K. N. Toosi University of Technology

Using Genetic Algorithm to Solve TSP Problem

Course:

Graph Theory and Algorithms

Instructor:

Dr. Farnaz Sheikhi

Group:

Hasti Khajeh
Negin Mohammadi
Tanaz Ghahremani
Sina Rostami

Spring 2023

1. Genetic Algorithms

In these algorithms, the search space of a problem is represented as a collection of individuals. The *path representation* is the most natural representation of a tour(individual), which is often referred to as chromosomes. Chromosomes are represented as a list of n cities. If city i is the j-th element of the list, city i is the j-th city to be visited. The purpose of using a genetic algorithm is to find the individual from the search space with the best “genetic material”. The quality of an individual is measured with an evaluation function. For TSP evaluation function calculates the sum of routes. The part of the search space to be examined is called the population.

First, the initial population is chosen, and the quality of this population is determined. Next, in every iteration parents are selected from the population. These parents produce children, which are added to the population. For all newly created individuals of the resulting population, a probability near to zero exists that they will “mutate”, i.e. that they will change their heriditary distinctions. After that, some individuals are removed from the population according to a selection criterion in order to reduce the population to its initial size. One iteration of the algorithm is referred to as a generation.

The operators which define the child production process and the mutation process are called the crossover operator and the mutation operator respectively. Mutation and crossover play different roles in the genetic algorithm. Mutation is needed to explore new states and helps the algorithm to avoid local optima. Crossover should increase the average quality of the population. By choosing adequate crossover and mutation operators, the probability that the genetic algorithm results in a near-optimal solution in a reasonable number of iterations is increased. For stopping AGA, the number of iterations needed is determined previously.

2. Crossover Methods

2.1. Partially-mapped crossover (PMX)

It passes on ordering and value information from the parent tours to the offspring tours. A portion of one parent’s string is mapped onto a portion of the other parent’s string and the remaining information is exchanged.

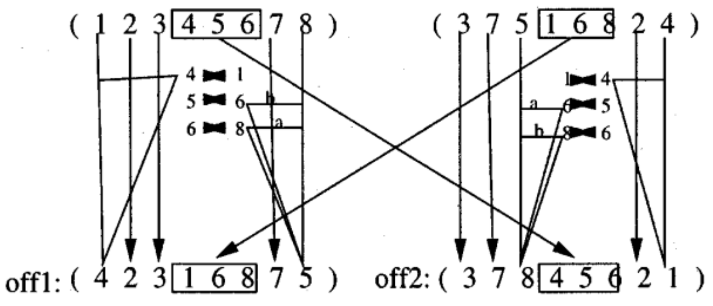


Figure 2. Partially-mapped crossover operator (PMX).

2.2. *Cycle crossover (CX)*

It attempts to create offspring from the parents where every position is occupied by a corresponding element from one of the parents.

Consider the parents in Figure 3, Now we choose the first element of the offspring equal to be either the first element of the first parent tour or the first element of the second parent tour. Hence, the first element of the offspring has to be a 1 or a 2. Suppose we choose it to be 1, analogously, we find that the fourth and the second element of the offspring also have to be selected from the first parent. The positions of the elements chosen up to now are said to be a cycle. Now consider the third element of the offspring. This element we may choose from any of the parents. Suppose that we select it to be from parent 2. This implies that the fifth, sixth, and seventh elements of the offspring also have to be chosen from the second parent, as they form another cycle.

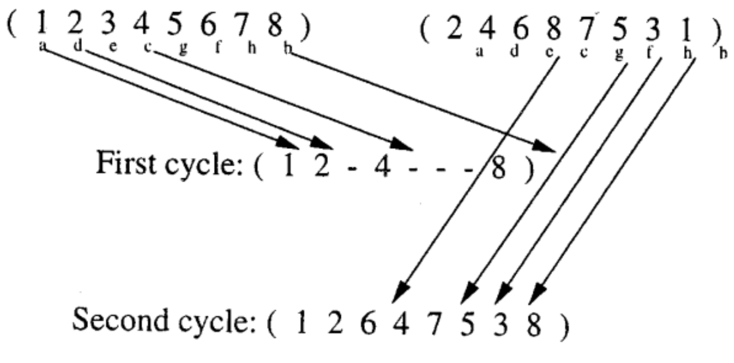

(1 2 3 4 5 6 7 8)
(2 4 6 8 7 5 3 1)
First cycle: (1 2 - 4 - - - 8)
Second cycle: (1 2 6 4 7 5 3 8)

Figure 3. Cycle crossover (CX).

