Nature-inspired Search and Optimisation Continuous Assessment 1 Report

Zhangda Xu

2088192

School of Computer Science

I. Introduction

1. Genetic Algorithm

Genetic algorithm is a metaheuristic for search inspired by the theory of natural evolution. It is considered as an adaptive strategy for searching the global optimisation, where solutions can be represented by vectors with permutation. The whole paradigm of genetic algorithms inherits from the modern synthesis of biology, focusing on the evolution on a scale of population genetics. By producing offsprings, the population can evolve over reproduction and a proportion will be selected. The evolution of solutions are carried out by the changes in the intrinsic characteristics of populations over generations. Eventually, the genetic variations that enhance the survivability within the population become more common and dominate the population of later generations.

For individuals of a population, the contribution to their population is represented as genotype, which is expressed by their phenotype to the environment in the form of offspring. This representation of solutions in the population enables variation operators to explore the evolutionary search space of the population by mating and mutating. Then individuals with better fitness will be selected and reproduced to create the next generation. The performance of genetic algorithm relies on the strategy of balancing exploration and exploitation to find global optimum.

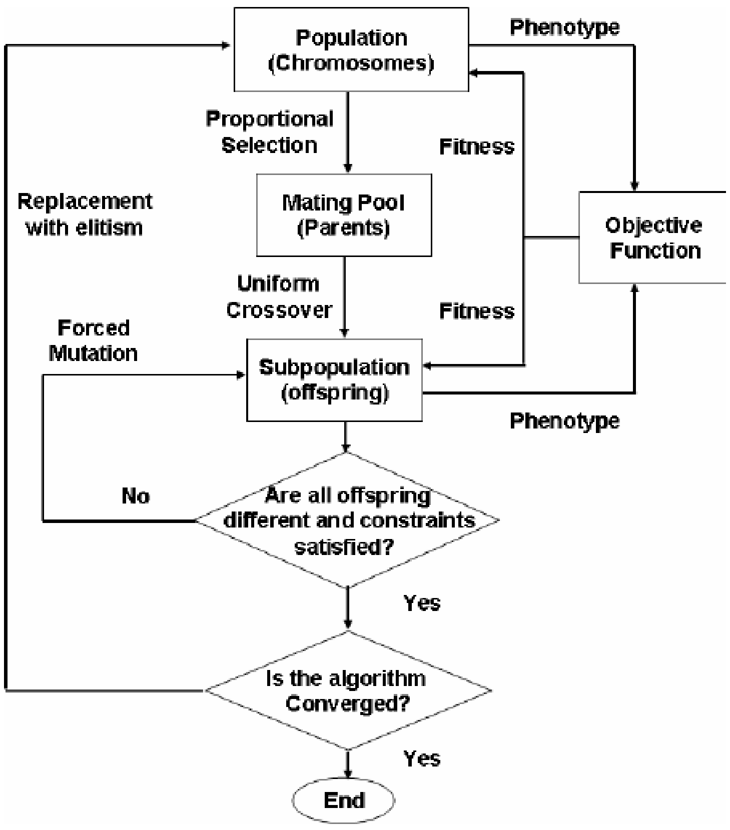


Fig 1. Flowchart of genetic algorithm

For the Travelling Salesman Problem (TSP), the goal of genetic algorithm is to find the shortest route between cities minimising the total distance (pseudo-Euclidean). In this context, a single route constrained as above is firstly represented as individuals with the cities as their gene. Each of them are evaluated over the distance as their fitness. The population is the collection of individuals of possible routes, among which the selected solutions for creating the next generation are called parents. New solutions are then reproduced as offsprings by crossover. For better exploration of the solution space, a mutation process is applied to offsprings by random permutation. After mating and mutation, offsprings and the best individuals of parents will be used to reproduce the next generation.

Representation: Different from other problems involving bit string or real numbers, each element in one solution to TSP is unique with each other. In this experiment, the genotype is set to be the index of cities, otherwise repeated visits could happen during evolution. The fitness of each individual is evaluated by the distance.

Reproduction: A fitness proportional selection is conducted to choose the parents of offsprings. Basically, it ranks the population by their fitness value and select individuals with higher fitness with higher probability.

