AN EVOLUTIONARY APPROACH TO THE TRAVELING SALESMAN PROBLEM WITH PICKUP AND DELIVERY BASED ON DEPOT INSERTION AND REMOVAL MOVES

(TOPLAMALI DAĞITIMLI GEZGİN SATICI PROBLEMİ İÇİN DEPO
YERLEŞTİRME VE ÇIKARMA TABANLI
BİR SEZGİSEL ALGORİTMA)

by

Volkan ÇINAR B.S.

Thesis

Submitted in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE

in

INDUSTRIAL ENGINEERING

in the

INSTITUTE OF SCIENCE AND ENGINEERING

of

GALATASARAY UNIVERSITY

January 2010

AN EVOLUTIONARY APPROACH TO THE TRAVELING SALESMAN PROBLEM WITH PICKUP AND DELIVERY BASED ON DEPOT INSERTION AND REMOVAL MOVES

(TOPLAMALI DAĞITIMLI GEZGİN SATICI PROBLEMİ İÇİN DEPO
YERLEŞTİRME VE ÇIKARMA TABANLI
BİR SEZGİSEL ALGORİTMA)

by

Volkan ÇINAR B.S.

Thesis

Submitted in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE

Date of Submission : December 31, 2009

Date of Defense Examination : January 27, 2010

ACKNOWLEDGEMENTS

I wish to express my profound gratitude to Assoc. Prof. Temel ÖNCAN. He encouraged me and never gave up providing me necessary guidance that allowed me to finish this dissertation.

I also wish to thank my family for their encouragement, enthusiasm and emotional support.

Volkan ÇINAR
January 27, 2010

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

The Traveling Salesman Problem with Pickup and Delivery (TSPPD) is an extension of the famous Traveling Salesman Problem (TSP). In the literature there are many problems related to pickup and delivery but they have different features (i.e. single commodity or multi commodities, single vehicle or multi vehicles, one-to-one or one-to-many etc.). TSPPD is a static simultaneous pickup delivery problem with a “one-to-many-to-one” structure where there is only one vehicle and that each customer is visited exactly once. The problem involves two sets of customers. “Delivery Customers” are served by delivery of the goods from a central warehouse. “Pickup Customers” need to deliver goods from their locations to the central warehouse. We call the central warehouse as “Depot” and all delivery and pickup services are done by a single vehicle with a given capacity which is equal to Q. Given two types of goods (delivery and pickup goods), a vehicle which departs from the depot with fully loaded satisfies all the customer’s needs and returns back to the depot. The objective of the TSPPD is to minimize the total travelling distance. The additional constraint of the TSPPD is that the vehicle load must be feasible throughout the tour; it must remain nonnegative and the total load should not exceed the vehicle capacity. TSPPD is an NP-hard combinatorial optimization problem since it as an extension of the well known TSP. To the best of our knowledge there is not many publications on the TSPPD. The motivation of this thesis is to propose an efficient Genetic Algorithm (GA) to solve TSPPD instances.

GAs are search algorithms based on the mechanics of natural selection and natural genetics. The basic principle of the GAs comes from “survival of the fittest” which is stated by Charles Darwin in “The Origins of Species”. They are adaptive methods which may be used to solve search and optimization problems. The power of GA comes from the fact that the method is robust and which can be used successfully for a wide range of problem areas including those which are difficult for other methods to solve.

The motivation behind of the proposed GA, lies in the Mosheiov Theorem. Mosheiov has proved that when we construct a Hamiltonian tour covering all pickup and delivery customers without the depot, there exists at least one point i_k on this tour such that when we insert the depot between i_k and i_{k+1} the resulting tour is feasible. Considering this theorem we have devised an efficient algorithm which consists of three steps. The first step is to solve Hamiltonian tour through all customers by using the GA. The second step is to find the best starting point to insert the depot. The final step is to apply a local neighborhood search which we term as "Tuning Phase". Computational experiments are reported on standard test instances from the literature. The experimental results show that our algorithm yields promising performance in terms of both accuracy and efficiency compared to existing algorithms in the literature. We should also report that the tuning phase step considerably improves the solution obtained after the second step. This shows us the power of our tour improvement procedure, which is specially designed for the TSPPD.

RESUME

Le Problème de Voyager-Vendeur avec ramassage et de livraison (TSPPD) est une extension du célèbre problème Voyageur-Vendeur. (TSP). Dans la littérature il existe de nombreux problèmes liés à la ramassage et la livraison, mais ces problèmes ont des caractéristiques différentes (Unique Commodité ou Multi Commodités, Unique véhicule ou plusieurs véhicules, une-à-une ou une-à-plusieurs).Le problème examinée dans cette thèse est un problème de Voyager-Vendeur avec ramassage et de livraison qui est statique et simultanée avec une « une-à-une » structure où il n'y a qu'un seul véhicule, et que chaque clients ont visité exactement une fois. Le problème comporte deux types de clients. "Les clients de livraison" sont servis par la livraison des marchandises dans un entrepôt central et «Les clients ramassage" nécessitent de fournir ses biens de leur location à l'entrepôt central. Nous appelons l'entrepôt central le "Depot" et tous les services de livraison et de ramassage sont fournis par un seul véhicule d'une capacité donnée, qui est égal à Q. Etant deux types de produits différents, un véhicule départ du dépôt, satisfait tous les besoins des clients et retourne au dépôt. L'objectif de TSPPD est de minimiser la distance totale du parcourus. La contrainte supplémentaire du TSPPD est que la charge du véhicule doit être faisable tout au long du tour. Elle doit rester positif et que la charge totale ne doit pas dépasser la capacité du véhicule. TSPPD est un problème NP-hard d'optimisation combinatoire comme TSP. Au meilleur de notre connaissance, il n'existe pas beaucoup de publications concernant le TSPPD dans la littérature. L'objectif de cette thèse est de proposer une Algorithme Génétique (GA) efficace pour résoudre le TSPPD.

Des GAs sont des algorithmes de recherche basés sur la mécanique de la sélection naturelle et de la génétique naturelle. Le principe de base du GA provient de «survie des meilleurs» qui est énoncée par Charles Darwin dans "The Origins of Species". Ils sont des méthodes adaptatives qui peuvent être utilisé pour résoudre les problèmes de recherche et d'optimisation. La force des GA vient du fait que la méthode est robuste et qui peut être utilisé avec succès pour un large éventail de problèmes, y compris celles qui sont difficiles à résoudre avec d'autres méthodes

L'idée de base de notre approche évolutive vient du théorème Mosheiov. Mosheiov a prouvé que si nous construisons une tournée Hamiltonien couvrant l'ensemble des clients sans le dépôt, il existe au moins un point i_k sur ce tour de telle sorte que si l'on insère le dépôt entre i_k et i_{k+1} le tour reste toujours faisable. Selon ce théorème, nous avons développé un algorithme génétique efficace qui consiste en trois étapes. La première étape est de résoudre le tour Hamiltonien en utilisant le GA. La deuxième étape est de trouver le meilleur point de départ pour insérer le dépôt. Enfin la dernière étape consiste à appliquer une recherche locale qu'on le nomme "Post optimalité". Les résultats expérimentaux montrent que notre algorithme évolutionnaire une performance prometteuse en termes d'exactitude et d'efficacité par rapport aux algorithmes existants dans la littérature. Nous devons aussi signaler que l'utilisation de l'étape « post optimalité » améliore nettement la solution obtenue après la deuxième étape. Cela nous montre la puissance de notre procédure d'amélioration du tour, qui est spécialement désigné pour le TSPPD.

