example

Activity. Find the closest point to the origin that lies on the line $y = 2x - 4$. In pairs or small groups, formulate the optimisation problem without any explicit constraints.

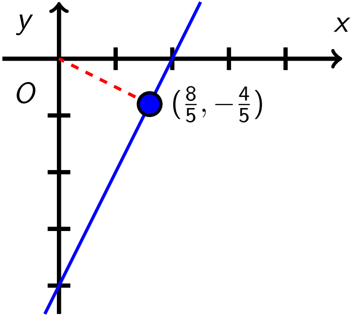
- **Design variable**
  $\bar{x} \in \{x, y : y = 2x - 4\}$, $\bar{x} = [x, y]^\top$
- **Search space**
  $\{x, y : y = 2x - 4\}$
- **Objective function**
  $\min f(\bar{x}) = \sqrt{(x - x_0)^2 + (y - y_0)^2}$
- **Constraints**
  No explicit constraints



Optimisation: An Example

Example. Find the closest point to the origin that lies on the line $y = 2x - 4$. Let us determine the algorithm operators.

- **Representation**
  Real vector of size 2
- **Initialisation procedure**
  Initialise with two real values picked uniformly at random
- **Neighbourhood operator**
  Add or subtract 0.1 to either variable
- Hill climbing and simulated annealing may return an infeasible solution



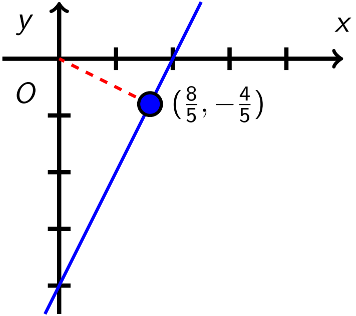
Constraint Handling: Algorithm Operators

- **Representation**
  Real vector of size 2

- **Initialisation procedure**
  Initialise with two real values
  $\bar{x} \in \{x, y : y = 2x - 4\}$
  picked uniformly at random

- **Neighbourhood operator**
  Add or subtract 0.1 to x
  and find corresponding y

- **Constraint strategy**
  Operators above

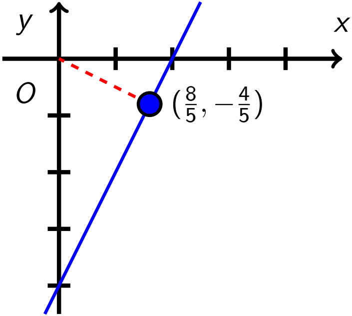


Constraint Handling: Algorithm Operators

- Pros:
  - **Infeasible** candidate solutions will not be generated, facilitating the search for optimal solutions
- Cons:
  - May be difficult to design, problem-dependent
  - Can restrict the search space too much, making it harder to find the optimal solution

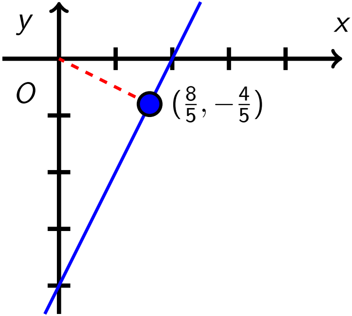
Constraint Handling: Objective Function

- **Design variable**
  $\bar{x} \in \mathbb{R}^2$, $\bar{x} = [x, y]^\top$
- **Search space**
  $\mathbb{R}^2$
- **Objective function**
  $\min f(\bar{x}) = \sqrt{(x - x_0)^2 + (y - y_0)^2}$
- **Constraints**
  $h(\bar{x}) = 0$, where
  $$h(\bar{x}) = \begin{cases} 0, & \text{if } y = 2x - 4, \\ 1, & \text{otherwise.} \end{cases}$$



Constraint Handling: Objective Function

- **Representation**
  Real vector of size 2
- **Initialisation procedure**
  Initialise with two real values picked uniformly at random
- **Neighbourhood operator**
  Add or subtract 0.1 to either variable
- How do we avoid infeasible solutions?