Variation operators: For creating the next generation, we apply variation operators including crossover and mutation. The crossover method is special for TSP because only children with non-repetitive steps are required. With a certain probability, we keep a subset of parent 1 and fill up other locations with the non-repetitive elements in parent 2 in the same order. A mutation is operated one each offsprings with a mutation rate, where a pair of cities is swapped.

Selection: The last step is to create a new generation with new offsprings. The top individuals in the population are kept as elites, which helps to preserve the best solutions. The new population is then filled by randomly selecting the offsprings.

Algorithm below provides a pseudocode of genetic algorithm for solving TSP:

Pseudocode for Genetic Algorithm

```
Input: $Population_{size}, Problem_{size}, P_{crossover}, P_{mutation}$
Output: $S_{best}$
Population $\leftarrow$ InitialisePopulation($Population_{size}, Problem_{size}$)
EvaluatePopulation(Population)
$S_{best}$ $\leftarrow$ GetBestSolution(Population)
While ($\neg$StopCriteria())
    Parents $\leftarrow$ MatingPool(Population, $Population_{size}$)
    Children $\leftarrow \emptyset$
    For ($Parent_1, Parent_2 \in$ Parents)
        $Child_1, Child_2 \leftarrow$ Crossover($Parent_1, Parent_2, P_{crossover}$)
        Children $\leftarrow$ Mutate($Child_1, P_{mutation}$)
        Children $\leftarrow$ Mutate($Child_2, P_{mutation}$)
    End
    EvaluatePopulation(Children)
    $S_{best}$ $\leftarrow$ GetBestSolution(Children)
    Population $\leftarrow$ Replace(Population, Children)
End
Return $S_{best}$
```

2. Simulated Annealing

Simulated annealing is a heuristic search algorithm for approximating the global optimum. It was first proposed by Kirkpatrick in 1983. The inspiration of this algorithm comes from the real annealing procedure in practical engineering. It is suggested that to achieve better structural and material properties, the substance is best processed by first melting it done then slowly lowering the temperature at a suitable level in the vicinity of freezing point. In the optimisation, this algorithm can escape from local optimal solutions.

Simulated annealing essentially applies a stochastic local search with random non-improving steps. After initiation, the algorithm first searches neighbour solutions and compare their fitness value with the current solution. The better solutions will be always accepted. We also accept worse solutions with a probability $P = exp(\frac{e-e_{new}}{T})$, which is parameterised by the temperature T. Generally, the temperature starts high and slowly decreases as the process goes. Heuristically, the probability of accepting worse solutions will decrease when it is closer to the optimum.

In the context of Travelling Salesman Problem (TSP), simulated annealing is efficient to explore the vast search space without being easily trapped in the local optimum. The neighbour solutions are generated by 2-opt algorithm in a stochastic approach. The accepted worse route solutions introduce random steps at the early stage and eventually become a local hill-climbing search approaching the final state.

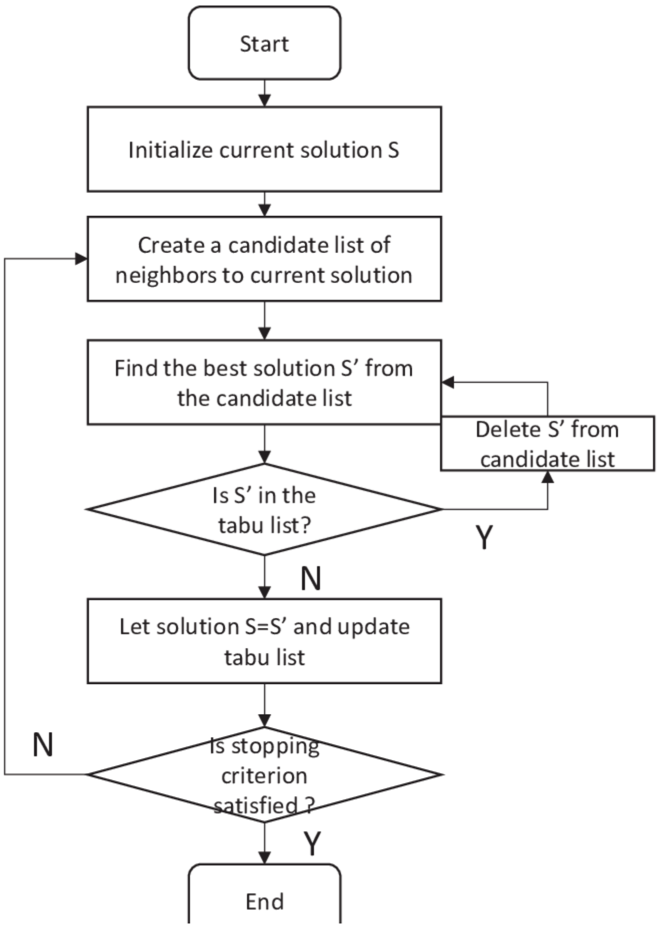
Algorithm below provides a pseudocode of simulated annealing for solving TSP:

