

Coevolutionary Memetic Algorithms for Solving Traveling Salesman Problem (TSP)

Şerife Uluçınar

Submitted to the
Institute of Graduate Studies and Research
in partial fulfillment of the requirements for the Degree of

Master of Science
in
Computer Engineering

Eastern Mediterranean University
January 2013
Gazimağusa, North Cyprus

Approval of the Institute of Graduate Studies and Research

Prof. Dr. Elvan Yılmaz
Director

I certify that this thesis satisfies the requirements as a thesis for the degree of Master of Science in Computer Engineering.

Assoc. Prof. Dr. Muhammed Salamah
Chair, Department of Computer Engineering

We certify that we have read this thesis and that in our opinion it is fully adequate in scope and quality as a thesis for the degree of Master of Science in Computer Engineering.

Asst. Prof. Dr. Ahmet Ünveren
Supervisor

Examining Committee

1. Prof. Dr. Omar Ramadan
2. Asst. Prof. Dr. Adnan Acan
3. Asst. Prof. Dr. Ahmet Ünveren

ABSTRACT

In this thesis, Coevolutionary Memetic Algorithms are used for solving the well-known Traveling Salesman Problem (TSP). Traveling Salesman Problem is NP-Complete which means no algorithm can solve this problem in a computing time that increases polynomially with respect to the problem size. The proposed solution approach to TSP is the combination of Coevolutionary Algorithms and Memetic Algorithms. The objective solution to TSP is to find the minimum tour length that the Traveling Salesman can make under some restrictions.

Coevolutionary Algorithms belong to the class of Evolutionary Algorithms. The main difference of Coevolutionary Algorithms from Evolutionary Algorithms is the way of interaction of individuals in the population. The individuals of the population should interact with each other to form a complete solution to the problem. Moreover, Memetic Algorithms are hybridized algorithms which combine the Evolutionary Algorithms with a local search method. Local search algorithms are the algorithms that refine the solutions by searching within the neighbourhood of promising solutions to find the better ones.

In experimental results, the proposed algorithm is tested with several test datas by different parameter values of the algorithm.

Keywords: Coevolutionary Algorithms, Memetic Algorithms, Traveling Salesman Problem.

ÖZ

Bu tezde, Birlikte Evrimleşen Memetik Algoritmaları iyi bilinen bir problem olan Gezici Satıcı Problemi'nin çözümü için kullanılmıştır. Gezici Satıcı Problemi, üzerinde çok çalışılan bir problem olması ile birlikte, bu problem hiçbir algoritma tarafından belirli bir hesaplama zamanında çözülebilecek bir problem olmaması ile de bilinir. Gezici Satıcı Problemi'ni çözmeye yaklaşımımız Birlikte Evrimleşen Algoritmalar ile Memetik Algoritmaları'nın birleşimi ile gerçekleşecektir. Bu problemdeki amacımız Gezici Satıcı'nın dolaşacağı şehirlerden oluşacak olan turdan bazı sınırlamalara uygun olarak en kısa tur mesafesini bulabilmektir.

Birlikte Evrimleşen Algoritmalar, Evrimsel Algoritmaların bir sınıfıdır. Birlikte Evrimleşen Algoritmalar ile Evrimsel Algoritmalar arasındaki esas fark, Birlikte Evrimleşen Algoritmaların nüfustaki bireylerin arasındaki etkileşim yoludur. Bu algoritmalarda üzerinde çalışılan problemde bütün bir çözüm elde edebilmek için, nüfustaki bireylerin birbirleriyle bir etkileşim içinde olmaları gerekir. Memetik Algoritmaları ise, Evrimsel Algoritmaların ve bir Bölgesel Arama Algoritması'nın birlikte kullanımı ile oluşan algoritmalarlardır. Bölgesel Arama Algoritmaları, Evrimsel Algoritmalar tarafından bulunan çözümün etrafında, bu çözümü daha iyi bir çözüm haline getirecek aramalar yapan algoritmalarlardır.

Birlikte Evrimleşen Memetik Algoritmaları, algoritma parametrelerinin farklı değerlerine göre birkaç test verileri ile test edilmiştir.

Anahtar Kelimeler: Birlikte Evrimleşen Algoritmalar, Memetik Algoritmaları, Gezici Satıcı Problemi.

ACKNOWLEDGEMENTS

First of all, I would like to sincerely thank to my supervisor Asst. Prof. Dr. Ahmet Ünveren for his support, patience, motivation, and assistance in my study throughout the research. His guidance helped me understand my research topic clearly. This thesis would not have been possible to complete without his inspiration.

Besides my supervisor, I would like to thank to Asst. Prof. Dr. Adnan Acan for his encouragement, comments, and support for my research.

I am so grateful for my family for their endless support, care, love and trust throughout my life. I would like to also thank to my fiancé for supporting me always.

TABLE OF CONTENTS

ABSTRACT	iii
ÖZ	iv
ACKNOWLEDGEMENTS	v
LIST OF FIGURES	viii
LIST OF TABLES	x
1 INTRODUCTION	1
2 TRAVELING SALESMAN PROBLEM	3
2.1 Traveling Salesman Problem (TSP).....	3
2.1.1 Related work on TSP	7
2.1.2 Description and Formulation of TSP	8
3 COEVOLUTIONARY MEMETIC ALGORITHMS AND LOCAL SEARCH ALGORITHMS	10
3.1 Evolutionary Algorithms.....	10
3.1.1 Components of Evolutionary Algorithms	11
3.1.1.1 Representation.....	11
3.1.1.2 Population	12
3.1.1.3 Evaluation Function (Fitness Function).....	12
3.1.1.4 Selection Mechanism	12
3.1.1.4.1 Tournament Selection	12
3.1.1.4.2 Roulette Wheel Selection.....	13
3.1.1.5 Recombination (Crossover) and Mutation.....	13
3.1.1.5.1 One-Point Crossover	14
3.1.1.5.2 Two-Point Crossover	14
3.1.1.5.3 Partially Matched Crossover (PMX).....	15

3.1.1.5.4 One-Point Mutation.....	16
3.1.1.5.5 Swap Mutation	17
3.1.1.5.6 Shift Mutation	17
3.2 Coevolutionary Algorithms.....	18
3.2.1 Cooperative Coevolutionary Algorithms	19
3.3 Memetic Algorithms	20
3.3.1 Local Search Algorithms.....	22
3.3.1.1 2-Opt Local Search Algorithm.....	23
3.3.1.2 Tabu Search Algorithm	24
4 SOLVING TRAVELING SALESMAN PROBLEM BY COEVOLUTIONARY MEMETIC ALGORITHMS	25
4.1 Cooperative Coevolutionary Memetic Algorithms for Solving TSP.....	25
4.1.1 Chromosome Representation	25
4.1.2 Construction of the Initial Population	26
4.1.3 Fitness Value Calculation	27
4.1.4 Genetic Operators.....	27
4.1.5 Local Search Algorithm	27
4.1.6 Algorithm Description	28
5 EXPERIMENTAL RESULTS	34
5.1 TSP Instances	34
5.2 Experimental Results for TSP	34
5.2.1 The Effect of Increased Population Size, Iteration Size	35
5.2.2 The Effect of Local Search Algorithm.....	36
6 CONCLUSION.....	43
REFERENCES.....	44

LIST OF FIGURES

Figure 2.1: Sample for initial cities.....	4
Figure 2.2: A Sample Distance Matrix for 10 cities in TSP	4
Figure 2.3: A sample instance and a tour in a TSP with 10 cities	5
Figure 2.4: A Symmetric TSP instance.....	6
Figure 2.5: An Asymmetric TSP instance.....	6
Figure 3.1: General Algorithm for EAs	11
Figure 3.2: An individual (chromosome) representation for TSP with 10 cities	11
Figure 3.3: The steps of Tournament Selection	12
Figure 3.4: One-Point Crossover	14
Figure 3.5: Two-Point Crossover.....	15
Figure 3.6: Partially Matched Crossover	16
Figure 3.7: One-Point Mutation Operator.....	17
Figure 3.8: Swap Mutation.....	17
Figure 3.9: Shift Mutation.....	18
Figure 3.10: Cooperative Coevolutionary Algorithm	20
Figure 3.11: Basic Memetic Algorithm Flowchart	21
Figure 3.12: Local Search Algorithm Pseudo-code	23
Figure 3.13: A TSP Tour without 2-Opt Local Search Method.....	23
Figure 3.14: A TSP Tour after 2-Opt Local Search Method.....	23
Figure 3.15: Pseudo-code for Tabu Search	24
Figure 4.1: Complete Chromosome Representation for TSP.....	25
Figure 4.2: Chromosome Representation for Sub-Populations.....	26
Figure 4.3: Flowchart of the Proposed Algorithm	29
Figure 4.4: Initial Population for First Loop.....	30

Figure 4.5: Division of an Individual into two Partial Solutions	30
Figure 4.6: Construction of Initial Sub-Populations	31
Figure 4.7: Combining Active-Frozen Sub-Populations.....	31
Figure 4.8: Combined Population of Sub-Populations	32

LIST OF TABLES

Table 5.1: Results-1 for the Cooperative Coevolutionary Algorithm without local search algorithm with small test datas	37
Table 5.2: Results-2 for the Cooperative Coevolutionary Algorithm without local search algorithm with increased initial population size in small test datas.....	37
Table 5.3: Results-3 for the Cooperative Coevolutionary Algorithm without local search algorithm.....	35
Table 5.4: Results-4 for the Cooperative Coevolutionary Algorithm without local search algorithm with increased PopSize.....	36
Table 5.5: Results-1 for Cooperative Coevolutionary Algorithm with 2-Opt local search algorithm.....	37
Table 5.6: Results-2 for Cooperative Coevolutionary Algorithm including 2-Opt with increased initial population size.....	38
Table 5.7: Results-3 for Cooperative Coevolutionary Algorithm with 2-Opt local search algorithm.....	39
Table 5.8: Results-4 for Cooperative Coevolutionary Algorithm with 2-Opt local search algorithm.....	40
Table 5.9: Results-5 for Cooperative Coevolutionary Algorithm with 2-Opt local search algorithm.....	40
Table 5.10: Results-6 for Cooperative Coevolutionary Algorithm with 2-Opt local search algorithm.....	40
Table 5.11: Results-7 for Cooperative Coevolutionary Algorithm including 2-Opt with increased initial population size.....	41
Table 5.12: Results-1 for Cooperative Coevolutionary Algorithm with Tabu Search	42

Chapter 1

INTRODUCTION

Evolutionary Algorithms (EAs) are search algorithms that are based on nature-inspired or bio-inspired search mechanisms. There are several types of EAs in the literature of Evolutionary Computing. Evolutionary Algorithms are population-based search algorithms. The aim of EAs is to find a solution to a given problem by searching the solution space using an evolving population of potential solutions. EAs have been applied to many different kinds of problems belonging to the class of combinatorial optimization problems, i.e. Traveling Salesman Problem (TSP), real valued optimization problems, and machine learning tasks.