ÖZET

Gezgin Satıcı Problemi (GSP) bir şehirden başlayıp, listedeki tüm şehirleri sadece bir kez ziyaret edip, tekrar başladığı şehre dönen bir satıcı için en kısa turun belirlenmesi problemidir. Toplamalı Dağıtımlı Gezgin Satıcı Problemi (TDGSP) de GSP'nin farklı bir türüdür. Literatürde toplamalı dağıtımlı problemlerin bir çok farklı karakteristikteki çeşitleri mevcuttur. (Ör: Tek ürünlü-çok ürünlü, tek araçlı – çok araçlı, tekten-teke, çoktan-çoka, vs.). Bu çalışmada incelenen problem, statik, eşzamanlı, tek araç ile yapılan ve bütün şehirlerin tek bir seferde ziyaret edildiği dağıtım ve toplama problemidir. Problem iki çeşit müşteriyi barındırır. Dağıtım yapılan müşteriler depodan mal talep müşterilerdir. Toplama yapılan müşteriler ise depoya mal gönderen müşterilerdir. Bütün toplama ve dağıtım işlemleri, sığası Q değerine eşit olan bir araç vasıtasıyla gerçekleştirilir. Bu araç tam yüklü olarak depodan çıkar ve bütün müşteri ihtiyaçlarına cevap vererek tekrar depoya geri döner. TDGSP'nin en önemli ilave kısıtı araç yükünün tur boyunca olurlu olması gerektiğidir. Araç yükü tur boyunca negatif olmamalı ve yük araç sığasını geçmemelidir. TDGSP, GSP gibi NP-zor problemler sınıfında yer almaktadır. Yapmış olduğumuz yazın taramalarına göre TDGSP hakkında çok sayıda yayın bulunmamaktadır. Bu tezin amacı bu boşluğu doldurmak ve TDGSP'nin çözümü için etkin bir Genetik Algoritma (GA) geliştirmektir.

GA'lar temelinde doğal seçimi esas alan arama algoritmalarıdır ve doğadaki evrim sürecini taklit etmeye dayalı sezgisel teknikleri barındırırlar. GA'nın temel prensibi Charles Darwin'in "Türlerin Kökeni" kitabında tanımladığı "güçlü olanın hayatta kalması" ilkesine dayanır. GA'lar diğer metotlar ile çözülemeyecek bir çok farklı alandaki problemin çözümünde fayda sağlamaktadır.

Bu tezde geliştirilen GA'nın temel prensibi Mosheiov'un TDGSP için ispatlamış olduğu teoreme dayanmaktadır. Mosheiov şu sonucu ispatlamıştır: "Eğer depo hariç diğer müşterilerin hepsini kapsayan bir Hamilton turu oluşturulursa bu tur içerisinde en az bir i_k noktası vardır ki; depo, i_k ve i_{k+1} arasına yerleştirilirse yeni oluşan Hamilton turu TSPPD için olurlu olur. Bu teoreme dayanarak bu tezde üç aşamalı bir algoritma geliştirilmiştir. Algoritmanın birinci aşamasında depo hariç diğer bütün müşterileri içinde barındıran bir Hamilton turu GA kullanılarak oluşturulur. İkinci aşamada depo, en optimum olurlu noktadan tura yerleştirilir. Son olarak ise "iyileştirme aşaması" olarak adlandırdığımız bir yerel arama metodu kullanılır. Geliştirilen algoritma literatürdeki mevcut TDGSP örnekleri ile test edilmiştir. Yapılan denemelerde gerek hız gerek çözüm iyiliği açısından iyi sonuçlar elde edilmiştir. Bununla birlikte algoritmanın 3. aşamasında kullanılan ve TDGSP'ye özel olarak geliştirilmiş iyileştirme aşamasının 2. aşamadan çıkan sonuçlar üzerinde hatırı sayılır iyileştirmelere sebep olduğu gözlemlenmiştir.

1 INTRODUCTION

1.1 The Traveling Salesman Problem

The Traveling Salesman Problem (TSP) is a well known NP-hard combinatorial optimization problem in which a salesman who starts from a home location visits N cities under the condition that each city is visited exactly once [1]. The objective of the TSP is to find a Hamiltonian tour. A Hamiltonian tour is a tour which visits all vertices exactly one time.

The TSP is one of the hardest of the Operation Research. The precise origins of TSP are unclear but to the best of our knowledge its history started with Euler as early as 1759, who has interested in solving the Knight's Tour Problem [2]. In 1832, a handbook was published for German travelling salesmen, which included examples of tours. In 1850s, Sir William Rowan Hamilton studied Hamiltonian circuits in graphs but the first use of the terms "Traveling Salesman Problem" in mathematical circles may date back to 1931-1932 [3]. The TSP has been studied in the first half of the twentieth century in the agricultural context [4]. An integer programming based solution approach for the TSP was provided by Dantzig et al. in 1954 [5]. The authors have solved a TSP instance with 49 cities to optimality. Within the last few decades researchers have offered algorithms to generate approximate solutions [2]. One of the first heuristics has been devised by Lin [6]. Ever since several heuristic and metaheuristic approaches have been proposed. Several real-world applications of the TSP arise in logistics, scheduling, production planning, etc… [1]. Several mathematical programming models have been proposed for TSP. Recently, an analytical comparison of these formulations have been discussed by Öncan et al [7]. One of the best known formulations of the TSP is designed by Dantzig et al. [5].

Given a finite set of points $\mathbf{V} = \{\mathbf{1}, \mathbf{2}, \mathbf{3} \ldots \mathbf{n}\}$ and a cost matrix $C = (c_{ij})_{nxn}$ (also referred to as the distance matrix or weight matrix) defined between each vertex i and $j \in V$, let $G = (V, E)$ be a graph and F be the family of all Hamiltonian tours in G. For each edge $e \in E$ a cost c_e is prescribed. Then the TSP is to find a tour in G such that the sum of the edge costs forming the tour is as small as possible [1]. According to the cost matrix TSPs can be divided in two categories;

- Symmetric TSP (STPS) with symmetric cost matrix; $\mathrm{c_{ij}} = \mathrm{c_{ji}}\ \forall\, \mathrm{i}, \mathrm{j} \in \mathrm{V}, \text{for } i \neq j$
- Asymmetric TSP (ATSP) with asymmetric cost matrix; $\mathrm{c_{ij}} \neq \mathrm{c_{ji}}\ \forall\, \mathrm{i}, \mathrm{j} \in \mathrm{V},\ \text{for } i \neq j$

An integer programming formulation of the ATSP can be formulated as follows [8];

$$Min\ Z = \sum_{\forall i,j\ i \neq j} d_{ij} X_{ij} \tag{1.1}$$

Subject to

$$\sum_{j=1\ j \neq i}^{n} X_{ij} = 1 \qquad i = 1, \ldots., n \tag{1.2}$$

$$\sum_{i=1\ i \neq j}^{n} X_{ij} = 1 \qquad i = 1, \ldots., n \tag{1.3}$$

$$\sum_{i,j \in S}^{n} X_{ij} \leq |S| - 1\,, \quad \forall\, S \subset N; \quad 2 \leq |S| \leq n - 2 \tag{1.4}$$

$$X_{ij} = \{1, 0\} \qquad \forall\, i, j = 1, \ldots., n; i \neq j \tag{1.5}$$

Where $X_{ij} = 1$, when the salesman visits city j immediately after city i. Constraints (1.2) and (1.3) ensure that each costumer is visited exactly once. They are the assignment constraints. Constraint (1.4) obviates subtours.