2.3. *Order crossover (OX1)*

The OX1 exploits a property of the path representation, that the order of cities (not their positions) are important. It constructs an offspring by choosing a subtour of one parent and preserving the relative order of cities of the other parent.

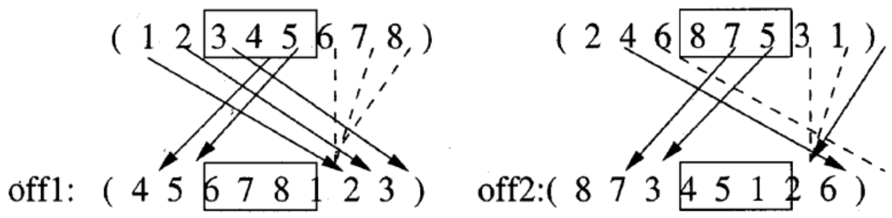


Figure 4. Order crossover (OX1).

2.4. *Order based crossover (OX2)*

It selects at random several positions in a parent tour, and the order of the cities in the selected positions of this parent is imposed on the other parent.

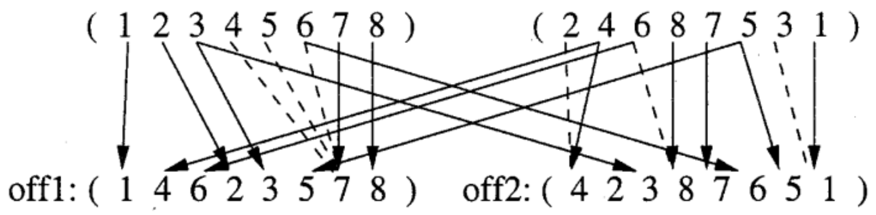


Figure 5. Position based crossover (POS).

2.5. *Genetic edge recombination crossover (ER)*

The genetic edge recombination operator works according to the following algorithm:

1. Choose the initial city from one of the two parent tours. (It can be chosen at random or according to criteria outlined in step 4). This is the “current city”.
2. Remove all occurrences of the “current city” from the left-hand side of the edge map. (These can be found by referring to the edge list for the current city).
3. If the current city has entries in its edge list go to step 4; otherwise, go to step 5.
4. Determine which of the cities in the edge list of the current city has the fewest entries in its own edge list. The city with the fewest entries becomes the “current city”. Ties are broken at random. Go to step 2.
5. If there are no remaining “unvisited” cities, then STOP. Otherwise, choose at random an “unvisited” city and go to step 2.

Mn vnddndscdskjcndskn

2.6. *Alternating-position crossover (AP)*

The alternating position crossover operator simply creates an offspring by selecting alternately the next element of the first parent and the next element of the second parent, omitting the elements already present in the offspring.

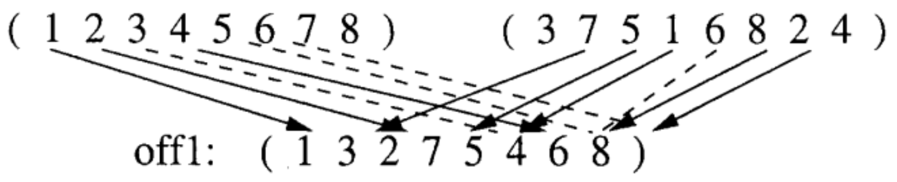


Figure 6. Alternating-position crossover (AP).

2.7. *Position based crossover (POS)*

The position based operator also starts by selecting a random set of positions in the parent tours. However, this operator imposes the position of the selected cities on the corresponding cities of the other parent.

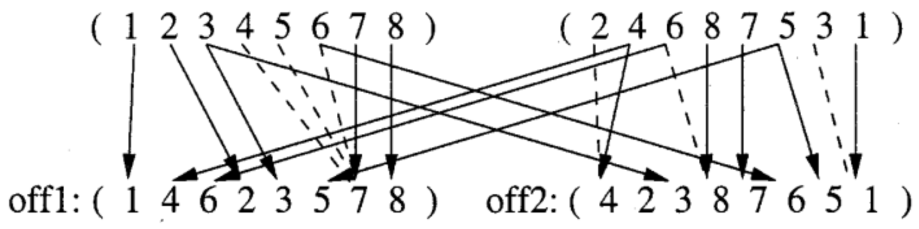


Figure 5. Position based crossover (POS).