Coevolutionary Algorithms (CAs) are one of the types of EAs which were introduced by Potter and de Jong [1] in 2000. CAs differ from the EAs in a way of evaluation of the candidate solutions that are based on the interaction of the solutions with each other. CAs manage two or more populations to evolve and interact individuals within populations. That is, the aim of CAs is to reduce the complexity of the given problem by decomposing the problem into sub-problems. Moreover, there are two types of Coevolutionary Algorithms which are Cooperative Coevolutionary Algorithms and Competitive Coevolutionary Algorithms. In this thesis, Cooperative Coevolutionary Algorithms are examined.

Memetic Algorithms (MAs) are hybrid EAs that are combined with local search algorithms to refine the solutions found by the EA. Furthermore, Local Search (LS)

Algorithms try to reach an improvement by making small moves around the current solution. There are several local search techniques such as Tabu Search [13], 2-Opt [16], 3-Opt [16] and etc. In this thesis, Tabu Search and 2-Opt local search techniques are used as the two local search procedures.

The proposed algorithm for solving TSP is a combination of Coevolutionary Algorithms and Memetic Algorithms which belongs to the class of Coevolutionary Memetic Algorithm. TSP is an NP-Hard problem which means no known algorithm can solve this problem in a polynomial time. In TSP, a salesman starts travelling from a city and passes through each city only once by the constraint of returning to the starting city. The goal of the salesman is to find the shortest tour all over the cities.

The experimental evaluations are carried out using several TSP instances that are taken from the TSPLIB [14], the library of the TSP instances. Using Coevolutionary and Memetic Algorithms over different number of generations, it is observed that without local search algorithms, a global search algorithm may not reach the possible optimal solution by itself. Therefore, a local search algorithm is generally used together with metaheuristics in the combinatorial optimization problems.

The arrangements of the remaining chapters are as follows: in Chapter 2, Traveling Salesman Problem and the recent research about this problem will be discussed. In Chapter 3, Coevolutionary Algorithms, Memetic Algorithms and local search methods are presented. Moreover, the proposed work of Coevolutionary Memetic Algorithms for solving TSP will be discussed in Chapter 4. The experimental results that are found by using the described algorithms are presented in Chapter 5. Finally, conclusion and future work are discussed in the Chapter 6.

Chapter 2

TRAVELING SALESMAN PROBLEM

2.1 Traveling Salesman Problem (TSP)

Traveling Salesman Problem is a well-known combinatorial optimization problem which is NP-Hard, where no algorithm can solve this problem in a polynomially increasing computing time [21]. TSP is one of the problems that is studied most widely as a research problem in the field of optimization. It is handled by many heuristics such as Ant Colony Optimization [20], Genetic Algorithms [17], Tabu Search [13] and Simulated Annealing [17] as a benchmark problem.

The TSP was firstly defined by the mathematicians William Rowan Hamilton and Thomas Penyngto Kirkman in the 1800s. The study for the general form of the TSP has been done by the mathematicians in the 1930s by Karl Menger in Vienna and Harvard [6]. Later, TSP was promoted by Hassler Whitney at Princeton University. Also, the name for the problem, Traveling Salesman Problem, was introduced by Hassler Whitney [6]. The improvements on solving TSP started to appear in the 1950s. Since then, TSP became a popular combinatorial optimization problem which has been studied by many researchers as benchmark problem for many heuristics. The problem became popular for many of the research areas such as computer science, mathematics, chemistry, physics and many other research areas. Later on, in 1991 TSPLIB was introduced [14]. TSPLIB is the library of benchmark sample instances for the TSP. Since then, this library has been used to compare the results

that researchers found by their algorithms with the ones available in the TSPLIB. Therefore, in this thesis, this library is also used to compare our results.

In Traveling Salesman Problem, cities and the distances between cities are given. Salesman starts travelling from the starting city and passes through each city only once and returns to the starting city at the end of the tour. The aim of the salesman is to find the shortest possible tour length between the cities. Figure 2.1 and Figure 2.2 below show a TSP instance and the distances between cities respectively.

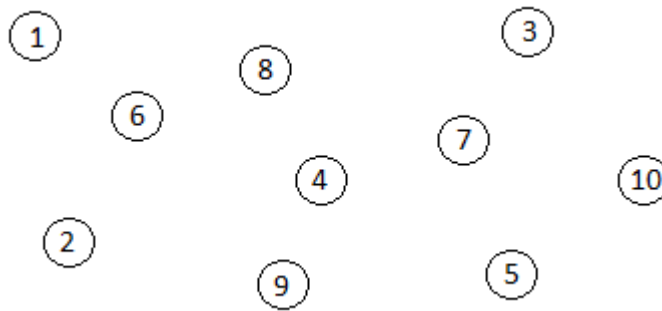


Figure 2.1: Sample for initial cities

Cities	1	2	3	4	5	6	7	8	9	10
1	0	2.6822	4.3577	7.3958	5.2305	6.0597	6.3892	6.9655	4.1359	6.0125
2	2.6822	0	3.0500	5.3794	7.3350	6.0389	5.0306	8.6914	6.4845	8.2578
3	4.3577	3.0500	0	3.1930	6.7818	3.3783	2.0778	7.4283	6.4777	7.8220
4	7.3958	5.3794	3.1930	0	9.7367	5.0776	2.0230	10.0001	9.5926	10.7789
5	5.2305	7.3350	6.7818	9.7367	0	5.3987	7.8830	2.1012	1.3532	1.0439
6	6.0597	6.0389	3.3783	5.0776	5.3987	0	3.0554	5.0842	5.8083	6.3644
7	6.3892	5.0306	2.0778	2.0230	7.8830	3.0554	0	8.0081	7.8862	8.9164
8	6.9655	8.6914	7.4283	10.0001	2.1012	5.0842	8.0081	0	3.4443	2.2721
9	4.1359	6.4845	6.4777	9.5926	1.3532	5.8083	7.8862	3.4443	0	1.8917
10	6.0125	8.2578	7.8220	10.7789	1.0439	6.3644	8.9164	2.2721	1.8917	0

Figure 2.2: A Sample Distance Matrix for 10 cities in TSP

This distance matrix is used to calculate the tour length of the visited cities. The above distance matrix example is for a TSP instance with 10 cities. The distance matrix has $N \times N$ dimension which is 10×10 cities in this example. Figure 2.3 illustrates a possible tour in a TSP with 10 cities.

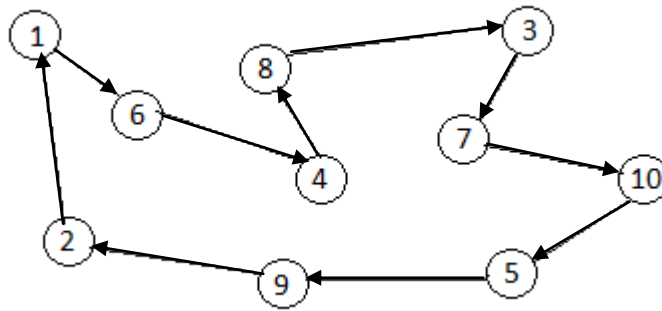


Figure 2.3: A sample instance and a tour in a TSP with 10 cities

The tour length for the above instance is the summation of the distances from the distance matrix. A possible tour according to the Figure 2.3 could be as (1,6,4,8,3,7,10,5,9,2,1). The tour length of this possible tour is the summation of distances between cities which results as 51,1237.

TSP instances are classified into groups based on the arrangement of distances and the types of the graph. These classified groups are called Symmetric TSP and Asymmetric TSP which are explained below.

- **Symmetric TSP:** In Symmetric TSP, the distance between two cities is symmetric. Meaning, the distance from *city A* to *city B* and the distance from *city B* to *city A* is the same [18].

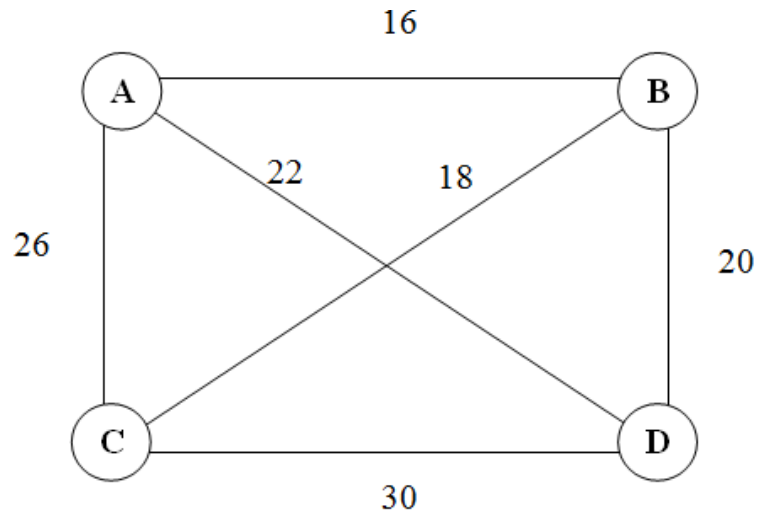


Figure 2.4: A Symmetric TSP instance

- **Asymmetric TSP:** In Asymmetric TSP, the distance between two cities is asymmetric. That is, the distance from *city A* to *city B* and the distance from *city B* to *city A* may be different. Also, the path between cities may not exist [19].

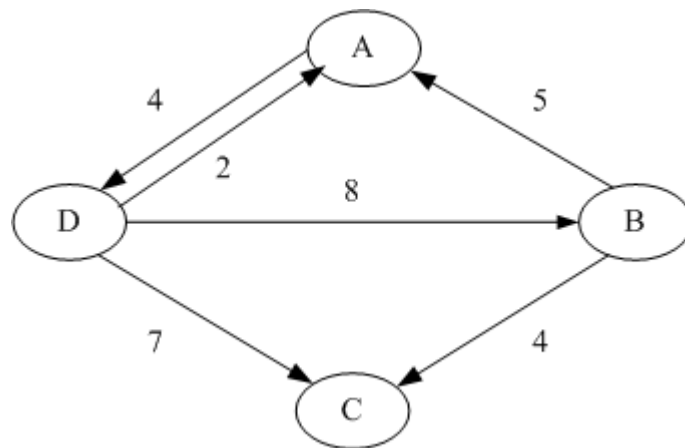


Figure 2.5: An Asymmetric TSP instance

However, in this thesis, only the Symmetric TSP instances are handled which are collected from the symmetric sample instances in TSPLIB.