Pseudocode for Simulated Annealing

```
Input: $Problem_{size}, iterations_{max}, temp_{max}$
Output: $S_{best}$
$S_{current} \leftarrow$ InitialiseSolution($Problem_{size}$)
$S_{best} \leftarrow S_{current}$
For ($i = 1$ to $iterations_{max}$)
    $S_i \leftarrow$ TwoOpt($S_{current}$)
    $temp_{current} \leftarrow$ CalculateTemperature($i, temp_{max}$)
    If (Cost($S_i$) $\leq$ Cost($S_{current}$)
        $S_{current} \leftarrow S_i$
    Elseif (Exp((Cost($S_{current}$) - Cost($S_i$))/$temp_{curr}$) $>$ Rand())
        $S_{current} \leftarrow S_i$
    End
End
Return $S_{best}$
```

3. Tabu Search

Tabu search is a widely applied search optimisation algorithm with an embedded heuristic. The main idea is to keep a memory structure called tabu list to constrain the local search process from visiting previous solutions. Other long-term memories can introduce more bias to avoid local optimum.



Start
Initialize current solution S
Create a candidate list of neighbors to current solution
Find the best solution S' from the candidate list
Delete S' from candidate list
Is S' in the tabu list?
Y
N
Let solution S=S' and update tabu list
N
Is stopping criterion satisfied ?
Y
End

Fig 2. Flowchart for Tabu search

For the Travelling Salesman Problem (TSP), Tabu search is applied to find the best route with shortest distance when traversing cities. A simple tabu list with short memory is maintained to exclude recently visited solutions in the neighbourhood. The neighbour solutions are generated by stochastic 2-opt local search. Heuristically, Tabu search guides the optimisation process away from local optima.

Algorithm below provides a pseudocode of Tabu search for solving TSP:

Pseudocode for Tabu Search

Input: $TabuList_{size}$

```
**Output:** $S_{best}$
$S_{best} \leftarrow$ InitialiseSolution()
TabuList $\leftarrow \emptyset$
**While** ($\neg$ StopCriteria())
    CandidateList $\leftarrow \emptyset$
    **For** ($S_{candidate} \in$ Neighbourhood($S_{best}$))
        **If** ($\neg$ ContainsAnyFeatures($S_{candidate}$, TabuList))
            CandidateList $\leftarrow S_{candidate}$
        **End**
    **End**
    $S_{current} \leftarrow$ Two-Opt(CandidateList)
    **If** (Cost($S_{current}$) $\leq$ Cost($S_{best}$))
        $S_{best} \leftarrow S_{current}$
        TabuList $\leftarrow$ ()
        **While** TabuList $> TabuList_{size}$
            DeleteFirst(TabuList)
        **End**
    **End**
**End**
**Return** $S_{best}$
```

II. Parameters

1. Genetic Algorithm

Population size: The size of population in each generation provides the search space for TSP. The size starts with 100 individuals per generation. By increasing it, an performance improvement is expected. However, population exceeding 100 individuals costs more time to converge.

Crossover rate: After selection of parents, some of them will breed offsprings. The default rate for the ordered crossover on individuals is 0.98. After research, this is the natural value for crossover.