3. Mutation Methods

3.1. *Displacement mutation (DM)*

The displacement mutation operator first selects a subtour at random. This subtour is removed from the tour and inserted in a random place.

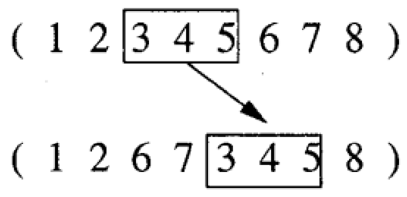


Figure 7. Displacement mutation (DM).

3.2. *Exchange mutation (EM)*

The exchange mutation operator randomly selects two cities in the tour and exchanges them.

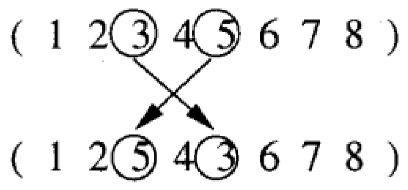


Figure 8. Exchange mutation (EM).

3.3. *Insertion mutation (ISM)*

The insertion mutation operator randomly chooses a city in the tour, removes it from this tour, and inserts it in a randomly selected place.

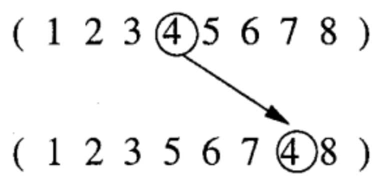


Figure 9. Insertion mutation (ISM).

3.4. *Simple inversion mutation (SIM)*

The simple inversion mutation operator selects randomly two cut points in the string, and it reverses the substring between these two cut points.

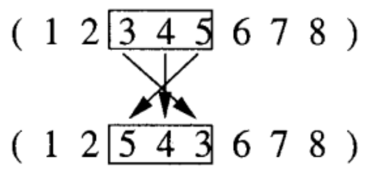


Figure 10. Simple inversion mutation (SIM).

3.5. *Inversion mutation (IVM)*

The inversion mutation is similar to the displacement operator. It also randomly selects a subtour, removes it from the tour and inserts it in a randomly selected position. However, the subtour is inserted in reversed order.

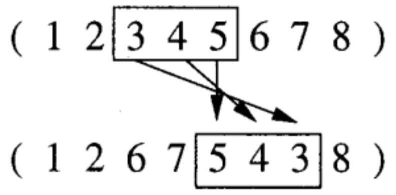


Figure 11. Inversion mutation (IVM).

3.6. *Scramble mutation (SM)*

The scramble mutation operator selects a random subtour and scrambles the cities in it.

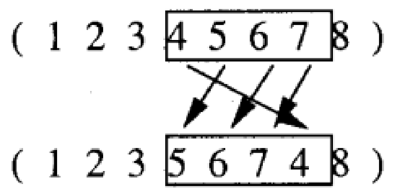


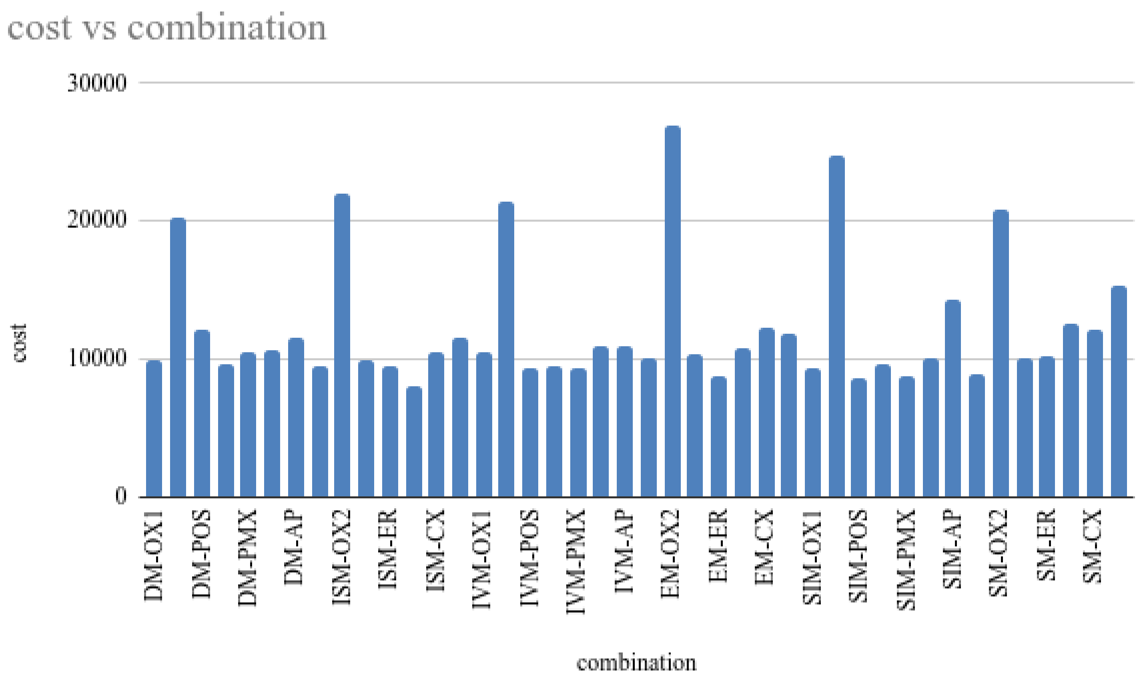
Figure 12. Scramble mutation (SM).