2.1.1 Related work on TSP:

Since TSP is very popular combinatorial optimization problem, it has been considered to many heuristics as a benchmark problem. It is modelled as a single and multi-objective optimization problem and here are some of the researches that have been done on the TSP with different heuristics.

Haibin Duan and Xiufen Yu [2] have showed a way of applying memetic algorithms on TSP. They used a Hybrid Ant Colony Optimization with Memetic Algorithms. This approach combines the global and local search techniques. Here, memetic algorithm is used to optimize the adjustable parameters of the ant colony optimization. Results have showed that this approach is helpful to improve solutions that are stuck at the local minima of the search space.

Zar Chi Su Su Hlaing and May Aye Khine [7] presented an approach for solving TSP which is Improved Ant Colony Optimization Algorithm. This algorithm is another version of the Ant Colony Optimization (ACO) which is based on the candidate list strategy. Also, the authors proposed a dynamic heuristic parameter update by local search.

Mohd. Junedul Haque and Khalid. W. Maghld [8] presented an Evolutionary, Memetic Algorithm and Edge Assembly Crossover for solving TSP. In this approach, Genetic and Memetic Algorithms are combined and a local search technique helped to improve the results for the TSP.

Another solution approach to TSP is presented using An Immune Coevolutionary Algorithm for N-th Agent's Traveling Salesman Problem by Naruaki Toma, Satoshi Endo and Koji Yamada [4]. An immune optimization algorithm is proposed, which is based on a biological immune coevolutionary phenomenon and cell-cooperation to

solve n-th agents' TSP that is a kind of division-of-labour problems. In this approach, it is focused on the issues of distribution of work for agents. The coevolutionary model searched for a solution with three kinds of distribution; equaling work assignment, optimizing a work-cost individually and optimizing all amount of work-costs. These distributions can be done through interactions between two kinds of agents, one of them is called immune agent and the other one is called antigen agent. This method solves the problem through combination of division, integration and coevolutionary approach. These functions are based on local interactions. These interactions are between agents and also between agents and environment.

These researches and works show that there are various ways of applying different algorithms and different local search techniques for solution of TSP. Moreover, local search techniques were used to improve the results in the most of the research that are mentioned above. This shows us the effect of local search in combinatorial optimization problems.

2.1.2 Description and Formulation of TSP

In order to have a complete solution for the TSP, the constraint for the problem must be considered to obtain a feasible solution. Therefore, the objective, feasibility and formulation of the TSP are explained below.

- **Objective:** The objective of TSP is to minimize the total distance travelled.

At the end of a tour, the result, which is the cost of the tour, is the total distance travelled. As mentioned before, single-objective optimization is applied for TSP.

- **Feasibility:** A feasible solution of a TSP is a tour which satisfies all problem constraints. That is, each city is visited once and only once, and the tour ends with the starting city.
- **Formulation:** Since, the objective is to find the shortest tour from the given cities; we can calculate the distance between the cities by the following formula which is the Euclidean Distance. The Euclidean Distance between two cities can be given as follows:

$$x = (x_1, x_2, x_3, \dots, x_d) \text{ and } y = (y_1, y_2, y_3, \dots, y_d)$$

$$D_{xy} = \sqrt{\left(\sum_{i=1}^d (x_i - y_i)^2\right)} \quad \text{Eq. 2.1}$$

where D_{xy} is the distance between x and y , and d represents the d -dimensional space, x_i and y_i are the locations of two cities which are used to find the distances between them. The Euclidean distance formula is used in optimization problems to compare the distances and to find the length of a tour in TSP.

Let $C = \{c_1, c_2, \dots, c_n\}$ be the feasible tour for n cities of a TSP problem. A Tour Length can be defined as:

$$TL = \left(\sum_{i=1}^{n-1} D_{c_i c_{i+1}}\right) + D_{c_{i+1} c_1} \quad \text{Eq. 2.2}$$

Objective is to find:

$\min TL_i, \forall i=1, \dots, k$ where k is the number of feasible solutions found by the proposed method.

Chapter 3

COEVOLUTIONARY MEMETIC ALGORITHMS AND LOCAL SEARCH ALGORITHMS

3.1 Evolutionary Algorithms

In the literature, there are many variants of Evolutionary Algorithms (EAs). EAs are population-based and generally stochastic algorithms. The main idea behind them is the same: a population of individuals is affected by the environmental pressure causing natural selection and this natural selection causes increase in fitness of the population [9]. Fitness of the population is measured by the quality function either to be maximized or minimized, it depends on the problem.

In EAs, the initial candidate solutions are created randomly and these candidate solutions are measured by the quality function resulting as the fitness of the candidate solution. Candidates that have better fitness values are selected to seed the next generation by applying variation operators which are crossover and mutation operators. Applying both of the variation operators, result in a set of new candidates that are called offspring. This process iterates until a good solution is found or the iteration limit is reached. Furthermore, this evolutionary process leads the population to better candidate solutions in each generation. The Figure 3.1 shows how EAs work.

```

BEGIN
  INITIALIZE population with random candidate solutions;
  EVALUATE each candidate;
  REPEAT UNTIL (Termination Condition is satisfied) DO
    SELECT parents;
    RECOMBINE pairs of parents;
    MUTATE the resulting offspring;
    EVALUATE new candidates;
    UPDATE new population for the next generation;
  END REPEAT
END

```

Figure 3.1: General Algorithm for Evolutionary Algorithm [9]

3.1.1 Components of Evolutionary Algorithms

There are some components for the EAs which are explained as follows:

3.1.1.1 Representation

This is the first step of an EA. Representation is where the real world problem is transformed into the EA world. This relation is explained with the phenotype and genotype where phenotype is the feature of an individual resulting from the interactions of its environment and genotype. In addition, genotype is the encoded version of the phenotype as the individuals (possible solution also called chromosome) in EAs. For example, 5 would represent the phenotype and 0101 would represent the genotype of the problem. Designing a chromosome representation for a problem should consider the structure of the problem. For example, for the TSP, which is a permutation problem, a representation for 10 cities can be seen as follows:

2	6	3	1	10	5	7	9	4	8	2
---	---	---	---	----	---	---	---	---	---	---

Figure 3.2: An individual (chromosome) representation for TSP with 10 cities

3.1.1.2 Population

Initial population is the set of individuals created randomly for a given problem with a given population size. The initial population for TSP is formed by random permutations of the cities.

3.1.1.3 Evaluation Function (Fitness Function)

Evaluation function is also known as fitness function. The role of the fitness function is to measure the fitter individuals that adapt themselves to their environments. Fitness function measures the quality of individuals of the population. Fitness function for TSP measures the total length of the tour travelled which was explained in section 2.1.2.

3.1.1.4 Selection Mechanism

Selection is the operator to select the fitter or better individuals to seed the next generations. It helps to improve the individuals of the next generation by allowing the better ones to become their parents. There are some operators for the selection mechanisms. The mostly used ones in the EAs are the Tournament Selection and Roulette Wheel Selection.

3.1.1.4.1 Tournament Selection

In tournament selection, there is a parameter k , called tournament size, which is less than the population size. Steps for tournament selection are shown below.

1. Pick a random spot within the population,
2. Get as many individuals as specified by tournament size= k ,
3. Sort the population by fitness,
4. Return the best two as parents.

Figure 3.3: The steps of Tournament Selection

The selected individuals of the tournament are the individuals with the highest fitness values of the k tournament competitors. Tournament selection gives several chances for the non-selected individuals to be selected.

3.1.1.4.2 Roulette Wheel Selection

In Roulette Wheel Selection (RWS) the probability of being selected is computed by the following formula:

For each individual i , with fitness value f_i ,

$$P_i = \frac{f_i}{\sum_{j=1}^N f_j} \quad \text{Eq. 3.1}$$

where N = Population Size.

Roulette Wheel Selection is also named as fitness proportional selection. In RWS, all individuals are sorted in decreasing order of P_i values. Then, in order to implement a distribution with respect to P_i values,

- A uniformly random value is generated,
- A partial sum over the list of sorted P_i values is started,
- The first individual in which the partial sum exceeds, the random value is the selected individual.

3.1.1.5 Recombination (Crossover) and Mutation

Crossover is a variation operator that merges the information from two parents and creates two offsprings from them. They are usually seen as the main search operator in EAs. There are many different types of crossover operators which are usually designed in a problem-specific way.

Mutation operator is another variation operator which is applied to one individual resulting in one offspring. Mutation operator is usually designed in a problem-

specific manner. This operator generally brings the new information to the problem by mutating one or more genes of the individual (chromosome). Some of the Crossover and Mutation operators are explained below.

3.1.1.5.1 One-Point Crossover

In one-point crossover, a randomly selected crossover-point from the chromosome sets the division point. The selected crossover-point divides the chromosomes into two parts as shown below.

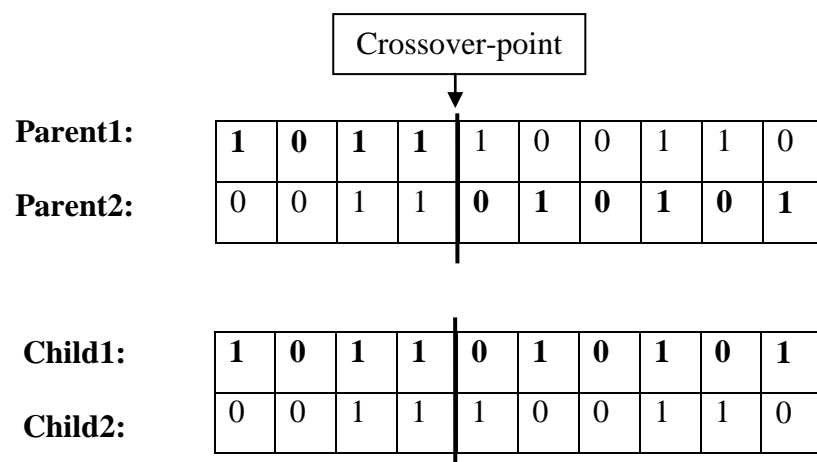


Figure 3.4: One-Point Crossover

The first part of the Parent1 is directly copied to Child1, and second part of the Parent2 is appended to Child1. For Child2, the first part of the Parent2 is copied directly to Child2, and the second part of Parent1 is appended to Child2. By this way, two offspring are created.