The TSP can also be viewed as a permutation problem. Any single permutation of n cities yields a solution in the search space with a size N! [2]; Given a set of cities *{ k_1, k_2,... k_N }* and assume that for each pair of distinct cities a distance of *d(k_i, k_j)* is defined. The aim of the TSP is to find an ordering π which minimize the total tour length; [9]

$$Min\, Z = \sum_{i,j \in S}^{n} d(c_{\pi(i)}, c_{\pi(i+1)}), \; + \; d(c_{\pi(N)}, c_{\pi(1)}) \tag{1.6}$$

As an illustrative example let us consider four cities in Turkey: Bursa, İstanbul, Ankara and Trabzon. Then we have;

$$\mathbf{V} = \{\mathbf{Bursa}, \mathbf{Trabzon}, \mathbf{Ankara}, \mathbf{İstanbul}\}.$$

The TSP is defined as follow; "In which order should the salesman visit these cities in order to minimize the total distance according to the cost matrix C given below?

$$C = \begin{bmatrix} 0 & 900 & 350 & 220 \\ 900 & 0 & 620 & 1100 \\ 350 & 620 & 0 & 450 \\ 220 & 1100 & 450 & 0 \end{bmatrix} \tag{1.7}$$

Note that when the cost matrix is symmetric there are (N-1)! feasible TSP tours. Considering all possible permutations clearly there are 6 possible tours. Clearly speaking they are;

$$\text{S1} = \text{Bursa} \rightarrow \text{İstanbul} \rightarrow \text{Ankara} \rightarrow \text{Trabzon} \rightarrow \text{Bursa}$$
$$\text{S2} = \text{Bursa} \rightarrow \text{İstanbul} \rightarrow \text{Trabzon} \rightarrow \text{Ankara} \rightarrow \text{Bursa}$$
$$\text{S3} = \text{Bursa} \rightarrow \text{Ankara} \rightarrow \text{İstanbul} \rightarrow \text{Trabzon} \rightarrow \text{Bursa}$$
$$\text{S4} = \text{Bursa} \rightarrow \text{Trabzon} \rightarrow \text{Ankara} \rightarrow \text{İstanbul} \rightarrow \text{Bursa}$$

$$\text{S5} = \text{Bursa} \rightarrow \text{Trabzon} \rightarrow \text{İstanbul} \rightarrow \text{Ankara} \rightarrow \text{Bursa}$$

$$\text{S6} = \text{Bursa} \rightarrow \text{Ankara} \rightarrow \text{Trabzon} \rightarrow \text{İstanbul} \rightarrow \text{Bursa}$$

When we calculate the total tour lengths of these 6 tours, we obtain the following vector;

$$L = \begin{pmatrix} \mathcal{L}(S1) \\ \mathcal{L}(S2) \\ \mathcal{L}(S3) \\ \mathcal{L}(S4) \\ \mathcal{L}(S5) \\ \mathcal{L}(S6) \end{pmatrix} = \begin{pmatrix} 2190 \\ 2290 \\ 2800 \\ 2190 \\ 2800 \\ 2290 \end{pmatrix} \tag{1.8}$$

Where *L(s)* denotes the length of the tour S. Since the cost matrix is symmetric we have two optimum solutions. **S** = {**S1, S4**}. Note that when N is small enough it's possible to find the optimal solution by enumeration. However when N is large the need to use an ingenious approach becomes obvious. To better expose this we give in Table 1.1 the computations time required to find the optimal solution of different TSP instances on a standard computer.

1.2 TSP Variants

With the simple transformations it's possible to formulate several variants of the TSP;

- **The MAX TSP:** The objective for this TSP variants is to find a tour in G where the total tour length is maximum [1].

- **The bottleneck TSP:** The objective of this problem is to find a tour in G such that the largest distance of edges in the tour is as small as possible [1].

- **TSP with multiple visits:** In this variant, a salesman starts from a node, visits each node at least once and returns back to the starting node. Objective of this problem is to find a tour with minimum total distance. [1]

- **Clustered TSP:** In this problem there are clusters $V_1, V_{2,\ldots,}V_k$ in G The constraint of this problem is that each city in the cluster must be consecutively visited. For example if the salesman enters into a city which belongs to cluster V_k, then he must visit all the cities belonging to cluster located inside the cluster V_k.

Table 1.1: Computationals time required to solve TSP instances.

Number of cities	**Required time in sec.**	**Required time in min.**	**Required time in hours**	**Required time in days**
15	8,7	0,1	0,002	0,0001
16	130,8	2,2	0,04	0,002
17	2092,3	34,9	0,58	0,024
18	35568,7	592,8	9,88	0,412
19	640237,4	10670,6	177,84	7,410
20	12164510,0	202741,8	3379,03	140,793

- **Generalized TSP (GTSP):** In the Generalized TSP, given clusters $V_1, V_{2,\ldots}, V_k$., the objective is to find a shortest tour which passes through exactly one city from each cluster.

There are many other extensions and variants of the TSP which we cannot enumerate all of them here for the sake of conciseness. Some of them are Time Dependent TSP, Period TSP, Black and White TSP, Angle TSP, The Selective TSP, Resource Constraint TSP, Serdyukov TSP, Ordered Cluster TSP, Precedence Constrained TSP, k-Peripatetic Salesman Problem, Covering Salesman Problem, Stochastic TSP, TSP with Time Windows, Moving Target TSP, Remote TSP [1, 3, 10, 11].

1.3 Real life applications of the TSP

There are many practical real-life applications of the TSP. Furthermore several other combinatorial optimization problems can be considered as a generalization or restriction of the TSP. Some of these extensions are as follows;

- **Vehicle Routing Problem:** Some instances of the Vehicle Routing Problem (VRP) can be modeled as a TSP. The VRP is to find which customers should be served by which vehicles and the minimum number of vehicles needed to serve each customer. There are several variations of the VRP [12].

- **Minimum Spanning Tree Problem:** A spanning tree of a graph is a tree connecting all vertices. The objective is to find a spanning tree of minimum total length. This problem is a relaxation of the TSP and the TSP is a restriction of the minimum spanning tree problem [10].

- **Computer Wiring Problem:** We have several modules each with a number of pins. We need to connect a subset of these pins with wires in such a way that no pin has more than two wires attached to it and the length of the wire is minimized [1].

- **Frequency Assignment Problem:** In a communication network the frequency assignment problem is to assign a frequency to each transmitter [1].

- **Machine Scheduling Problem:** Scheduling problems are the most studied application areas for TSP. Consider that there are n jobs *{1,2.,...,n}* to process on a machine. Let c_{ij} be the set up cost for processing the job j immediately after the job i. The objective is to find an order such that the total setup cost of jobs is minimum [1].

- **Longest Path Problem**: The objective of the longest path problems is to find a longest path in a network between a specified pair of vertices. Maximization and minimization problems can be converted into one another by multiplying the objective with -*1* [3].

The TSP is applicable in a variety of other situations including, data analysis in psychology, X ray crystallography, overhauling gas turbine engine, warehouse order-picking, wall paper cutting, arc routing problems, VLSI chip fabrication, matroid intersection, etc… [3, 9, 1]. There are many other TSP applications which we do not cite all of them here.

2 THE TRAVELING SALEMAN PROBLEM WITH PICKUP AND DELIVERY

Traveling Salesman Problem with Pickups and Deliveries (TSPPD) is a generalization of the TSP. In the literature there are many problems related to pickup and delivery but they have different features (single commodity or multi commodities, single vehicle or multi vehicles, one-to-one or one-to-many etc…) [13]. In this section we will give a short description of various Pickup and Delivery Routing Problems in order to show their relations to the TSPPD.

2.1 Pickup and Delivery Problems Variants

2.1.1 The TSP with Pickup and Delivery

The TSPPD is a simple generalization of the TSP. The problem involves two sets of customers. “Delivery Customers” are served by delivery of the goods from a central warehouse. “Pickup Customers” need to deliver goods from their locations to the central warehouse [14]. We call the central warehouse as “Depot” and all delivery and pickup services are done by a single vehicle with a given capacity which is equal to Q. There are two types of goods (delivery and pickup goods), a vehicle which departs from the depot with fully loaded, satisfies all the customer’s needs and returns back to the depot. The objective of the TSPPD is to minimize the total travelling distance. The additional constraint of the TSPPD is that the vehicle load must be feasible throughout the tour; it must remain nonnegative and the total load should not exceed the vehicle capacity [14].

There are many real world applications of the TSPDP. In several industries, vehicles must visit customers and perform at each visit a pickup and a delivery without exceeding the vehicle capacity [15]. Two real applications of the TSPPD are as follows;

- One common application arises in the soft drinks delivery where full bottles are delivered to the customers and the empty bottles are picked-up.

- Another specific application of the TSPPD has been provided by the Fresh Air Fund which is a non-profit welfare organization. The organization arranges summer vacations for under privileged children. Children were sent to the volunteer families to spend two weeks out of the city. The children are picked up and transported by buses. Each bus has a defined region. After assigning the regions to the buses, the remaining problem becomes TSPPD [14].