4. Results

Distances in kilometers between the 38 locations in Djibouti have been used in the empirical study. Indeed, this file consists of 38 coordinates of the locations.

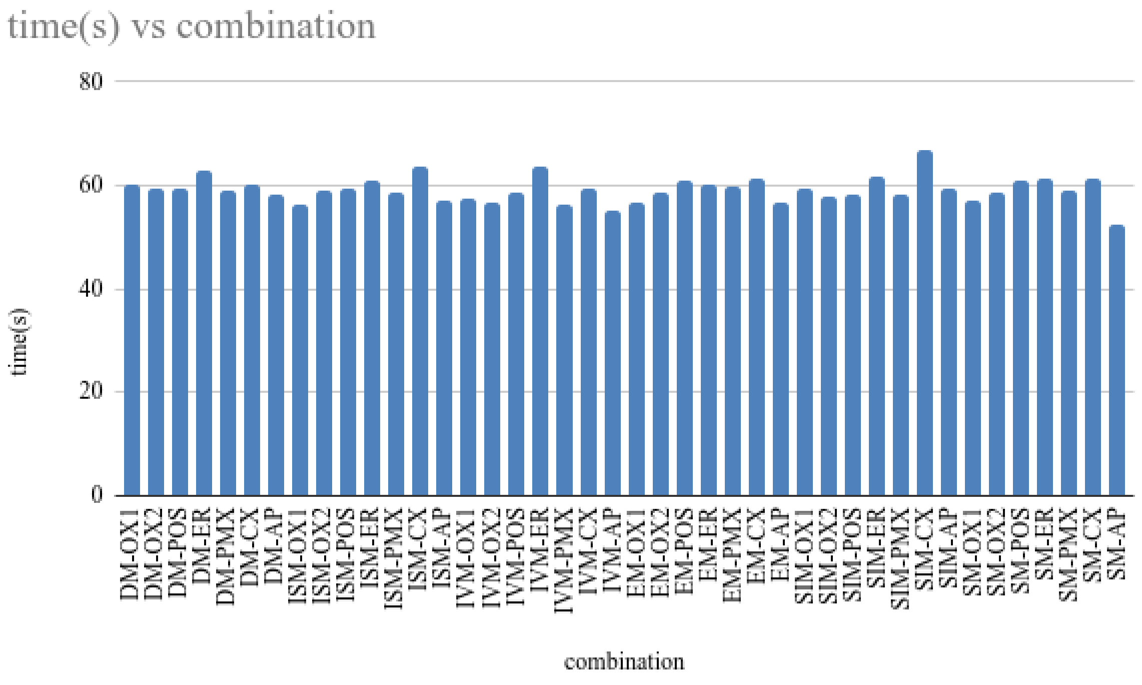
4.1. *Cost Comparison*

Chart 1 shows the best results and the average results obtained for each possible combination between the crossover and mutation parameters considered. Distances in kilometers have been used.



