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  $s_{best}$
```

Population-based methods operate on a population of candidate solutions rather than a single solution. The population is typically initialized randomly or using heuristic techniques. Multiple solutions are evaluated simultaneously, allowing for the exploration of the search space more efficiently. Algorithm 3 describes the overall steps for a genetic algorithm optimisation method.

Algorithm 3 Genetic algorithm optimisation method.

```
1: $t \leftarrow 0$
2: Initialise $P(t)$ ▶ initial population
3: Evaluate $P(t)$
4: repeat
5:     Generate offspring $C(t)$ from $P(t)$ ▶ using crossover and mutation
6:     Evaluate $C(t)$
7:     Select $P(t+1)$ from $P(t) \cup C(t)$
8:     $t \leftarrow t + 1$
9: until a termination criterion is satisfied
10: return  Best individual found from $P$
```

2 Memoryless and memory-based methods

2.1 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.

2.2 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:

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

- 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 $f(512, 404.2319) = -959.6406627106155$.

3 Population-based methods

In this section, we will develop a genetic algorithm to evolve towards the string "HELLO WORLD" from an initial random population.

3.1 Setup

(a) Start creating a population of random strings (using only capital letters and spaces). Initially, the population will contain 100 individuals. Each individual within the population will be a string of length equal to the length of "HELLO WORLD".

(b) Create a fitness function to evaluate each individual solution. The fitness function will return +1 for each character that is in the right position in comparison to "HELLO WORLD". For instance, the individual "HAYUHATOSLA" will have a fitness value equal to 3 as letters H, O, and L are in the right position.

(c) Compute the fitness values for the entire population. If the best individual is equal to "HELLO WORLD," you have reached the target, and the algorithm can stop.

(d) Create a method for crossover. The method should receive two parents (previously selected with a stochastic sampling, such as a roulette wheel) and return two children that are new individuals from the parents using a randomly determined crossover point.

(e) Apply a mutation operator to both children using a mutation rate equals to 0.01 to each character within the individual. If the character is selected for mutation, then it is replaced by any random capital letter or space.

(f) Iterate the genetic algorithm for 1000 generations. The population should remain the same size n during the generations. In case n is an odd number, the population size will be $n - 1$ at the end, as we are creating two children from each couple of parents selected.

(g) Print the final individual selected and its fitness value.

3.2 Experiments

(a) Modify the algorithm parameters such as population size, mutation rate, and generations. Observe the results.

(b) Allow the algorithm different sampling methods and/or allow the algorithm to increase the population size by including intergenerational solutions.