2.1.2 The One – Commodity Pickup and Delivery TSP

Like Pick-up and Delivery Problems, the one commodity Pick-up and Delivery TSP is a generalization of the TSP where a finite set of cities are identified as customers and one specific city is considered as a vehicle depot. Customers are divided into two different groups according to the type of service required: *Delivery Customers* and *Pickup Customers*. Each *delivery* customer requires a given product amount and each *pickup* customer provides a given product amount. Any amount collected from a pickup customer can be supplied to a delivery customer. A vehicle must start and end its route at the depot with a fixed upper limit capacity. Then the objective of the problem is to minimize the total distance route to satisfy the customers' requirements without exceeding the capacity [10]. There are two versions of the problem; The symmetric 1-PDTSP where cost matrix is symmetric and the asymmetric 1-PDTSP where cost matrix is asymmetric.

As we have mentioned, an important assumption of the 1-PDTSP is that a product collected from a pickup customer can be served to the delivery customer. Real-life applications of the 1-PDTSP arise in the collection of milk from cow farms to serve the private residences [10]. Another application is the transfer of money between bank branch offices. An important application can be cited in the retailers inventory repositioning where there is a finite set of retailers dispersed in a region. Due to the different demands some retailers may have an excess of inventory and the others may have surplus of products. In this case the objective is to transfer the inventory between retailers to satisfy the client's demands.

Differences between 1-PDTSP and TSPPD

- In TSPPD, one type of commodity is collected from customers and the other type is delivered from the depot to the customers. From this point of view, TSPPD is a many to one commodity and one to many commodity type problem. On the other hand in 1-TSPPD there is only one type of commodity and this commodity which is collected from a pickup customer is served to a delivery customer [13].

- A TSPPD solution is feasible if and only if the vehicle capacity is greater or equal to the maximum of the total sum of pickup demands and the total sum of the delivery demands. This condition is not required for 1-PDTSP in which the vehicle capacity should be equal to at least the maximum customer demand.

2.1.3 The Capacitated TSP With Pickup and Delivery

The problem the special case of the 1-PDTSP is called Capacitated TSP with Pick and Delivery where all delivery and pick-up quantities are equal to one unit. It consists of picking up and delivering single objects from source. This problem is also called Q-Delivery TSPPD where Q is the capacity of the vehicle [13]. Q-Delivery TSPPD is a single vehicle and single-commodity problem and that that each city, except the depot, supplies or demands one unit. Anily and Bramel have proposed a better worst case algorithm based on a matching procedure and they have discussed an important application of the CTSPPD in the context of inventory repositioning [10, 16].

2.1.4 The Swapping Problem

The swapping problem is a more general problem where several commodities must be transported from many origins to many destinations with a limited vehicle capacity [13].

The problem is many – to many structure with multi-commodity. There is non-necessary Hamiltonian route and the commodity could be stored in an intermediate customer. The problem has been introduced in [17] and "The Swapping Problem on a Line" has been analyzed in [18].

2.1.5 The Traveling Salesman Problem With Backhauls

The TSP with Backhauls is a closely related problem with TSP where an uncapacitated vehicle must visit all delivery customers before visiting pick-up customers [19]. In other words it's a particular case of the TSP with the additional constraints that a set of locations must be routed before the rest of locations [13].

2.1.6 The Dial - a - Ride Problem

The Dial - a - Ride Problem (DARP) is a special case of the pick-up and delivery problem where there is a one-to-one correspondence between each pick-up and delivery customers. Mostly the commodities transported in the DARP are people [13]. There are many variants of the DARP according requirements, features and optimization functions which can be quoted as follows;

- **Capacitated Dial-a-Ride Problem:** In Capacitated Dial-a-Ride Problem (CDARP) the vehicle with a given capacity should move one unit of the commodity from its origin to its destination.

- **The Stacker Crane Problem:** When the capacity Q is equal to 1, the CDARP is known as the stacker crane problem. Frederickson et al. have proposed an efficient heuristic algorithm for this problem [20].

- **The Pick-up and Delivery Traveling and Salesman Problem:** When there is no vehicle capacity the problem is called "The Pick-up and Delivery Traveling Salesman Problem – PDTSP" [13].

Most articles which refer to the DARP consider additional features, such as multi vehicle version, time windows and/or dynamic requests. Also the objective functions may be different than to minimize the total distance such as; to minimize number of vehicles, to minimize the waiting time of the people, etc…

2.1.7 The Multi Commodity Pickup and Delivery TSP

The M-commodity Pick-up and Delivery Problem is an extended version of the 1-PDTSP. The number of products is incremented from 1 to m [13]. The vehicle must visit a set of customers exactly once with a limited vehicle capacity. Additionally there are several products and each customer can require and / or offer quantities of m different products. In this problem the separation of the customers as delivery and pick-up customers is not possible customer can collect some units of a product and supply some units of a product. TSPPD is also a special case of the 2-PDTSP where the depot is the only origin of one commodity and the only destination of the other. The other particular case of the m-PDTSP is called “One-to-One Multi Commodity PDTSP” where each commodity has only one destination and one origin. Hernandez and Gonzalez [21] have proposed two mixed integer linear programming models to solve this problem.

2.1.8 The General Pickup and Delivery Problem

In the survey paper by Savelsberg and Sol [22] it has been summarized several pickup and delivery problems until mid nineties. The survey describes the General Pickup and Delivery (GPDP) as the problem of transporting different products between different locations [13]. Each vehicle has a given capacity, a start location and a finish location. For each transportation request the size of the load and its destination has been defined [22]. There are three well known particular cases of the GPDP in the literature;

- **The pick-up and delivery problems:** The vehicles depart from and return to a central depot. Also each transportation request has only one pick-up and delivery location.

- **The Dial - a - Ride Problem:** The quantity transported for each product is equal to one unit. (Typically the different products are people)

- **Vehicle Routing Problem:** All origins and all destinations are located at the same location i.e. depot.

A wide variety of objective functions can be found in the GPDP; "to minimize duration, to minimize completion time, to minimize travel time, to minimize route length, to minimize client inconvenience, to minimize the number of vehicles, to minimize profit etc..." [22].

2.1.9 Pickup and Delivery Problem with Time Windows

The problem is called Pickup and Delivery Problem with Time Windows (PDPTW) where a time window is introduced in which the service at any location must take place [22]. Given a delivery and pickup plan, a time window and a loading and unloading time are specified [23]. The customers and the depot have time windows. The time window of a location *i* is determined by an interval $[e_i, l_i]$, where e_i and l_i represent the earliest and latest arrival times. Vehicles must arrive at a location before the end of the time window l_i. They may arrive early but they have to wait until time e_i to begin service [24]. The PDPTW models a variety of operational planning problems in transportation logistics. Applications range from local area courier services to less-than truck load transportation and long-distance haulage. PDPTW also matches typical situations in public transit [23].

2.1.10 Pickup and Delivery Problem With FIFO Loading

The problem is called TSPPD with FIFO Loading (TSPPDF) where the pickup and delivery operations must be done in a first- in-first-out approach. If the pickup of a request *i* is done before the pickup request *j*, then the delivery request *i* must be performed before the delivery request *j* [25].

2.1.11 Dynamic Pickup and Delivery Problem

A problem is said to be static when all input data of the problem are known before routes are constructed. In a dynamic approach some of the input data are updated during the operations. In contrast to what happens in a static problem, the planning horizon of a dynamic problem may be unbounded [26]. One example of a dynamic PDP is the transportation of handicapped and elderly people in urban areas. The dynamic PDP arises when the transportation requests are sometimes received the same day they need to be served. The basic strategy for solving a dynamic problem is to adapt an algorithm to solve the static version of the problem. At this point two different approaches can be formulated; [26]

- The first approach consists of solving the static version of the problem each time with new updated information. The weakness of this approach is that reoptimization of the new information will require too much time and may occurs incoherence with real time setting.