3.1.1.5.2 Two-Point Crossover

In this crossover operator, two crossover-points are randomly selected from the chromosomes dividing them into three parts. Figure 3.5 below shows how this is done.

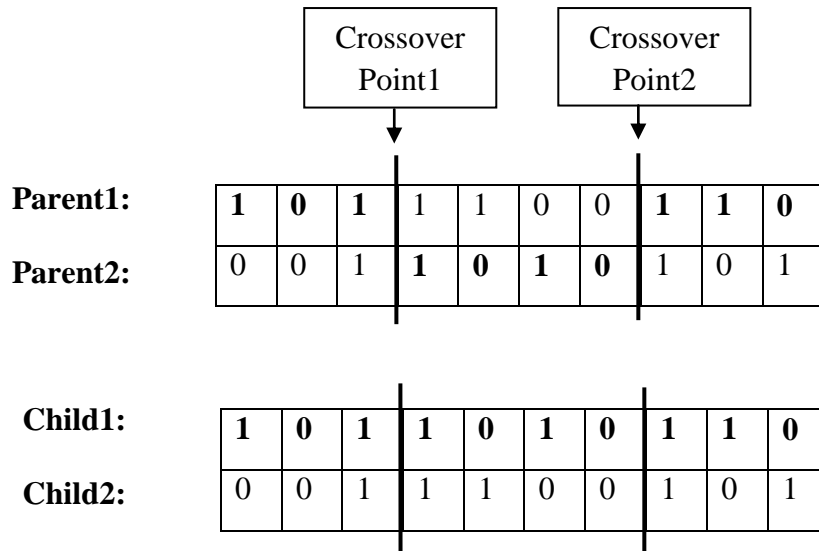
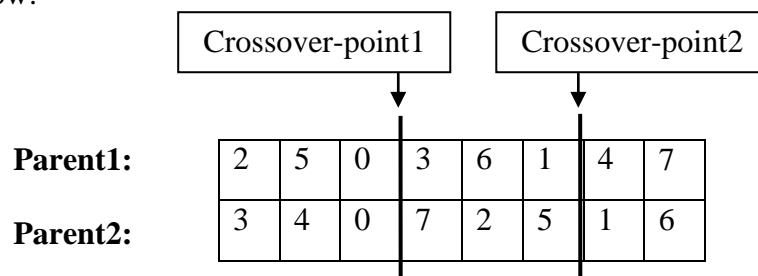


Figure 3.5: Two-Point Crossover

Two-Point crossover is the same as one-point crossover but in this crossover operator there are two crossover-points. First and third parts of Parent1 and Parent2 are directly copied to the same positions of Child1 and Child2 respectively. Then, second parts of Parent1 and Parent2 are swapped with each other and second part of Parent1 is copied to the second part of Child2 and second part of Parent2 is copied to the second part of Child1.

3.1.1.5.3 Partially Matched Crossover (PMX)

PMX takes a sub path between two random cut points from one parent, and completes the tour by preserving the order and position of as many cities as possible from the other parent [15]. So, illegal tours are avoided by a mapping. The PMX is illustrated below:



In this example, 3 in Parent1 is mapped to 7 in Parent2. Moreover, 6 in Parent1 is mapped to 2 in Parent2. Also, 1 in Parent1 is mapped to 5 in Parent2. So, here steps of creating offsprings can be seen.

Step 1: Swapping 3 and 7.

Child1:	2	5	0	7	6	1	4	3
Child2:	7	4	0	3	2	5	1	6

Step 2: Swapping 6 and 2.

Child1:	6	5	0	7	2	1	4	3
Child2:	7	4	0	3	6	5	1	2

Step 3: Swapping 1 and 5.

Child1:	6	1	0	7	2	5	4	3
Child2:	7	4	0	3	6	1	5	2

Figure 3.6: Partially Matched Crossover

So, the resulting offsprings are the ones that are found in Step 3. This crossover operator is problem-specific which is designed for permutation problems. As shown, there is no repetition in the permutation which ensures not having infeasible solutions for the TSP.

3.1.1.5.4 One-Point Mutation

For one-point mutation, a point in the chromosome is selected randomly and mutated. Moreover, in this operator one parent is selected and one offspring is created from this parent. This mutation operator causes the replacement of a gene

with another gene in the chromosome. Also, it is very easy to implement. Figure 3.7 illustrates one-point mutation.

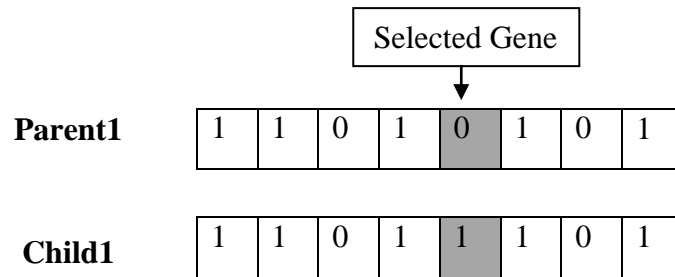


Figure 3.7: One-Point Mutation Operator

Selected gene which is in grey colour is mutated from 0 to 1. A new offspring is created by one-point mutation.

3.1.1.5.5 Swap Mutation

In this mutation operator two points are randomly selected to be swapped. Moreover, it is very easy to implement this operator. Illustration of this operator is given below in Figure 3.8.

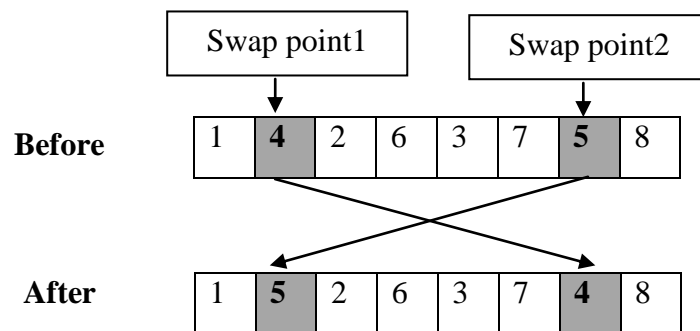


Figure 3.8: Swap Mutation

3.1.1.5.6 Shift Mutation

In shift mutation, two points are randomly selected as point1 and point2. Then, point2 replaces the position of point1 and the genes between the points shift forward until the position of point2 is reached. Illustration of the shift mutation is shown below in Figure 3.9.

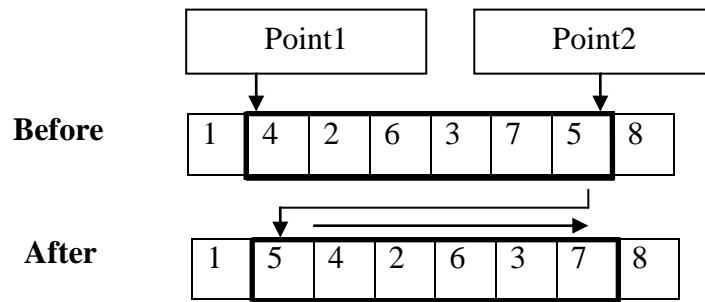


Figure 3.9: Shift Mutation

3.2 Coevolutionary Algorithms

Coevolutionary Algorithms are the collection of Evolutionary Algorithms in which evaluation of individuals is based on interactions between individuals. Therefore, these algorithms try to manage two or more populations simultaneously, by evolving populations and allowing interactions between individuals of the populations. That is, an individual can be evaluated by having an interaction with other individuals of other populations.

The goal of Coevolutionary Algorithms is to improve the results from each population separately. The interactions of individuals with other individuals can be done with individuals in all other population or by random pairings. The populations that work separately on a problem reduce the complexity of the problem by giving separate results for the problem. Therefore, in order to have a complete solution for a problem, the results from each sub-population are merged together in a coevolutionary algorithm.

There are two types of coevolutionary algorithms which are called Cooperative Coevolutionary Algorithms and Competitive Coevolutionary Algorithms. In this thesis, it has only been dealt with the Cooperative Coevolutionary Algorithms. In the literature, coevolutionary algorithms have been applied to many different problems

such as function optimization [12], instance selection [5], large scale optimization [3] and many others.

3.2.1 Cooperative Coevolutionary Algorithms

Cooperative coevolution means the co-existence of some interacting populations that are evolving simultaneously [22]. Cooperative coevolution was introduced by Potter and de Jong in 2000 [1]. They solved a complex problem by decomposing it into sub-problems and coevolving the solutions of each sub-problem. In cooperative coevolutionary algorithms, the given problem is decomposed into sub-problems which this technique is called divide and conquer. Each sub-problem represents sub-populations. Sub-populations have individuals (collaborators) that are parts of the complete solution. These collaborators of the sub-problems merge within each other and cooperatively form the complete solution for the problem. Then, the complete solution is evaluated by the fitness function. Moreover, merging of individuals between sub-populations can be made by one of the following ways:

- Evaluate an individual against every single individual in the other population,
- Use random individual,
- Use the best individual of last generation.

The cooperative coevolutionary algorithms work in the following way: first of all, the given problem is decomposed into sub-problems meaning that the population of the problem is divided into sub-populations. Each sub-population may have the same or different population size for the problem. For the initial generation, the individuals of each sub-population are matched with randomly chosen individuals from other sub-populations. Then individuals of each sub-population are evaluated by the fitness function and the best individual from each sub-population is found. The actual cooperation starts with the next generation. In order to have cooperation between

individuals, the algorithm should cycle in a repeated phase. This phase includes the active and frozen sub-populations. In a cycle there should be only one active sub-population, while other sub-populations are frozen. In order to combine these active and frozen sub-populations, all individuals from active sub-population are matched with the best found individuals from frozen sub-populations. By this way, the complete global form of the solution can be found for the problem. Moreover, for the next generation, the formerly active sub-population becomes the frozen sub-population this time, and the formerly frozen sub-population becomes the active sub-population. That is, turns are taken by these sub-populations in the run of the algorithm. This cycling continues until the condition of termination is reached. For better understanding of the cooperation between individuals the pseudo-code for the algorithm is given in Figure 3.10.

1. Divide Population into two or more sub-populations,
2. Match individuals randomly from all other sub-populations,
3. Evaluate fitness of each individual in sub-populations,
4. Find the best individual of each sub-populations,
5. Coevolution starts here;
 - a) Set active and frozen sub-populations,
 - b) Match all individuals in active sub-population with the best individuals of frozen sub-populations,
 - c) Composition of the best individuals represents the solution to the problem.
 - d) End of evolution.