Mutation rate: A small operator is applied to mutate the offsprings. By default, the mutation rate is set as 1/48, which means approximately one change for each solution. However in TSP the mutation appears in pairs. The parameter is then changed to 1/24 for better heuristics and performance. Mutation rate should be small to generate better solution.

Elite size: Elite is the top individuals kept to the next generation. It introduces aspects of local search in evolution. The elite size should be small to avoid local optimum. No significant improvement is reached after tuning the size.

Tuning	**Default**	**Trial**	**Best**
pop_size	100	300, 1000	100
p_crossover	0.98	0.98, 1	0.98
p_mutation	1/48	0, 1/24	1/24
elite_size	20	0, 10, 50	20

2. Tuning Simulated Annealing

Initial temperature: Tuning the simulated annealing algorithm is to change the properties of the *temperature* parameter. Since the probability of adopting a worse solution should , by principle, start high and decrease during iteration, the initial temperature is set to be a big value, which declines at every step. Heuristically, setting a low temperature means the algorithm decays rapidly to a local search after few steps, which is not desirable for solving TSP. In the other hand, starting at a higher temperature could unnecessarily prolong the convergence without improvement.

The experiment is aimed to find the optimal parameter of initial temperature in simulated annealing. The decline function used in the experiment is $T = 0.98^k . initial$, a changed version of the function proposed by Kirptrick. This function will not be compared to other variations, because the change rate of temperature is too fine to tune in this scope. Experimented value starts from 100 to 100,000. Every group is controlled by the times of iteration as well to compare their performance (shortest distance) on different settings. Distances in the table are mean values of the results recorded in 30 individual runs. The result is listed below:

iteration ↓ /temp →	100	1000	10000	100000
100	32555	34659	45545	49677
500	18108	18520	20169	23041
1000	14549	14245	14896	15752
2000	12248	12127	12401	12643
3000	11674	11595	11574	11853

It is shown that increasing temperature leads to no significant improvement on performance with over 1000 iterations. The change in each column suggests that higher initial temperature needs more iterations to converge. With to following experiment, the temperature is set to be 10000 at the start for a reliable performance.

3. Tuning Tabu Search

Tabu size: This report analyses a simple implement of Tabu search with a short term memory of recently visited neighbours. The Tabu list records most recent changes between solutions caused by 2-opt. More specific, it enlists the index of two nodes involved in the transformation. The default size of Tabu list is 15. By increasing the size of the list, it is expected to yield better convergence. The result is listed below:

iteration ↓ /Tabu size →	15	30	50
300	22402	22699	22428
1000	14721	14601	14577
3000	11545	11820	11768

It is shown that by increasing the size of the Tabu list, no significant improvement is achieved. The convergence rate is similar between different sizes as well. Nevertheless, this algorithm outperforms the local search by simply remembering recently visited solutions. In this experiment, the size of Tabu list is set to be 15.

III. Results

After setting the right parameters, the three algorithms are evaluated by 30 individual runs with 3000 iterations. The mean value and standard deviation of the results yielded by 3 algorithms are listed in the table, respectively.

Algorithm	Mean value	Standard deviation
Genetic algorithm	59507.83	1969.10
Simulated annealing	11680.77	302.64
Tabu search	11672.80	354.73

To analyse the result it statistically, we apply Wilcoxon signed-rank test to results from 3 algorithms. It is a paired difference test which compares repeated measurements on a single sample to assess whether their population mean ranks differ. It is assumed that data are paired between groups but not necessarily follow normal distributions.

The null hypothesis here is that results from the simulated annealing and Tabu search algorithms come from the same distribution. This returns a p-value = 0.893, which is larger than the 0.05 significance level. It fails to reject the null hypothesis, and states that the results from two algorithms are from the same distribution.

Pairwise comparisons to the results from genetic algorithm reject the null hypothesis. It is possibly because the algorithm does not converge well in the limited generations and is often trapped in the local minimum. Another reflection is that TSP does not fit perfectly the scenario of genetic algorithm as demonstrated in the introduction. A different representation of solutions could lead to improved results.