- The second approach is to solve once a time the static version of the problem then to use different heuristics such as *insertion heuristics, deletion heuristics and interchange moves* to update new information to the static optimum solution of the problem. These update mechanisms are sufficiently fast to adapt to the real time setting.

2.2 Pickup and Delivery Problems Variants Classification

There are two important surveys in the literature for TSPPD. Savelsberg and Sol [22] give a definition of the GPDP and classify the problem according to objective functions, time constraints and transportation requests. In their survey, they have divided pickup and delivery problems in two main subgroups; *"static"* and *"dynamic"* pickup and delivery problems. For each static PDP one can formulate its dynamic version. The survey prepared by Berbeglia et al. [27] introduce three fields of classification scheme for the static pickup and delivery problems. These fields are *"Structures"*, *"Visits"* and

"Vehicles". The first field namely *"structure",* defines the number of origins and destinations. *"many-to-many"* problems, *"one-to-many-to-one"* problems and *"one-to-one"* problems are three subclasses on this field. The second field which is *"Visits"* provides information for pickup and delivery operations at customers. The third field i.e *"vehicles"* gives the number of vehicles used in the operation. Henceforth according to the Figure 2.1 we can state that the TSPPD is a static simultaneous pickup delivery problem with a "one-to-many-to-one" structure where there is only one vehicle and that each customers have visited exactly once.

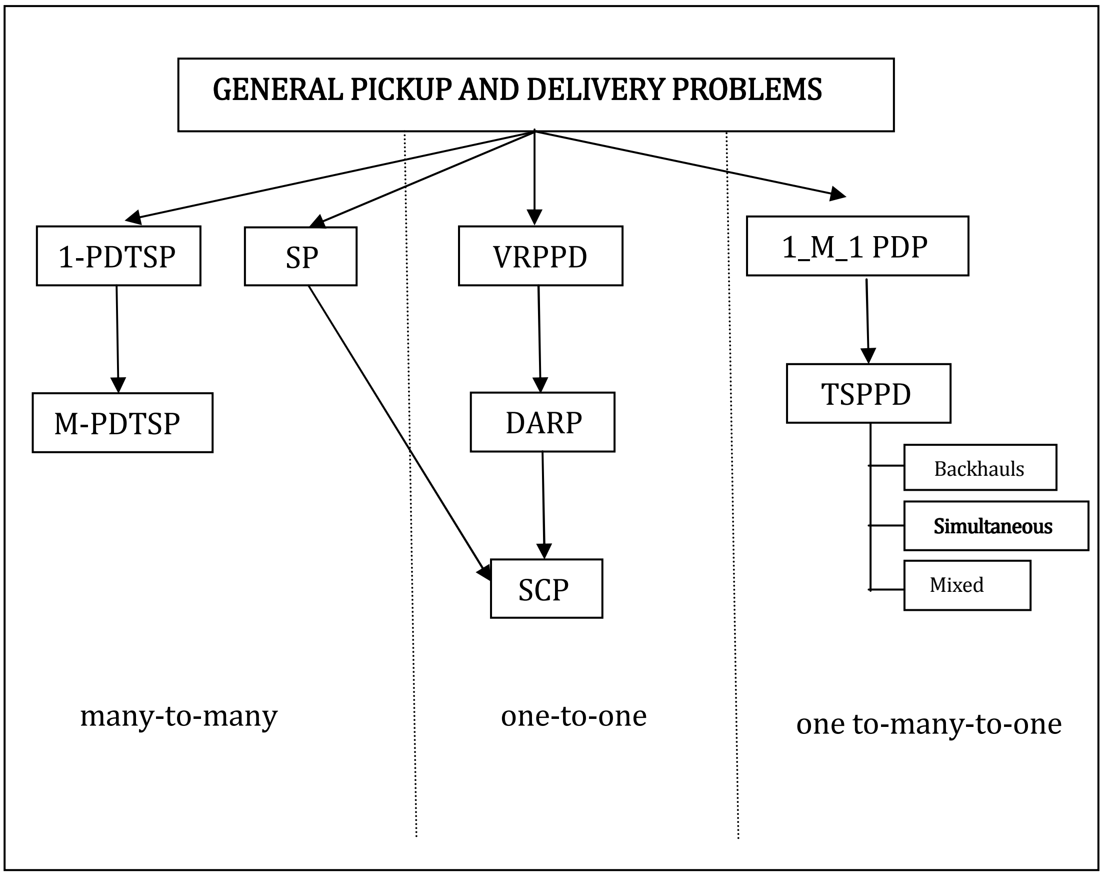

Figure 2.1: Classification of Pickup and Delivery Problems.

2.3 A Mathematical Programming Formulation of the TSPPD

Before presenting the mathematical model we would like to recall the assumptions of the TSPPD;

- The product collected from the pick-up customer is different than the product delivered to the delivery customers. To ensure the feasibility of the problem the vehicle capacity must be equal or greater than the maximum of the total delivery and total pick-up quantities. When the capacity is less than one of these two quantities there is no feasible solution. When the capacity is greater than their sum the problem is automatically reduced to TSP [14].

- There is only one vehicle which serves to the customers. The vehicle departs from the depot fully loaded with the total demand and returns back to the depot with total pick-up quantities.

- Each city is visited exactly once. The delivery and pick-up quantities are served simultaneously for each city. The total load of the vehicle along the tour should never exceed the vehicle capacity [28]. We can clearly see that if delivery and pickup quantities are equal for each city than the problem is automatically reduced to TSP.

To formulate a standard form of the problem we assume both the total pick-up and total delivery quantities are equal to the vehicle capacity. According to the assumptions TSPPD can be formulated as follows [14];

Let P be a set of pick-up locations, D be a set of delivery locations, N be a set of all locations with $N = P \cup D$ and $|N| = n$. Let 0 be the depot location. $C(i,j)$ be the distance between i and j, p_i be the pick-up demand at location i with $i \in P$, d_i be the pick-up demand at location i with $i \in D$ and q be the vehicle capacity. $(p_i \geq 0 \text{ for all } i \in P, d_i \geq 0 \text{ for all } i \in D)$

For standardization we assume that;

$$\sum_{i \in P}^{n} \mathrm{p_i} = \sum_{i \in P}^{n} \mathrm{d_i} = \mathrm{q} \tag{2.1}$$

Decision variable X_{ij} equals to 1 when the vehicle travels along the arc (i, j), Y_{ij} equals to 1 if and only if the total load picked-up carried along arc (i, j) and Z_{ij} equals to 1 if and only if the total load on the vehicle carried along arc (i, j)

TSPPD can be formulated as follows;

$$\text{Min} \sum_{i=0}^{n} \sum_{j=0}^{n} c(i,j) X_{ij} \tag{2.2}$$

Subject to

$$\sum_{i=0}^{n} X_{ij} = 1, \qquad j = 0, \dots, n \tag{2.3}$$

$$\sum_{j=0}^{n} X_{ij} = 1, \qquad i = 0, \dots, n \tag{2.4}$$

$$\sum_{j=0}^{n} Y_{ij} - \sum_{k=0}^{n} Y_{ki} = \begin{cases} p_i & if\ i \in P \\ -q & if\ i = 0 \\ 0 & if\ i \in D \end{cases} \tag{2.5}$$

$$\sum_{j=0}^{n} Z_{ij} - \sum_{k=0}^{n} Z_{ki} = \begin{cases} -d_i & if\ i \in D \\ q & if\ i = 0 \\ 0 & if\ i \in P \end{cases} \tag{2.6}$$

$$Y_{ij} + Z_{ij} \leq q.Z_{ij} \qquad i,j = 0,\dots,n \tag{2.7}$$

$$\begin{aligned} X_{ij} &= 0,1 \\ Y_{ij} &\geq 0 \\ Z_{ij} &\geq 0 \end{aligned} \tag{2.8}$$

In this formulation; constraints (2.3) and (2.4) guarantee that each costumer is visited exactly once. (2.5) and (2.6) are the “multi commodity flow” constraints. If *i* is a pick up location than by constraints (2.5) the amount of p_i is added to the load pick-up without changing the delivery load. If *i* is a delivery location than by constraints (2.5) the amount of d_i is unloaded without changing the pick-up load [14]. Constraint (2.5) and (2.6) ensure that the vehicle departs from the depot with a load equal to the total delivery amount and returns back to the depot with a load equal to the total pick-up load. Finally constraints (2.7) guarantee that the total load of the vehicle will never exceed the vehicle capacity.