Figure 3.10: Cooperative Coevolutionary Algorithm

3.3 Memetic Algorithms

Memetic Algorithms (MAs) are usually called hybrid EAs which combines an EA as global search to perform exploration and local search method as local search to perform exploitation for the given problem [11]. MAs are population-based metaheuristics that include much hybridization of EAs, Simulated Annealing (SA)

and Tabu Search. Moreover, MAs are a way of understanding how population-based and local search algorithms work together to achieve hybridization. Therefore, the key issue of a memetic algorithm is to have a local search method with a metaheuristic algorithm. Furthermore, MAs are popular in some of the areas such as Combinatorial Optimization, Biochemistry, Networking and Communication. Figure 3.11 below is the basic memetic algorithm flowchart.

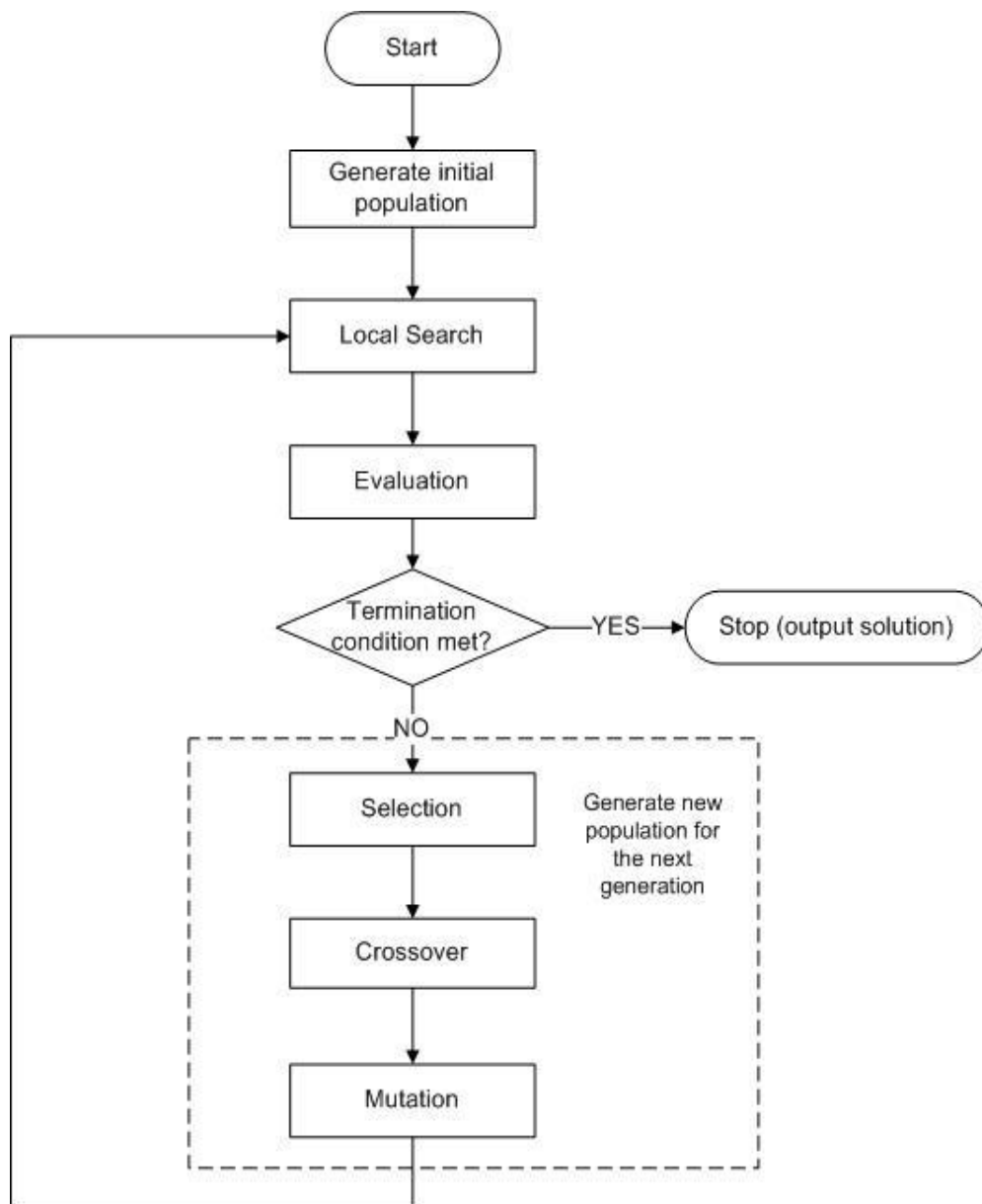


Figure 3.11: Basic Memetic Algorithm Flowchart

As it can be seen, MA is composed of mainly three components which are selection of parents, reproduction on selected parents and resulting new population. The first component which is the selection operator can be any of the selection mechanism depending on the problem space. Second component which is the reproduction consists of crossover, mutation operators and individual improvement. Crossover and mutation operators can be any of them which are suitable for the problem. Moreover, local search aims to improve the quality of the current solution. Lastly, the new population is the updated version of the current population after local search procedure of the algorithm. As a result, while this cycle continues, local search algorithms improve the solutions by searching around the current solution space. Since the memetic algorithms are hybrid algorithms combined with local search algorithms.

3.3.1 Local Search Algorithms

A local search algorithm [11] is a metaheuristic method that iteratively tries to improve a candidate solution for optimizing. Local search is also called exploitation, meaning the ability of an algorithm to move into an improved direction. Local search algorithms try to solve a given problem by moving from solution to solution in the search space of the problem. By moving in the search space, they try to make local changes to the solution to refine it. Local search algorithms continue to search for better solutions until a certain amount of iteration is reached or the best solution is found. Pseudo-code for local search algorithm is illustrated in Figure 3.12.


```

Local Search Procedure (current solution)
BEGIN
  REPEAT
    new = GenerateNeighbour (current solution);
    if (FitnessEvaluation(new) < FitnessEvaluation (current))
      current = new;
    end if
  until TerminationCondition;
  return current
END

```

Figure 3.12: A Local Search Algorithm

They are used in many optimization problems and in many other areas. Examples of local search algorithms can be 2-Opt and Tabu searches which are explained in the next sections.

3.3.1.1 2-Opt Local Search Algorithm

2-Opt local search algorithm is also called Pairwise Exchange. It removes two edges iteratively and replaces them with two different edges, then reconnects the edges resulting in a new shorter tour. This method is in the group of k-opt method. In k-opt method, k is the size for the edges. K-opt method deletes the k disjoint edges from the given tour [15]. This method is very suitable for the TSP that is why it is used as the local search method in this thesis. Figure 3.13 shows a tour before applying 2-Opt local search method.

7	6	8	10	9	5	1	2	3	4	7
---	---	---	----	---	---	---	---	---	---	---

Figure 3.13: A TSP Tour without 2-Opt Local Search Method

7	6	8	10	5	9	1	2	3	4	7
---	---	---	----	----------	----------	---	---	---	---	---

Figure 3.14: A TSP Tour after 2-Opt Local Search Method

After applying 2-Opt local search method to the tour in Figure 3.13, two edges are replaced with two different edges which is shown in Figure 3.14. The edges that are exchanged with each other are shown in bold, which are 5 and 9, in Figure 3.14.

3.3.1.2 Tabu Search Algorithm

Tabu search is another local search algorithm. It is widely used in many problems. The main idea behind tabu search is to move recently visited solution, which is called tabu, into a tabu list. A tabu list is the list that contains the visited solutions which are tabus. This process is done to make the exploration of new solutions easily.

Tabu search is also used in this thesis. Figure 3.15 below illustrates the pseudo-code for tabu search.

<pre> 1. Choose (construct) an initial solution S_0, 2. Set $S := S_0$, $f^* := f(S_0)$, $S^* := S_0$, $T := 0$, 3. while <i>termination condition not satisfied</i> do Select S in $f(S')$, $S' \in \overline{N}(S)$ if $f(S) < f^*$, then set $f^* := f(S)$, $S^* := S$; Record tabu for the current move in T (delete oldest entry if necessary) Update tabu and aspiration conditions 4. end while </pre>

Figure 3.15: Pseudo-code for Tabu Search [13]

In this algorithm, S is the current solution, S^* is the best-known solution, f^* is the value of S^* , $N(S)$ is the neighbourhood of S , $\overline{N}(S)$ is the subset of $N(S)$ that is non-tabu and T is the tabu list. When a tabu move results in a solution better than any visited so far, then its tabu classification may be overridden. A condition that allows such an override is called an aspiration criterion.

Chapter 4

SOLVING TRAVELING SALESMAN PROBLEM BY COEVOLUTIONARY MEMETIC ALGORITHMS

4.1 Cooperative Coevolutionary Memetic Algorithms for TSP

In this chapter, the proposed work for the solution of TSP by Coevolutionary Memetic Algorithms is discussed. Combination of Cooperative Coevolutionary and Memetic Algorithms employing genetic operators and local search algorithms are presented. The proposed algorithm is described in detail in the following sections.

4.1.1 Chromosome Representation

A complete individual for the TSP is represented in the following way. If we are given n within a set C , a tour of n cities is represented by a permutation of cities from C_1 to C_n . In order to have a complete tour, salesman has to return to the starting city. Therefore, to represent a complete tour the starting city of the tour is put to the $(n+1)^{\text{st}}$ position of the chromosome. Moreover, the fitness value of the tour is stored in the $(n+2)^{\text{nd}}$ position of the chromosome. The following Figure 4.1 illustrates the chromosome representation of a generic TSP.

C_1	C_2	C_3	...	$C_{1(n+1)}$	Fitness $_{n+2}$
-------	-------	-------	-----	--------------	------------------

Figure 4.1: Complete Chromosome Representation for TSP

The above representation is for a complete individual of the TSP. However, we have one more representation in our algorithm.

As it was mentioned before, our main algorithm is Cooperative Coevolutionary Algorithm which is based on divide and conquer approach [10]. Therefore, the population is divided into two sub-populations. The sub-populations' representation is as follows. The division of the population is made equally, that is, two sub-populations have the same amount of cities. So, if we have $n/2$ amount of cities in each sub-population, representation of cities starts from C_1 to $C_{n/2}$. Also, the fitness value of the partial individual is represented in the $n+1$ position of the chromosome. This representation of chromosome is applied to both sub-populations. It should be noticed that, in the sub-populations' representation, there is no complete tour. That is, we don't use the ending city which was the starting city in the complete individual representation. Moreover, for the fitness of each sub-population, it is represented in the $(n/2) + 1$ position of the chromosome. Figure 4.2 below illustrates the chromosome representation for the sub-populations.

