Artificial Intelligence I 2023/2024

Week 11 Tutorial and Additional Exercises

Simulated Annealing & Constraint Handling

School of Computer Science

29th April 2024

In this tutorial. . .

In this tutorial we will be covering

- Simulated Annealing.
- Constraint Handling.
- Examples of algorithm designs.

Exercise 1

Recall the algorithm for Simulated Annealing.

Algorithm 1: Simulated Annealing (maximisation).

```
Input: $T_0$: initial temperature, $T_f$: minimum temperature
Output: $\mathbf{x}$: final solution
1  $\mathbf{x} \leftarrow$ random initial solution;
2  $T \leftarrow T_0$;
3  repeat
4      $\mathbf{x}' \leftarrow$ random neighbour of $\mathbf{x}$;
5      if $quality(\mathbf{x}') \leq quality(\mathbf{x})$ then
6          with probability $e^{\frac{\Delta E}{T}}$ : $\mathbf{x} \leftarrow \mathbf{x}'$
7      else
8          $\mathbf{x} \leftarrow \mathbf{x}'$
9      $T \leftarrow schedule(T)$
10 until $T \leq T_f$ or $\mathbf{x}$ "stops changing";
11 return $\mathbf{x}$
```

where

- $\Delta E = quality(\mathbf{x}') - quality(\mathbf{x})$.
- $T > 0$ is the temperature.
- $schedule(T)$ is a non-increasing function that updates the temperature.
- $e \approx 2.71828\ldots$ is a constant.

How would you change algorithm 1 to do minimisation?

Exercise 2

We revisit the knapsack problem: Given N items with weights $w_1, \ldots, w_N$ and profits $p_1, \ldots, p_N$, find the set of items to be loaded into a knapsack so as to maximise the total profit of loaded items. The total weight of loaded items should be at most W. More formally:

$$\begin{array}{rl} \text{maximise} & f(\mathbf{v}) = \sum_{i=1}^{N} v_i p_i \\ \text{subject to} & g(\mathbf{v}) = \sum_{i=1}^{N} v_i w_i - W \leq 0 \end{array}$$

where $\mathbf{v} = (v_1, \ldots, v_N)$; and $\forall i \in \{1, \cdots, N\}$, v_i equals 1 if item i is loaded and 0 if not.

Exercise 2 (continued)

- We will apply the hill-climbing algorithm to find heuristic solutions to the knapsack problem.
- We start with some initial solution, then for each iteration, we generate neighbour solutions and pick the best of them.
- To use hill-climbing we need an initial solution and a method to generate neighbour solutions from a current solution.
- Exercise: Given the problem parameters, design two algorithms: One that will generate an initial solution and one that will generate neighbour solutions, given a current solution.
- You must ensure that any generated solution is feasible, so you might want to include a strategy to deal with the constraint!

Exercise 3

Consider the knapsack problem with $W = 10$ and parameters:

i	1	2	3	4	5
w_i	3	5	9	4	1
p_i	10	20	40	15	3

If the current solution is $\mathbf{v} = (0, 1, 0, 1, 0)$, find the next two neighbours using the same algorithm (repeated to the right). Write all neighbours we visit in the process, along with their weights and profits.

Algorithm 2: Neighbourhood operator (repeated).

```
Input: $\mathbf{w} = (w_1, \dots, w_N)$: weights,
       $\mathbf{p} = (p_1, \dots, p_N)$: profits, $W$:
       maximum weight, $\mathbf{v}$: current
       solution
Output: $l_u$: list of neighbour solutions
1  $l_u \leftarrow []$;
2  for $i = 1, \dots, N$ do
3      $\mathbf{u} \leftarrow \mathbf{v}$;
4      $\mathbf{u}[i] \leftarrow 1 - \mathbf{v}[i]$;
5      $j \leftarrow 0$;
6      while $\mathbf{u}^T\mathbf{w} - W > 0$ do
7          $j \leftarrow j + 1$ (but skip the value
             of $i$);
8          $\mathbf{u}[j] \leftarrow 0$
9      if $\mathbf{u}^T\mathbf{w} - W \leq 0$ then
10         $l_u.add(\mathbf{u})$
11 return $l_u$
```

Exercise 4

Recall the knapsack problem formulation from Exercise 2. Design a strategy to deal with the constraint by modifying the objective function of this problem.

PS: Note that this is a maximisation problem, rather than a minimisation problem. So, you will need to change the general format of the strategy to deal with the constraint seen in the lecture.

Up next...

Advanced Material

Advanced Exercise 1

Implementing algorithms from the field of artificial intelligence has several similarities to implementing algorithms from other fields. However, one key difference is that several artificial intelligence algorithms involve probabilistic decisions. An example is Simulated Annealing, which makes use of the probability of replacing the current solution by a neighbour of equal or worse quality.

This question is intended as an example of how to make probabilistic decisions when implementing such algorithms. It is a challenge question, because you may never have implemented algorithms that involve probabilistic decisions before. However, once you understand how to make probabilistic decisions in the implementation, the same coding strategy can be used when implementing other algorithms that involve probabilistic decisions.

Advanced Exercise 1 (continued)

Consider that you would like to implement Simulated Annealing in Java. You have a pseudo random number generator able to give you pseudo random numbers in the interval $[0,1)$.[1]

```
class Random:
public double nextDouble()
```

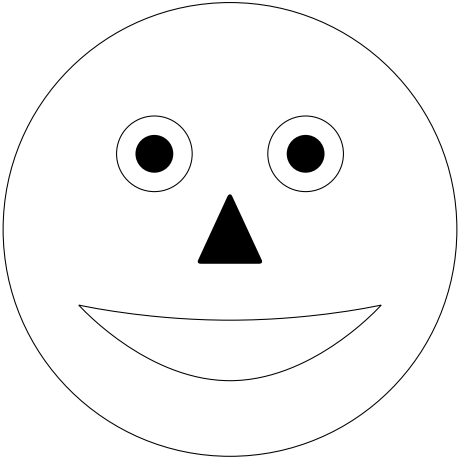
Write a piece of Java code that implements line 6 in algorithm 1.

[1]The notation $[0,1)$ means that 0 is included in the interval, but 1 is not.

Q&A

Any questions?

Until the next time...



Thank you for your attention!