2.4 TSPPD Literature Review

In his seminal work Mosheiov [14] defined the TSPPD and proposed a mathematical formulation. He also analyzed two TSP-based methods to solve the problem. Anily and Mosheiov [29] have proposed an efficient polynomial heuristic based on the computation of shortest spanning trees. After constructing the minimum spanning tree, they used a linear time exact algorithm for the special case of TSPPD. Renaud [30] proposed a new heuristic which is composed of two phases: The first phase is the *Double Insertion* heuristic (DI), which inserts each delivery customer simultaneously with its associated pickup customer and the second phase, namely the *Deletion and Re-Insertion* heuristic, is an improvement procedure that employs the 4-Opt improvement heuristic. Gendreau et al. [15] have developed two different heuristics: One is based on the optimal solution of the special case arising when the graph *G* is a cycle and the other is “tabu search” approach using a two-exchange neighborhood. Baldacci [28] has proposed a two commodity flow formulation of the TSPPD. Hernandez-Perez and Salazar-Gonzalez

[31] have proposed two methods to solve 1- PDTSP instances. First method is a greedy algorithm with a k-optimality criterion and second method is an incomplete optimization based on the branch-and-cut procedure. They have also used these two methods to find the exact solution of the TSPPD instances. Hernandez-Perez and Salazar-Gonzalez have focused on 1- PDTSP and proposed several methods; a branch and cut algorithm in [32], another branch and cut algorithm for new inequalities in [33] and finally a mixed heuristic which combines Greedy Randomized Search Procedure (GRASP) and Variable Neighborhood Descent (VND) in [34]. Recently Zhao et al. have proposed a GA to solve the TSPPD and 1-PDTSP [35, 36]. To the best of our knowledge there are no other articles on TSPPD and especially about the use of the GA for TSPPD.

3 EVOLUTIONARY ALGORITHMS

3.1 Evolutions Programs

During the last thirty years there has been a growing interest in problem solving systems based on principles of evolution and hereditary. These systems have some selection processes based on fitness of individuals and some genetic operators. Evolution Programs (EP) is the common term of these systems. The structure of an EP can be formulated as follows [37]. Let P(t) denote the population at time t;

```
begin
    t ⟵ 0
    initialize P(t)
    evaluate P(t)
    while ( not termination condition ) do
    begin
        t ⟵ t+1
        select P(t) from P(t-1)
        alter P(t)
        evaluate P(t)
    end
end
```

Any EP must have the following five components to formulate a given problem;

- a genetic presentation for potential solutions,
- a way to create an initial population of potential solutions,

- an evolution function as the role of the environment to rate solutions according their fitness,
- a genetic operator that alters the composition of children,
- various parameters; population size, probabilities of using genetic operators and mutation operators etc...

Clearly, any EP can be formulated for a given problem, such programs may differ in many ways; they can use different data structures for implementing a single individual, “genetic” operators for transforming individuals, method for creating an initial population, methods for handling constraints of the problem and parameters (population size, probabilities of applying different operators) [37]. However, they share a common principle: The population of individuals undergoes some transformations and during this evolution processes strong individuals strive for survival [37]. This common principle of evolution program constitutes also the basic idea of GA. The conceptual difference between GA and EP can be illustrated with Figure 3.1;

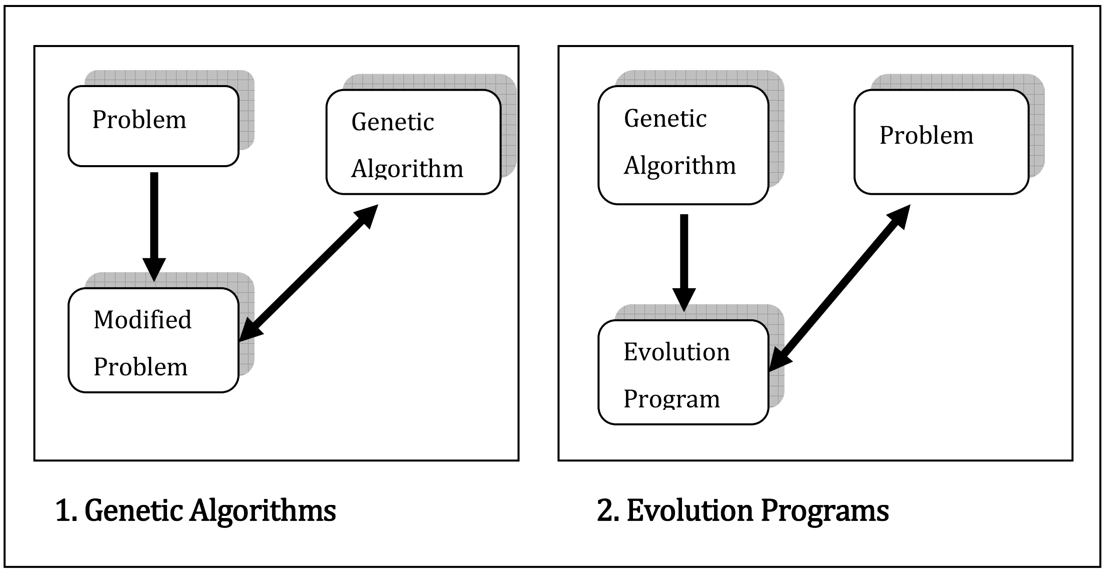

Figure 3.1: GA and EP approaches.

To solve a problem with GAs, we require a modification of the original problem like mapping between potential solutions, binary representations, taking care of decoders or repair algorithms etc... On the other hand evolution programs leave the problem unchanged and adapt the GA process to the problem by modifying genetic operators like chromosome representation of the potential solutions. In summary we can define "Evolution Programs" such as modified GA [37]. The purpose of this chapter is to give detailed information about genetic algorithm which will constitute the basic idea of our evolutionary approach to solve TSPPD.

3.2 Genetic Algorithms

3.2.1 Basic Principle of Genetic Algorithm

GAs are search algorithms based on the mechanics of natural selection and natural genetics [38]. They are adaptive methods which may be used to solve search and optimization problems [39]. The basic principle of the GAs comes from the "survival of the fittest" which is stated by Charles Darwin in "The Origins of Species". In the nature, individuals in a population competes each other for water, food and shelter. Naturally the individuals which are more successful in surviving will have relatively larger numbers of offspring but on the other side poorly performing individuals will produce relatively few offspring [39]. This means that the genes of the strong individual will spread in an increasing number for the successive generations. Then the combination of good characteristics from different parents will produce fitness offspring. The GA uses the direct analogy of this natural behavior. It starts with an initial population and try to obtain fitness offspring by crossing with a predefined rule. The traditional GA can be formulated as follows;

```
Begin
    Generate initial population
    Compute the fitness of each individual
    while not finished do
    begin
            for population size / 2
            begin
                    Select two individuals from old generation for mating
                    Recombine the two individuals to give two offspring
                    Compute fitness of the two offspring
                    Insert offspring
```

```
                Compute fitness of the two offspring
            end
            if population has converged then
                finished: = TRUE
    end
end
```

The power of GA comes from the fact that the method is robust and which can be used successfully for a wide range of problem areas including those which are difficult for other methods to solve [39]. GA differs from the other optimization and search procedures in four ways [38];

- GA works with a coding of the parameter set, not the parameters themselves: The first step of the GA is always to find a way to code the parameters as a finite –length string over some finite alphabet.

- GA searches from a population of points not a single point: In the application of many optimization methods we move in the search space from one single point to the next using some transition rules to determine the next point. The method point-to-point could be dangerous if we are in a multimodal (many peaked) search space. On the other hand GA works with a rich database of points thus the probability of finding a false peak is reduced over methods that go point to point. GA starts with a population of strings and thereafter generates successive population of strings.