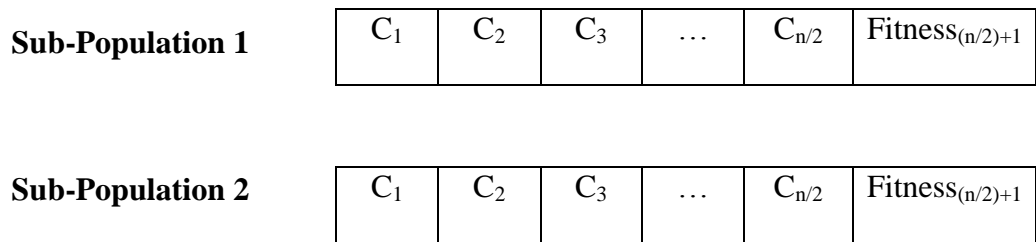


Figure 4.2: Chromosome Representation for Sub-Populations

4.1.2 Construction of the Initial Population

The initial population is constructed randomly in which each of the individual is different permutations of the given cities. Initially, the problem has complete individuals meaning it has complete tours. Later in the algorithm, when the cooperation starts the population of the problem is divided into sub-populations.

4.1.3 Fitness Value Calculation

Fitness value of each tour is calculated by Euclidean Distance formula which is given in Eq.2.1 and Eq.2.2. This formula uses the distances between cities to calculate the distance travelled by the salesman. In order to form a distance matrix for the cities, the x and y coordinates of the cities are given in advance. So, by using these coordinates in Euclidean Distance Formula, it is easy to calculate the total distance travelled by the salesman.

4.1.4 Genetic Operators

In this algorithm, tournament selection has been used for the selection of parents. The details of tournament selection have been explained in section 3.1.1.4.1. Moreover, for the crossover operator Partially Matched Crossover has been used and it has been explained in section 3.1.1.5.3. Lastly, swap mutation and shift mutation have been used as the mutation operators which have been discussed in sections 3.1.1.5.5 and 3.1.1.5.6 respectively.

4.1.5 Local Search Algorithm

In addition to cooperative coevolutionary algorithms, local search algorithm has been used to form a memetic algorithm. In this algorithm, 2-Opt local search algorithm has been used which has been explained in section 3.3.1.1. Furthermore, local search algorithm is applied after cooperation evolution was completed. It is applied to the best individual found by the cooperative coevolutionary algorithm. The further explanations for the general structure of the algorithm are discussed in the next section.

4.1.6 Algorithm Description

This section combines all the algorithms, methods and genetic operators that were discussed in the previous sections. The study of these algorithms, methods and genetic operators helped us to form our algorithm which is called Cooperative Coevolutionary Memetic Algorithm. In this section, the information is explained in detail for our algorithm answering the questions of how these algorithms are combined and how they are applied to TSP. The Figure 4.3 below shows the flowchart of the Cooperative Coevolutionary Memetic Algorithm for TSP.

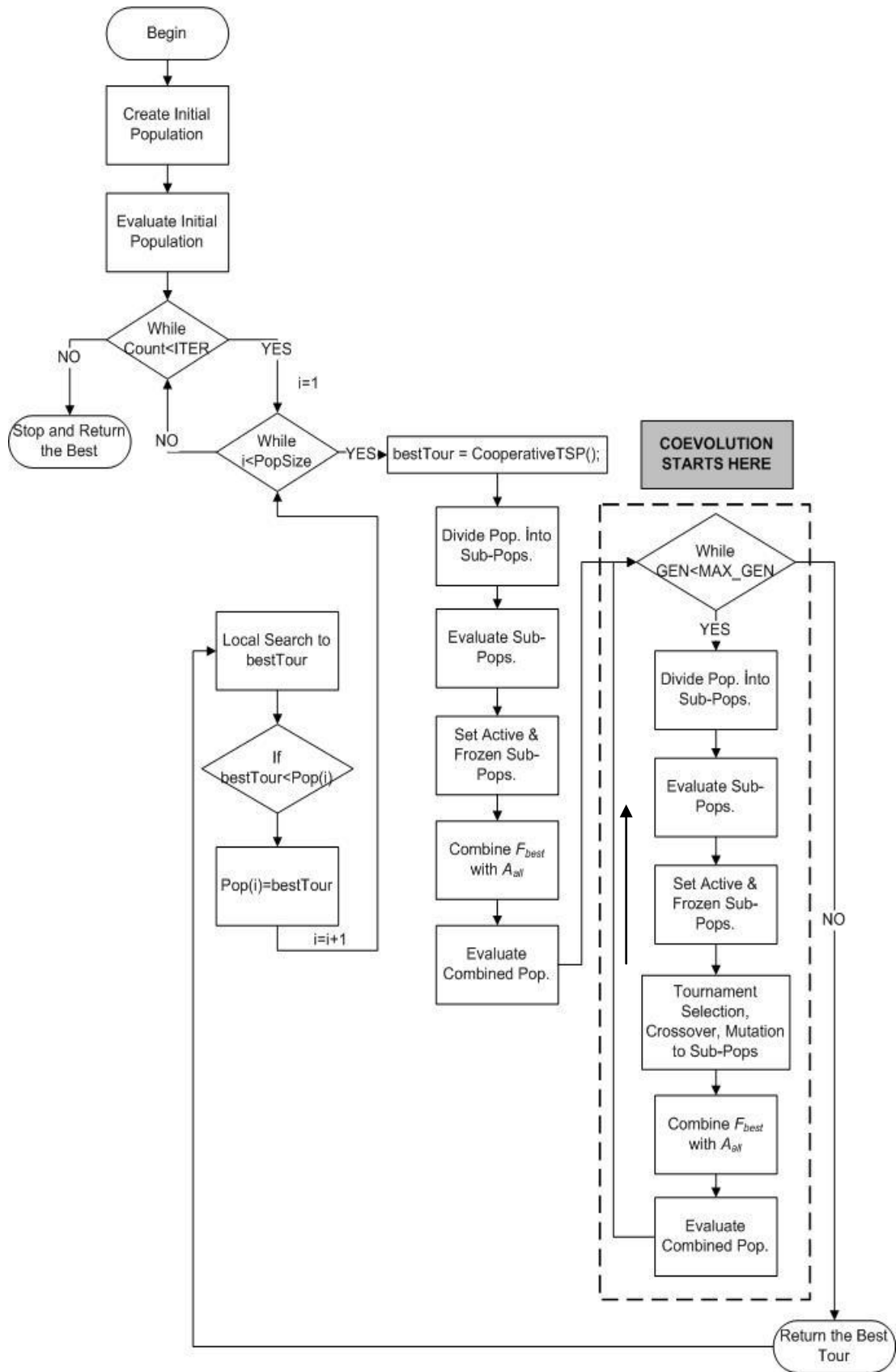


Figure 4.3: Flowchart of the Proposed Algorithm

In this algorithm, we have two different loops as you can see in the Figure 4.3. In the first loop of the algorithm which is outer loop, the initial population is randomly constructed in Figure 4.4. The proposed algorithm is explained by the figures below.

$i=1, N=10;$

$pop(i)$	2	10	5	7	3	4	1	9	8	6	2
	8	3	9	1	7	2	5	4	6	10	8
	⋮					...					
$pop(N)$	10	6	7	4	5	1	9	8	3	2	10

Figure 4.4: Initial Population for First Loop

Then each of the individuals, $pop(i)$, is evaluated by the fitness function for their fitness values. Each individual in the population $pop(i)$ is divided into two sub-populations by the division point and sent to the second loop (inner loop) where cooperation occurs. If we consider the division point as the middle of the chromosome which has 10 cities, then division point would be 5. Division of an individual is shown in Figure 4.5.

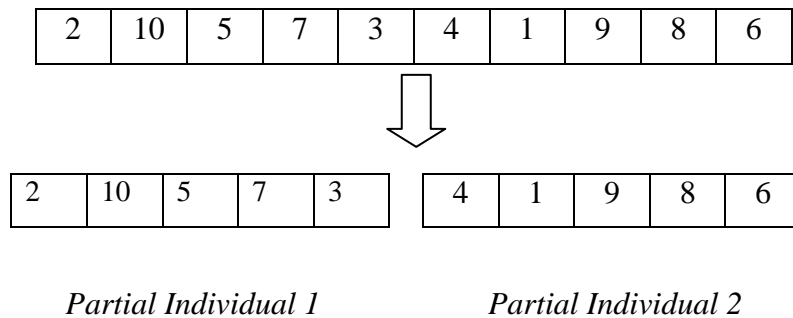


Figure 4.5: Division of an Individual into two Partial Solutions

If we consider the example in Figure 4.5, after dividing the individual into two partial solutions, each partial solution constructs its own population by random permutations of cities that belong to *Partial Individual1* and *Partial Individual2*

without mixing both Partial Individuals' cities. By this way, feasible solutions are constructed for the sub-populations. The population size of sub-populations is set in the cooperation loop. The combination process of sub-populations is shown in Figure 4.6 below.

$k=1, M=100;$

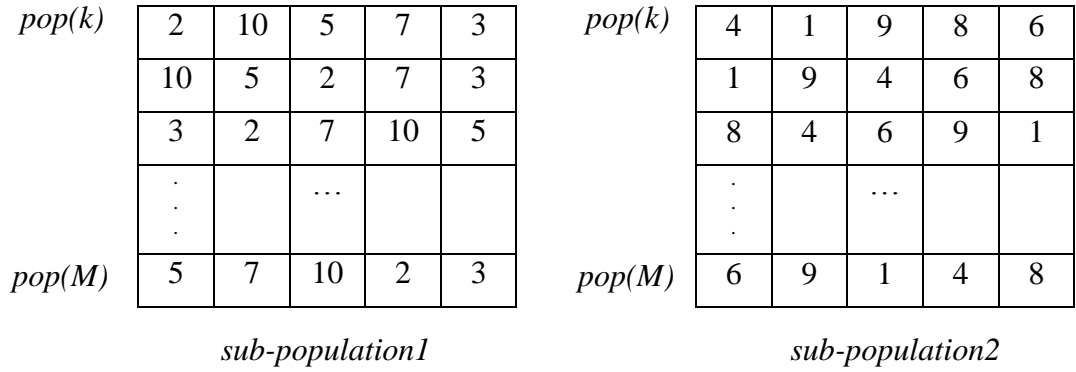


Figure 4.6: Construction of Initial Sub-Populations

After constructing the sub-populations, individuals of each sub-population are evaluated by the fitness function and each sub-population is sorted by their fitness values. After that, active and frozen sub-populations are set for interacting individuals to cooperate and form a complete individual. The cooperation of individuals between sub-populations is made in the following way. The best individual from the frozen sub-population, F_{best} , is matched with all the individuals of the active sub-population, A_{all} . By this way, a population of combined individuals is created.

