Artificial Intelligence

Tutorial week 5 – Metaheuristics

COMP9414

1 Theoretical Background

Metaheuristics are algorithms designed to find approximate solutions for optimisation problems. They are general-purpose techniques that can be applied to a wide range of problems where finding the optimal solution is computationally expensive or infeasible.

Metaheuristics are often inspired by natural or social phenomena and mimic processes such as evolution, swarm behaviour, or physical annealing. They typically involve iterative improvement processes that gradually refine the candidate solution (or a population of candidate solutions) over multiple iterations.

Particularly, the simulated annealing algorithm is inspired by the annealing process in metallurgy. The algorithm starts with an initial solution and iteratively explores the solution space by allowing moves to worse solutions based on the temperature parameter. As the temperature decreases, the algorithm becomes more selective and focuses on exploiting the local search space. Algorithm 1 shows the simulated annealing optimisation method.

Algorithm 1 Simulated annealing optimisation method.

```
Require: Input($T_0, \alpha, N, T_f$)
 1: $T \leftarrow T_0$
 2: $S_{act} \leftarrow$ generate initial solution
 3: while $T \geq T_f$ do
 4:   for cont $\leftarrow$ 1 TO $N(T)$ do
 5:     $S_{cond} \leftarrow$ Neighbour solution [from ($S_{act}$)]
 6:     $\delta \leftarrow f(S_{cond}) - f(S_{act})$
 7:     if rand(0,1) $< e^{-\delta/T}$ or $\delta < 0$ then
 8:       $S_{act} \leftarrow S_{cond}$
 9:     end if
10:   end for
11:   $T \leftarrow \alpha(T)$
12: end while
13: return Best $S_{act}$ visited
```

On the other hand, tabu search a memory-based approach is used to avoid getting stuck in local optima. They maintain a short-term memory (tabu list) to store recently visited solutions and prevent revisiting them in the search process. Algorithm 2 describes the steps in the tabu search optimisation method.

Algorithm 2 Tabu search optimisation method.

```
1: $s_0 \leftarrow$ generate initial solution
2: $s_{best} \leftarrow s_0$
3: tabuList $\leftarrow \{s_0\}$
4: repeat
5:     $\{s_1, s_2, ..., s_n\} \leftarrow$ generate neighbourhood from $(s_{best})$
6:     $s_{candidate} \leftarrow s_1$
7:     for i $\leftarrow$ 2 TO $n$ do
8:         $\delta \leftarrow f(s_i) - f(s_{candidate})$
9:         if $s_i$ is not in tabuList and $\delta < 0$ then
10:            $s_{candidate} \leftarrow s_i$
11:        end if
12:    end for
13:    $s_{best} \leftarrow s_{candidate}$
14:    Add $s_{candidate}$ to tabuList
15: until a termination criterion is satisfied
16: return $s_{best}$
```

2 Setup

(a) Using Python, create the fitness function $y = x^2$. We will use this fitness function to evaluate the solutions.

(b) In a range of -10 to +10 plot the fitness function and show the optimal value. At this point, consider the optimum a known value.

(c) Implement the simulated annealing optimisation method shown in Algorithm 1 as a function, taking into account the following:.

- For the initial solution, use a random number approximately between -10 and 10 from a standard normal distribution (i.e., $\mu = 0, \sigma = 1$).
- For the neighbour solution, use the current solution + a random number from a standard normal distribution.

(d) Implement the tabu search optimisation method shown in Algorithm 2 as a function, taking into account the following:

- For the initial solution, use a random number approximately between -10 and 10 from a standard normal distribution (i.e., $\mu = 0, \sigma = 1$).
- For the Tabu List use a numpy array.

- Each neighbour is computed based on the current best solution as: best solution + a random number from a standard normal distribution.

3 Experiments

(a) Run the simulated annealing method using the following parameters: $T_0 = 1 \times 10^1, \alpha = 0.9, N = 100, T_f = 1 \times 10^{-5}$, and seed the random numbers to 999 using `np.random.seed(999)`. Print the returned solution and its fitness value.

(b) Run the tabu search method using the neighbourhood size = 50 for each iteration, termination criterion as finding a solution with fitness lower than 1×10^{-10}, and seed the random numbers to 55 using `np.random.seed(55)`. Print the returned solution and its fitness value.

(c) For simulated annealing method:

- Modify the algorithm to store in an array the last current solution for each temperature cycle and return this array.
- Plot the solution scores for each temperature cycle.
- Similarly, store the temperatures used in an array, return it, and plot the temperature decrease.
- Perform variations in the following parameters and observe the results:
  - Seed value for random numbers.
  - Number of iterations for each temperature N.
  - Temperature descent α.
  - Initial T_0 and final T_f temperature.

(d) **Challenge:** The eggholder function is a test function used in optimisation. It allows testing optimiser algorithms as it has multiple local minima. The function is defined in a search domain $-512 \leq x, y \leq 512$ as follows:

$$f(x,y) = -(y+47)\sin\sqrt{\left|\frac{x}{2} + (y+47)\right|} - x\sin\sqrt{|x-(y+47)|} \quad (1)$$

- Create a fitness function that receives x and y and returns the eggholder value function.
- Plot the surface function.
- Adapt the previous simulated annealing code to work with the new fitness function (i.e., accepting x, y inputs).

- Adapt how the initial and the neighbour solutions are generated (consider the step-size for the neighbour solution).
- Evaluate only valid neighbour solutions using a rejection strategy, i.e., immediately discard solutions out of the variable limits.
- Run the simulated annealing algorithm to find the best possible solution. Modify as many parameters as needed.
- **Hint:** the optimal minimum is $f(512, 404.2319) = -959.6406627106155$.