- GA uses payoff (objective function) information, not derivative or other auxiliary knowledge: Many search techniques require auxiliary information like gradient techniques which require derivatives. By contrast GAs have no need for an extra information. They only need payoff values (objective function values) associated with individuals strings.

- GA uses probabilistic transition rules, not deterministic rules: GAs use random choice as a tool to guide a search toward regions of the search space with likely improvement.

Assembling together these four differences, use of coding, search from a population, blindness to auxiliary information and randomized information" assure to GAs robustness In the next section we will discuss about the elements of a GA.

3.2.2 The Elements of a Genetic Algorithm

GA is significantly more complicated than neighborhood search methods, with several interacting elements [40];

3.2.2.1 Chromosome Representation – Coding

The potential solution to a problem may be represented as a set of parameters. These parameters so called *genes* are joined together to form a *chromosome* [39]. The position of a *gene* in a string called its *locus* and *allele* is the set of values that the *genes* can assume [40]. Many researchers still believe that the ideal to represent a chromosome is to use a binary alphabet for the string [39]. On the other hand, Janikow and Michalewichz [41] made a comparison between binary and floating-point representations and have shown that the floating point version gave faster and more accurate results. The important advantage of using non-binary representation is that we can easily adapt and use different mutation and crossover techniques [42]. Let us consider a simple example; suppose that we have a black box with bank of five input switches. For every setting of the five switches there is an output signal. Simple code can be generated by considering a string of five 1's and 0's where each of the five switches is presented by a 1 if the switch is on and a 0 if the switch is off. A coding (10000) represents that the first switch is on and the others are off. Chromosome representation is a very important and difficult step which affects clearly the accuracy of the GA.

3.2.2.2 Fitness Function

A fitness function must be formulated for each problem to be solved. For a chosen crossover the fitness function returns a single numerical "fitness". For many problems especially for function optimization the fitness function should measure the value of the function. But this is not always the case, for example with combinatorial optimization. In a realistic bridge design task there are many performance measures which we may want to optimize such as strength/weight ratio, span, maximum load, cost, construction time etc… [39].

3.2.2.3 Initial Population

The size of the initial population and the method to generate an initial solution are two important questions for initial population. GAs converge more rapidly with smaller population but better results are obtained with larger population. Intuitively, there should be some optimal value for a given chromosome length [40]. Goldberg has reported that the population size increase as an exponential function of the chromosome length. [43, 44] The author has shown that a linear dependence of population size on chromosome length is adequate, but even there is a linear relation between population size and chromosome length in some cases we have to work with larger population. At this point the question we could ask is to find a minimum population size for a meaningful search [40]. According to the principle of "every point in the search space should be reachable from the initial population by crossover only" which was adopted in [45] we can conclude that there is at least one instance of every allele at each locus in the whole population [40]. Assume that the initial population is generated by a random sample. According to this assumption the probability that at least one allele is present at each locus can be formulated as follows;

$$P = (1 - (1/2)^{N-1})^{l} \tag{3.1}$$

using an exponential function approximation;

$$P \approx exp(-l/2^{N-1}) \tag{3.2}$$

We can easily establish that;

$$N \approx 1 + \log\left(-\frac{l}{ln\,P}\right)/log2 \text{ holds.} \tag{3.3}$$

According to this formulation; A population of size 17 is strong enough to ensure a probability of 99.9% for strings of length 50.

On the other hand the method to generate an initial solution is also another important issue. Many researchers assume that initialization of the initial solution should be random [40] but Schmitt and Amini have reported that for various problem classes and sizes, a hybrid initial population yields superior results over a pure random initial population [46]. Ahuja and Orlin have also reported that an initial solution obtained from a heuristic can help a GA to find better solutions more quickly than a random start [47].

3.2.2.4 Convergence and Termination Criterion

Neighborhood search methods terminate when a local optimum is reached [40]. On the other hand GAs could in principle run forever if we don't determine a termination criterion. Common approaches are to set a limit for fitness evaluation, computer clock or to track the population diversity [40].

3.2.2.5 Reproduction: Crossover and Mutation

During the reproduction phase of the GA, parents are selected from the population and recombined to obtain offspring for the new generation. Crossover is a methodology to recombine selected parents. Basically crossover takes two individuals, cuts their chromosome strings at some randomly chosen positions to produce two "head" and two "nail" segments. At this stage the tail segments are then swapped to produce two new chromosomes. This crossover method called "single point crossover" can be observed in the Figure 3.2;

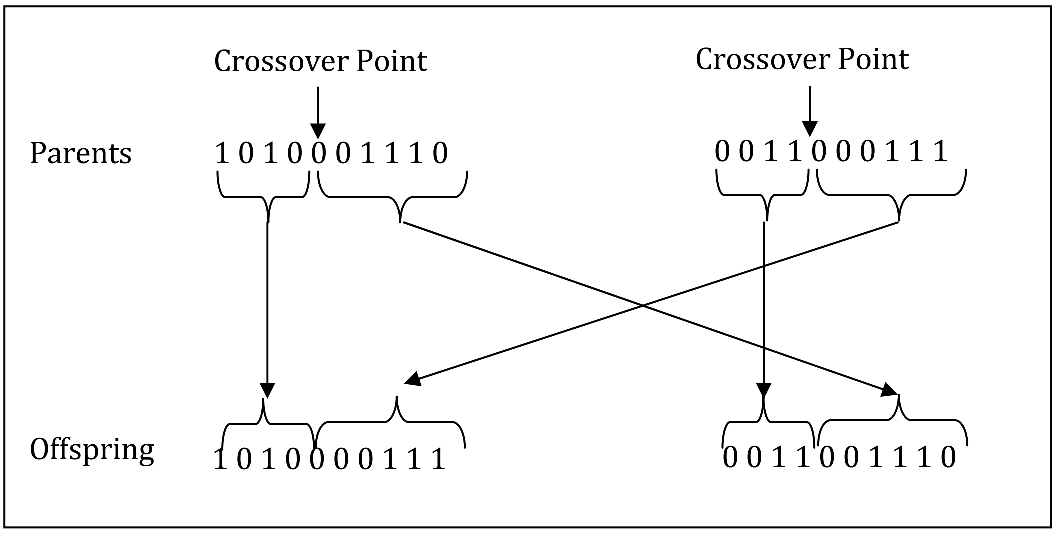

Figure 3.2: Crossover Operation

The Mutation operation has been applied to each child after crossover. The principle of the mutation is to change randomly a gene of a chromosome with a small probability. (see Figure 3.3)

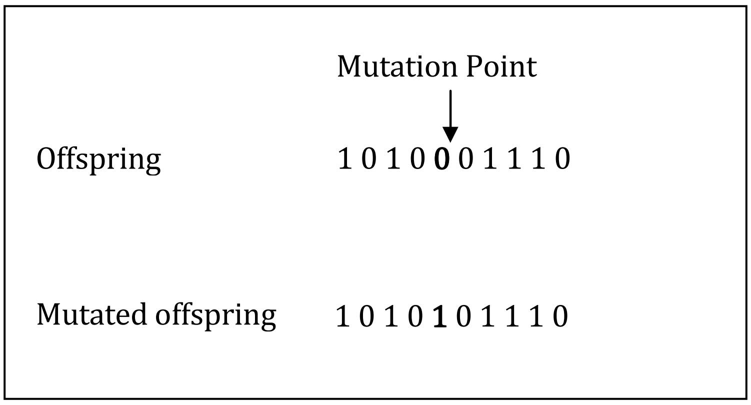

Figure 3.3: Mutation Operation

Mutation provides a small amount of random search and helps us that there isn't any point in the search space which has a zero probability of being examined.

3.2.2.6 New Population

When the reproduction phase is completed we need to define a selection strategy between parents and offspring to generate the new population. Slightly different strategies are commonly used in the literature. Some GAs assume that parents are replaced by their children [40]. Many implementations use the tactic of deleting the worst members of the population [48]. However this strategy may need large populations and high mutation rates to prevent a rapid loss of diversity [49].