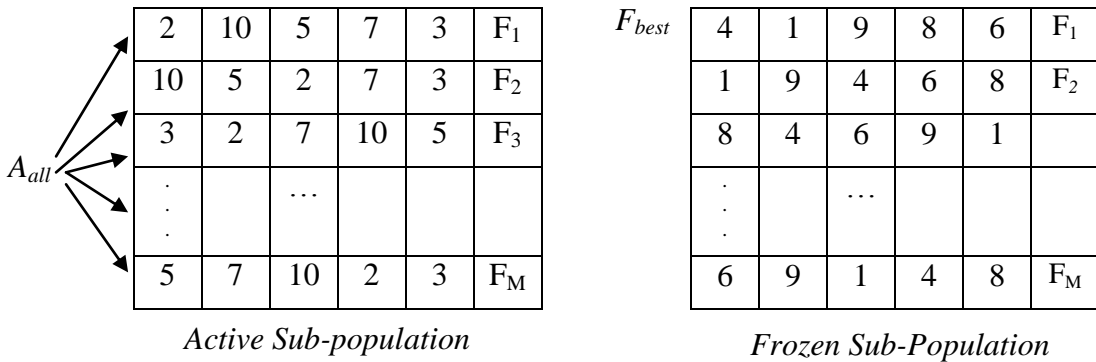


Figure 4.7: Combining Active-Frozen Sub-Populations

The resulting combined population will be as shown in the Figure 4.8 below.

2	10	5	7	3	4	1	9	8	6
10	5	2	7	3	4	1	9	8	6
3	2	7	10	5	4	1	9	8	6
⋮		...			⋮		...		
⋮					⋮				
5	7	10	2	3	4	1	9	8	6

Figure 4.8: Combined Population of Sub-Populations

As it is shown in the Figure 4.8, all individuals from the active sub-population are combined with the best individual from frozen sub-population. Then, a swap mutation is applied to the combined individuals and they are evaluated by the fitness function. These processes are made to construct the initial population for the loop where main cooperation is made between sub-populations. The initial population could be constructed randomly but the reason for constructing the initial population by this way is to start evolution of cooperation by good individuals.

At the beginning of the innermost loop which is generations loop, swap mutation is applied to the combined individuals in every, for example, 5 generations to search for the new complete tours. The innermost loop is where the main cooperation occurs, in each run of the generations one of the sub-population is set as active while the other is set as frozen sub-population. For example, when the *sub-population1* is active and *sub-population2* is frozen in a generation, *sub-population1* will be frozen and *sub-population2* will be active in the next generation. This is where cooperation occurs between individuals of different sub-populations.

In addition, tournament selection is applied to select the parents. Then, crossover operator is applied to the active sub-population and mutation operator to the frozen sub-population. The reason that we used recombination operators in this way is the

following. After some number of generations the frozen sub-population's individuals become the same. For this purpose, to prevent repetition of partial solutions in the frozen sub-population, the shift mutation operator was used. Moreover, the reason why that mutation operator was used instead of a crossover operator in the frozen sub-population is that mutation operator is an exploitation operator which brings the new information and jumps out the boundaries for the solution of the problem. However, with the crossover operator we can only search for the solution in the boundaries.

Furthermore, after applying genetic operators, the sub-populations are combined to form a global complete solution. This combination process of two sub-populations is made in the same way that was explained above in Figure 4.7. Then, the combined population's individuals are evaluated by the fitness function and the best of the individuals, *bestTour*, is sent back to the first loop of the algorithm as the result of cooperation algorithm. Here is where local search is applied to the best solution.

2-Opt or tabu search local search algorithms are applied to the *bestTour* to refine it. Finally, the resulting solution from the local search algorithm is compared with the *pop(i)*, if the local search result is better than the *pop(i)* solution, *pop(i)* is replaced with the *bestTour*. This way of comparing helps to reach better solutions in each generation of the algorithm.

The experimental results of this algorithm are discussed in the next chapter.

Chapter 5

EXPERIMENTAL RESULTS

5.1 TSP Instances

In this chapter, experimental results for TSP are discussed. In this thesis, experiments for the TSP are made by six symmetric TSP datas that are taken from TSPLIB. These symmetric TSP datas that are used for experimenting are berlin52, eil76, st70, kroA100, kroA150, kroA200, kroB100, kroB150 and kroB200. Moreover, the algorithm is also evaluated with Coordina10, Coordina30 and Coordina50 test datas.

The data structures of these symmetric datas are the same. In the first line, the name for the problem is given, for example, kroA100. Second line is the type of the problem which is in this case, tsp and followed by the comment line of the problem. Next line gives the information for the dimension of the problem, in this example it is 100. Then, the edge weight size is given, which is EUC_2D. Lastly, the x and y coordinates of the cities are given. In Coordina test datas the numbers 10, 30 and 50 are the number of cities and only x and y coordinates of the cities are given.

5.2 Experimental Results for TSP

Experimental results are evaluated by some components of the algorithm which are initial population size and iteration size, cooperation population size, and the local search effect on the algorithm. These components are discussed in the following sections.

5.2.1 The Effect of Increased Population Size, Iteration Size

As it was discussed before, the algorithm has initial loop for the initial population construction. Initial loop has population size and iteration size. In the initial loop, we aimed to have different permutations of cities by constructing them randomly. In each iteration of the initial loop, an individual is sent to the cooperation loop and in return a local search is applied and the result is compared with the initial solution that was sent. Therefore, in order to see the effects of population size and iteration size of the algorithm, population size and iteration size are increased in the initial loop.

Furthermore, the algorithm has another loop which is cooperation loop. This is the main algorithm that applies cooperation between sub-populations. In addition, this algorithm has population size and generation size. Here, the population size is for the cooperation population and generation size is the normal generations for the algorithm. Therefore, their sizes are also increased to see the results with different sizes of the algorithm components.

First of all, the algorithm was applied with different parameter values without the local search algorithm for some of the problems. The results without the local search algorithm are shown in tables below.

Table 5.1: Results-1 for the Cooperative Coevolutionary Algorithm without local search algorithm

TEST DATA	InitialPop=10 IterSize=10 PopSize=100 MaxGen=1000	InitialPop=10 IterSize =20 PopSize=100 MaxGen=1000	InitialPop=10 IterSize =30 PopSize=100 MaxGen=1000
kroA100 Opt=21282	46019	86107	46522
kroA150 Opt=26524	68719	66486	66458
kroA200 Opt=29368	86817	84011	81803

Table 5.1 shows the results of Cooperative Coevolutionary Algorithm without local search algorithm. It is applied to three of the test datas. In each row of the table, name of the test data and its optimal solution is given with the variable named as Opt.

Table 5.2: Results-2 for the Cooperative Coevolutionary Algorithm without local search algorithm with increased PopSize

TEST DATA	InitialPop=20 IterSize=10 PopSize=100 MaxGen=1000	InitialPop=20 IterSize =20 PopSize=100 MaxGen=1000	InitialPop=20 IterSize =30 PopSize=100 MaxGen=1000
kroA100 Opt=21282	47485	45929	46585
kroA150 Opt=26524	64903	67636	65530
kroA200 Opt=29368	85418	79523	82919

In Table 5.2, initial population size is increased for each test data. It is aimed to see how initial population size affects the algorithm results. In comparison with the results of Table 5.1 and Table 5.2, increasing initial population size resulted in better solutions.

5.2.2 The Effect of Local Search Algorithm

In combinatorial optimization problems, the need for local search methods is observed by the results shown in the above tables in section 5.2.1. In our algorithm, the best found individual from the cooperative coevolutionary algorithm became better by applying local search algorithm which was 2-Opt local search or tabu search algorithm. Because, the local search makes small moves around the given solution and tries to find the better version of the given solution. Therefore, it is observed that without the local search algorithms in any of the combinatorial optimization problems, it is not easy to find a good solution by an EA itself. As a

result, our solution approach to TSP proves the need for the local search algorithms in combinatorial optimization. The tables below show the results that are found by cooperative coevolutionary algorithm combined with the local search algorithm which is 2-Opt.

Table 5.3: Results-1 for the Cooperative Coevolutionary Algorithm with 2-Opt local search algorithm with small test datas

TEST DATA	InitialPop=10 IterSize=10 PopSize=100 MaxGen=1000	InitialPop=10 IterSize =20 PopSize=100 MaxGen=1000	InitialPop=10 IterSize =30 PopSize=100 MaxGen=1000
Coordina10 Opt=27.8929	27.8929	27.8929	27.8929
Coordina30 Opt=46.6474	46.6474	46.6474	46.6474
Coordina50 Opt=58.1261	58.4555	58.2331	58.2280

Table 5.4: Results-2 for the Cooperative Coevolutionary Algorithm with 2-Opt local search algorithm with increased initial population size in small test datas

TEST DATA	InitialPop=20 IterSize=10 PopSize=100 MaxGen=1000	InitialPop=20 IterSize =20 PopSize=100 MaxGen=1000	InitialPop=20 IterSize =30 PopSize=100 MaxGen=1000
Coordina10 Opt=27.8929	27.8929	27.8929	27.8929
Coordina30 Opt=46.6474	46.6474	46.6474	46.6474
Coordina50 Opt=58.1261	58.1261	58.1261	58.1261

Table 5.3 and Table 5.4 show the results for the Cooperative Coevolutionary Algorithm with 2-Opt local search algorithm. In Table 5.3 only the iteration size is increased by 10. As the iteration size was increased, the algorithm found the optimal solutions or got nearly optimal solutions for some of the test datas. In Table 5.4 only

the initial population size and the iteration sizes are increased. The results for Coordina50 got better than the results in Table 5.3.

Table 5.5: Results-3 for Cooperative Coevolutionary Algorithm with 2-Opt local search algorithm

TEST DATA	InitialPop=10 IterSize=10 PopSize=100 MaxGen=1000	InitialPop=10 IterSize =20 PopSize=100 MaxGen=1000	InitialPop=10 IterSize =30 PopSize=100 MaxGen=1000
berlin52 opt=7542	7578	7578	7578
eil76 opt=538	614	613	613
st70 opt=675	724	730	723

Table 5.6: Results-4 for Cooperative Coevolutionary Algorithm including 2-Opt with increased initial population size

TEST DATA	InitialPop=20 IterSize=10 PopSize=100 MaxGen=1000	InitialPop=20 IterSize =20 PopSize=100 MaxGen=1000	InitialPop=20 IterSize =30 PopSize=100 MaxGen=1000
berlin52 opt=7542	7578	7578	7578
eil76 opt=538	616	608	610
st70 opt=675	723	722	722

The proposed algorithm along with the 2-Opt local search algorithm is firstly tested by small test datas which their results are shown in Table 5.3, Table 5.4, Table 5.5 and Table 5.6. These results from each table are near to the optimal solutions. Next table shows the results from the bigger test datas.