Approach 1: Death Penalty

- Given the canonical formulation of an optimisation problem:

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

- We avoid infeasible solutions by modifying the objective function

$$\min \quad f(x) + Q(x),$$

where the penalty term $Q(x)$ is defined as:

$$Q(x) \begin{cases} 0, & \text{if } x \text{ is feasible,} \\ C, & \text{otherwise.} \end{cases}$$

- The term C is a large positive constant, ensuring infeasible solutions are worse than feasible solutions

Approach 1: Death Penalty

- Given the canonical formulation of an optimisation problem:

$$\begin{aligned} \min \quad & f'(x), \\ \text{s.t.} \quad & g_i(x) \leq 0, \quad i = 1, \ldots, m, \\ & h_j(x) = 0, \quad j = 1, \ldots, n. \end{aligned}$$

- We can also use the following formulation:

$$f'(x) = \begin{cases} f(x), & \text{if } x \text{ is feasible}, \\ C, & \text{otherwise}. \end{cases}$$

- The term C is a large positive constant, like before
- Weakness: all infeasible solutions have the same penalty
- Algorithms will struggle to find feasible solutions in regions of the state space dominated by infeasible solutions

Approach 2: Levels of Infeasibility

- Given the canonical formulation of an optimisation problem:

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

- The idea is to distinguish the objective value of infeasible solutions to help the algorithm find a feasible solution

$$\begin{aligned} Q(x) = & C g_1(x) + C g_2(x) + \cdots + C g_m(x) \\ & + C|h_1(x)| + C|h_2(x)| + \cdots + C|h_n(x)|. \end{aligned}$$

- The objective function itself will guide the algorithm towards feasible solutions (and near-optimal solutions, too!)

Approach 2: Levels of Infeasibility

- Given the canonical formulation of an optimisation problem:

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

- We should only add the penalties corresponding to violated constraints

$$\begin{aligned} Q(x) = & v_{g_1} C g_1(x) + v_{g_2} C g_2(x) + \cdots + v_{g_m} C g_m(x) \\ & + v_{h_1} C |h_1(x)| + v_{h_2} C |h_2(x)| + \cdots + v_{h_n} C |h_n(x)|, \end{aligned}$$

where $v_{g_i} = 1$ if g_i is violated and $v_{g_i} = 0$ otherwise,
and $v_{h_i} = 1$ if h_i is violated and $v_{h_i} = 0$ otherwise

Approach 2: Levels of Infeasibility

- Given the canonical formulation of an optimisation problem:

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

- We should only add the penalties corresponding to violated constraints

$$\begin{aligned} Q(x) = & v_{g_1} C g_1(x)^2 + v_{g_2} C g_2(x)^2 + \cdots + v_{g_m} C g_m(x)^2 \\ & + v_{h_1} C h_1(x)^2 + v_{h_2} C h_2(x)^2 + \cdots + v_{h_n} C h_n(x)^2. \end{aligned}$$

- **Squared values** make the distinction between objective values even larger, distinguishing different infeasible solutions more

Constraint Handling: Objective Function

- Pros:
  - Usually easier to design
- Cons:
  - Algorithms have to search for feasible solutions to design

Final Remarks: Completeness

- If we use a strategy that never enables infeasible solutions to be generated, then hill climbing and simulated annealing are complete
- Otherwise:
  - Hill climbing is not complete if the objective function has local optima (a local optimum may be infeasible)
  - Simulated annealing is not guaranteed to find a feasible solution within a reasonable amount of time; it might terminate before finding a feasible solution

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 4.1.2 "Simulated Annealing".

Aims of the Session

You should now be able to:

- Understand the weaknesses in Hill Climbing and how to mitigate them
- Explain the steps involved in Simulated Annealing
- Deal with constraints in optimisation problems

Thank you!