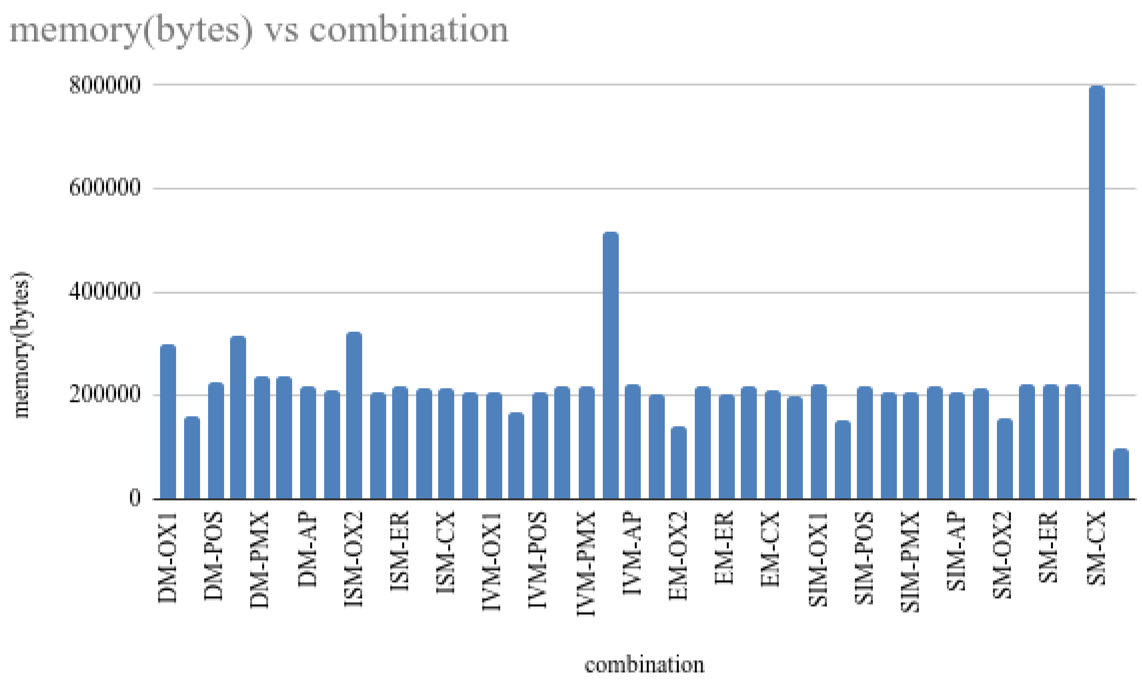
4.2. *Time Complexity Comparison*

Chart 2 shows the best results and the average time obtained for each possible combination between the crossover and mutation parameters considered. Times in seconds have been used.



4.3. *Memory Complexity Comparison*

Chart 3 shows the best results and the average memory usage obtained for each possible combination between the crossover and mutation parameters considered. Memory usage in — have been used.



4.4. Accurate results

combination	memory(bytes)	time(s)	cost
DM-OX1	300752	60.29824	9873.001542
DM-OX2	159668	59.319668	20193.49958
DM-POS	224729	59.409481	12027.32767
DM-ER	315908	62.798728	9562.171888
DM-PMX	235317	59.204807	10526.99335
DM-CX	235585	60.258834	10586.1653
DM-AP	216437	58.305733	11525.49882
ISM-OX1	210484	56.461665	9474.329294
ISM-OX2	322479	58.921787	21990.50088
ISM-POS	207294	59.316116	9953.146079
ISM-ER	215972	60.808017	9485.098451
ISM-PMX	212857	58.481026	8090.755715
ISM-CX	211918	63.560226	10489.73031
ISM-AP	207002	57.116105	11469.90681
IVM-OX1	206164	57.296184	10487.20722
IVM-OX2	165520	56.769581	21382.19442
IVM-POS	206603	58.772991	9357.847547
IVM-ER	217790	63.723461	9539.875722
IVM-PMX	219540	56.404971	9268.08281
IVM-CX	516976	59.535842	10932.60494
IVM-AP	222221	54.960731	10887.95898
EM-OX1	200989	56.52289	10080.0397
EM-OX2	141152	58.552327	26993.88751
EM-POS	216824	60.834943	10343.57765
EM-ER	201546	59.992901	8694.823018
EM-PMX	216496	59.632786	10766.62835
EM-CX	210599	61.444681	12250.05738
EM-AP	199864	56.814706	11744.716
SIM-OX1	220017	59.302128	9314.321704
SIM-OX2	153665	57.79178	24758.19597
SIM-POS	218887	58.167122	8548.67796
SIM-ER	207085	61.65067	9620.543041
SIM-PMX	205921	58.303939	8693.258585
SIM-CX	216345	66.610393	10035.66326
SIM-AP	207518	59.27296	14212.38908
SM-OX1	215071	57.23676	8932.086174
SM-OX2	156063	58.642873	20774.85327
SM-POS	220679	60.941046	10056.80389
SM-ER	220197	61.186335	10201.98032
SM-PMX	221787	59.178128	12543.13142
SM-CX	798443	61.224875	12072.96324
SM-AP	99041	52.451782	15333.10174