Table 5.7: Results-5 for Cooperative Coevolutionary Algorithm with 2-Opt local search algorithm

TEST DATA	InitialPop=10; IterSize=10; PopSize=100; MaxGen=100;
kroA100 Opt=21282	21360
kroA150 Opt=26524	27077
kroA200 Opt=29368	30444
kroB100 Opt=22141	22413
kroB150 Opt=26130	26794
kroB200 Opt=29437	30487

Table 5.7 shows the evaluation results of bigger test datas that are applied to the proposed algorithm with 2-Opt local search algorithm. The datas are evaluated with 100 maximum generation size. The proposed algorithm with 2-Opt local search algorithm found the nearest optimal solutions even with the small number of generation size. These results show the effect of a local search algorithm.

Table 5.8: Results-6 for Cooperative Coevolutionary Algorithm with 2-Opt local search algorithm

TEST DATA	InitialPop=10 IterSize=10 PopSize=100 MaxGen=200	InitialPop=10 IterSize=20 PopSize=100 MaxGen=200	InitialPop=10 IterSize=30 PopSize=100 MaxGen=200
kroA100 Opt=21282	21554	21375	21404
kroA150 Opt=26524	27124	27283	27124
kroA200 Opt=29368	30643	30261	30312
kroB100 Opt=22141	22491	22487	22319
kroB150 Opt=26130	26808	26887	26771
kroB200 Opt=29437	30487	30897	30748

Table 5.8 shows the results that are found by increasing only the maximum generation size which is set to 200.

Table 5.9: Results-7 for Cooperative Coevolutionary Algorithm with 2-Opt local search algorithm

TEST DATA	InitialPop=10 IterSize=10 PopSize=100 MaxGen=600	InitialPop=10 IterSize =20 PopSize=100 MaxGen=600	InitialPop=10 IterSize =30 PopSize=100 MaxGen=600
kroA100 Opt=21282	21513	21455	21449
kroA150 Opt=26524	27075	27051	27221
kroA200 Opt=29368	30770	30750	30248
kroB100 Opt=22141	22713	22354	22568
kroB150 Opt=26130	27057	26836	26817
kroB200 Opt=29437	31172	27018	30720

Table 5.10: Results-8 for Cooperative Coevolutionary Algorithm with 2-Opt local search algorithm

TEST DATA	InitialPop=10 IterSize=10 PopSize=100 MaxGen=1000	InitialPop=10 IterSize =20 PopSize=100 MaxGen=1000	InitialPop=10 IterSize =30 PopSize=100 MaxGen=1000
kroA100 Opt=21282	21537	21630	21355
kroA150 Opt=26524	27104	27306	27027
kroA200 Opt=29368	30701	30005	30202
kroB100 Opt=22141	22716	22398	22283
kroB150 Opt=26130	26785	26566	26696
kroB200 Opt=29437	30590	30264	30489

In Table 5.9 and Table 5.10, only the maximum generation sizes are increased to 600 and 1000 respectively. Increasing maximum generation sizes let the algorithm to find better results in comparison with previous results of the other tables.

In the above tables, the results for the different parameters with different values are examined. It is observed that when the iteration size of the initial population and maximum generation size are increased, the algorithm gives better solutions compared to the other results that the iteration size and maximum generation size are small. The following table shows the results that are found by increasing the population size of the initial loop. In the previous tables many of the good results were found when the maximum generation size was 1000. That is why; increased initial population size is only tested with maximum generation size of 1000.

Table 5.11: Results-9 for Cooperative Coevolutionary Algorithm including 2-Opt with increased initial population size

TEST DATA	InitialPop=20 IterSize=10 PopSize=100 MaxGen=1000	InitialPop=20 IterSize =20 PopSize=100 MaxGen=1000	InitialPop=20 IterSize =30 PopSize=100 MaxGen=1000
kroA100 Opt=21282	21448	21375	21449
kroA150 Opt=26524	27372	27041	27051
kroA200 Opt=29368	30131	30097	29895
kroB100 Opt=22141	22502	22326	22226
kroB150 Opt=26130	26668	26628	26444
kroB200 Opt=29437	30567	30564	30281

In Table 5.11, only the initial population size is increased to 20 and the other parameters are set as the same. Increased population size makes the solution space of

the problem bigger. Therefore, possibility of finding the optimal or nearly optimal solutions becomes higher.

The previous results showed us the effect of 2-Opt local search algorithm along with the cooperative coevolutionary algorithm to TSP with increased population size, iteration size and generation size. In the next table, the effect of tabu search algorithm along with the cooperative coevolutionary algorithm is shown.

Table 5.12: Results-1 for Cooperative Coevolutionary Algorithm with Tabu Search

TEST DATA	InitialPop=10 IterSize=10 PopSize=100 MaxGen=1000	InitialPop=10 IterSize =20 PopSize=100 MaxGen=1000	InitialPop=10 IterSize =30 PopSize=100 MaxGen=1000
kroA100 Opt=21282	29888	32302	30810
kroA150 Opt=26524	40340	44360	40661
kroA200 Opt=29368	54891	53584	54264

Besides 2-Opt local search algorithm, the proposed algorithm is evaluated with Tabu search algorithm. The results for Tabu search algorithm are shown in Table 5.12. Tabu search algorithm can also reach the optimal or nearest optimal solutions but 2-Opt local search algorithm reached better solutions compared with Tabu search algorithm. However, the results in Table 5.12 can be improved by increasing the iteration size and maximum generation size of the test datas.

Chapter 6

CONCLUSION

In this thesis, the implementation and results for our solution approach to TSP are presented by using Cooperative Coevolutionary Memetic Algorithms. Our solution approach is mainly based on the population-based algorithms along with the local search algorithms where combination of these algorithms represents the idea of the memetic algorithms. Recently, there are many algorithms that use population-based and local search algorithms to find better solutions for the problems. Furthermore, from the experimental results we can conclude that memetic algorithms are successful algorithms on the combinatorial optimization problems. Also, they can find the optimal solution by increasing the number of iterations for the problem.

Furthermore, in this thesis, cooperative coevolutionary algorithms are examined by interactions between two sub-populations which were active and frozen sub-populations. The future work related to cooperative evolution would be done by applying two active sub-populations instead of active-frozen sub-populations. In addition to this, an archive for positions of cities, which shortens the tour length, can be formed. This archive could store the positions of cities in order to improve finding shorter tours for TSP.

REFERENCES

- [1] M. A. Potter, K. A. De Jong, “Cooperative Coevolution: An Architecture for Evolving Coadapted Subcomponents”, *Massachusetts Institute of Technology*, Vol. 8, No. 1, 1-29, 2000.
- [2] Duan, H. & Yu, X., “Hybrid Ant Colony Optimization Using Memetic Algorithm for Traveling Salesman Problem”, *Proceedings of the 2007 IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning*, 2007.
- [3] Omidvar, M. N., & Li, X. Cooperative Coevolutionary Algorithms for Large Scale Optimization.
- [4] Toma, N., & Endo, S., & Yamada, K. An Immune Co-evolutionary Algorithm for N-th Agent’s Traveling Salesman Problem.
- [5] Garcia-Pedrajas, N., Romero del Castillo, J. A., Ortiz-Boyer, D., “A Cooperative Coevolutionary Algorithm for Instance Selection for Instance-Based Learning”, edited by Risto Miikkulainen, 381-420. Springer, 2010.
- [6] A. Schrijver, ”On the history of combinatorial optimization (till 1960)”, The University of Amsterdam.

- [7] Z. C. S. S. Hlaing, M. A. Khine, “Solving Traveling Salesman Problem by Using Improved Ant Colony Optimization Algorithm”, *International Journal of Information and Education Technology*, Vol. 1, No. 5, pp. 404-409, December 2011.
- [8] M. J. Haque, K. W. Magld, “Improving the Solution of Traveling Salesman Problem Using Genetic, Memetic Algorithm and Edge Assembly Crossover”, *International Journal of Advanced Computer Science and Applications*, Vol. 3, No. 7, 108-111, 2012.
- [9] “What is an Evolutionary Algorithm?” Last modified February 2, 2004, <http://www.cs.vu.nl/~gusz/ecbook/Eiben-Smith-Intro2EC-Ch2.pdf>.
- [10] Rou, L. Y., & Asmuni, H., “A Study of Cooperative Co-evolutionary Genetic Algorithm for Solving Flexible Job Shop Scheduling Problem”, *World Academy of Science, Engineering and Technology* 72,2010.
- [11] William E. Hart, N. Krasnogor, J. E. Smith, *Recent Advances in Memetic Algorithms*, (New York: Springer, 2005), 3-22.
- [12] F. Seredynski, A. Y. Zomaya and P. Bouvry, “Function Optimization with Coevolutionary Algorithms”
- [13] “Tabu Search”, Last modified December 17, 2008, <http://www.inf.ufpr.br/aurora/disciplinas/topicosia2/livros/search/TS.pdf>

- [14] TSPLIB, Last modified, August 6, 2008,
<http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/>
- [15] M. Nojournian, D. K. Nair, “Comparing Genetic Algorithm and Guided Local Search Methods by Symmetric TSP Instances”.
- [16] D. S. Johnson, L. A. McGeoch, “The Traveling Salesman Problem: A Case Study in Local Optimization”, November, 1995.
- [17] Adewole A.P., Otobamowo K., Egunjobi T. O., “A Comparative Study of Simulated Annealing and Genetic Algorithm for Solving the Traveling Salesman Problem”, *International Journal of Applied Information Systems*, Vol. 4, No. 4, pp. 6-12, October, 2012.
- [18] B. Freisleben, P. Merz, “A Genetic Local Search Algorithm for Solving Symmetric and Asymmetric Traveling Salesman Problems”.
- [19] M. Shaikh, M. Panchal, “Solving Asymmetric Traveling Salesman Problem using Memetic Algorithm”, *International Journal of Emerging Technology and Advanced Engineering*, Vol.2, Issue 11, November, 2012.
- [20] S. G. Yaseen, N. M. A. Al-Slamy, “Ant Colony Optimization”, *International Journal of Computer Science and Network Security*, Vol. 8, No. 6, pp. 351-357, June, 2008.

[21] S. Goyal, “A Survey on Traveling Salesman Problem”, *University of North Dakota*.

[22] M. Shi, “An Exploration and Optimization of Cooperative Coevolution”, *Norwegian University of Science and Technology*, December, 2012.