3.2.3 An Illustrative Example

At this stage, let us try to solve a simple function by using GA. Suppose that we have the following with one variable.

$$f(x) = x.\sin(10\pi.x) + 1.0 \qquad (3.4)$$

The problem is to find the value of x which maximizes the function f from the range [-1, 2] (See Figure 3.4 for the graph of the function f)

$$f'\ (x) = \sin(10\pi.x) + 10\pi.x.\cos(10\pi.x) = 0 \ \Rightarrow\ tan(10\pi.x) = -10\pi.x \qquad (3.5)$$

The equation (3.5) has an infinite number of solutions. Since the domain of the problem is $x \in [-1,2]$ the function reaches his maximum at $x_{max} = \frac{37}{20} + \varepsilon$ where ε is very small number. $f(x_{max})$ is slightly larger than $f\left(\frac{37}{20}\right) = \frac{37}{20} * \sin\left(18\pi + \frac{\pi}{2}\right) = 2.85$

Now we will devise a simple GA for the solution of this problem.

3.2.3.1 Chromosome Presentation

The domain of the variable x is in the interval [-1,2] and the length of this interval is 3. Suppose that we wish solve the problem with a required precision of six digits. This precision implies that the interval [-1,2] should be divided into at least 3*10000000 equal size range. For our binary vector 22 bits is required;

$$2097152 = 2^{21} < 3000000 \leq 2^{22} = 4194304$$

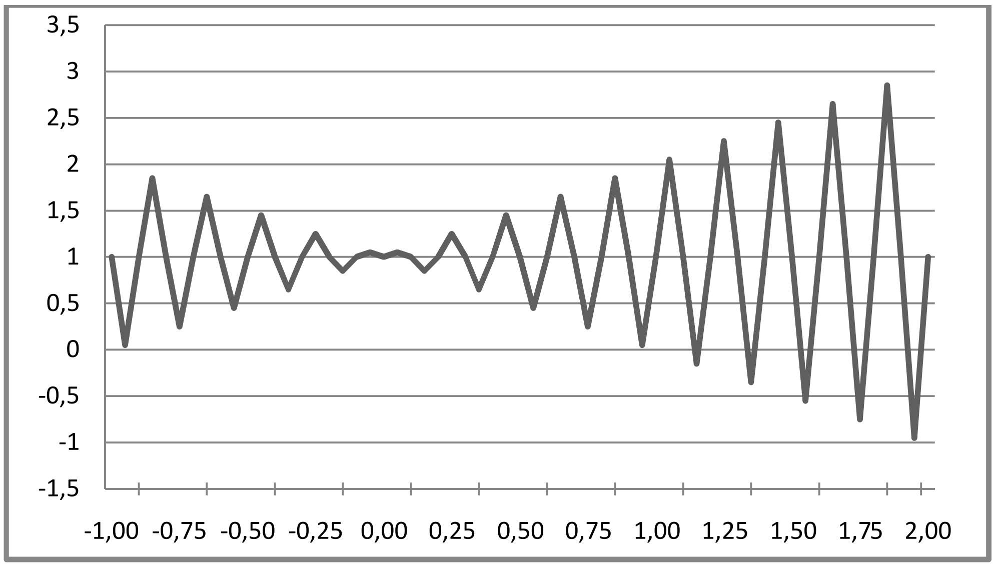

Figure 3.4: Graph of the function f(x).

The transformation of a binary string to a real number x from the interval [-1,2] will be done in two steps;

- Convert the binary string from the base 2 to base 10.

$y = (b_{21}; b_{20}; \ldots.; b_0)_2$

- Find a corresponding number x from the range [-1,2];

$x = -1{,}0 + y * \frac{3}{2^{22}-1}$

For instance to transform a chromosome (1000101110110101000111) to a real number x from the interval [-1,2], we perform the following operations;

First we transform the chromosome from base 2 to base 10;

$y = (1000101110110101000111)_2 = 2288967$ where

Than we compute the corresponding number within the interval [1, 2].

$$x = -1{,}0 + 2288967 * \frac{37}{2^{22}-1} = 0.637197$$

Note that the following chromosomes (0000000000000000000000) and (1111111111111111111111) represent of the boundaries of the domain -1.0 and 2.0 respectively.

3.2.3.2 Initial Population.

The initial population of the genetic algorithm is generated randomly. We create a population of chromosomes where each chromosome is a binary vector of 22 bits.

3.2.3.3 Fitness Function

The fitness function for binary vectors is equivalent to function f; fit(v) = f(x). The fitness function plays the role of the environment to evaluate potential solutions of the problem. For example, three chromosomes;

v_1 = (1110110110110110110110) corresponds x_1=1.785714
v_2 = (0010010010010010010010) corresponds x_2=-0.571430
v_3 = (1110001110001110000000) corresponds x_3=1.666657

fit(v_1) = f(x_1) = 0.225193
fit(v_2) = f(x_2) = 0.553254
fit(v_3) = f(x_3) = 2.443620
Clearly the third chromosome is the best of the three because its fitness value returns the highest value.

3.2.3.4 Crossover and Mutation Operators

"The single point crossover" is used with a probability p_c and a simple one point mutation is used with a probability p_m (Recall that genetic operators have been discussed in the section 2.2.5)

3.2.3.5 Parameters

The following parameters have been used in our genetic algorithm;

- Population size is equal to 50.
- Crossover and mutation probabilities p_c and p_m are equal to 0.25 and 0.01 respectively
- The new population is generated by replacing the 25 best solutions of the offspring with 25 worst solution of the parents population. The algorithm ends by obtaining 150 generations.

3.2.3.6 Experimental Results

The best chromosome after 150 generations was v_{max}= (1111001101000100000101) corresponds to x_{max}=1.850773 where $f(x_{max})$ = 2.850227

The Table 3.1 shows the observed improve of the generations;

Table 3.1: Improvements of the generations.

Generation number	1	10	40	99	145
Fitness Function	1.441942	2.250363	2.345087	2.849246	2.850227

3.2.4 Mathematical Background of Genetic Algorithm

3.2.4.1 Schemata Theorem

Holland's schemata theorem was the first rigorous explanation of GAs mathematical background [50]. The theoretical foundations of GAs rely on binary string presentation of solutions and on the notion of a schema [37]. A schema (similarity template) is a pattern of gene values which may be presented by a string of characters in the alphabet {1, 0, *} In this alphabet * means "wild card" or "don't care symbol". As an example consider the strings (chromosomes) of length 6 in a population A. The schema (1110*0) represents two strings {(111000), (111010)} and the schema (*0000*) describes a subset of 4 strings; {(100001), (100000), (000001), (000000)} Of course the schema (111100) represents one string only and the schemata (******) represents all strings of length 6. The total number of schemata in a population A consists of strings with a length l is 3^l. In the previous example there are 3.3.3.3.3.3 = 3^6 = 729 schemata because each of the six positions may be 1, 0 and *. Without loss of generality for an alphabet with k elements (in our example k is equal to 2) there are only $(k+1)^l$ schemata in a population. Note that in the previous example with $l = 6$ there are $2^6 = 64$ strings. Why should we consider 729 schemata instead of 64 strings? Why to make the problem more difficult by enlarging the space of concern? The answer of these questions is that by interesting in strings alone we have restricted information about the population but if we consider the similarities among the strings in a population we add a wealth of new information to help our search [38]. Each string of the length l is matched by 2^l schemata [37]. For example let us consider a string (10100000000011111111) from a population A. This string is matched with the following 2^{20} schemata;

(1 0 1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1)

(* 0 1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1)

(1 * 1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1)

(1 0 * 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1)

.

.

(1 0 1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 *)

(* * 1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1)

(* 0 * 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1)

.

.

(1 0 1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1**)

(* * * 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1)

.

.

(* * * * * * * * * * * * * * * * * * * *)

In a population of size n there are n×2^l different schemata may be presented. We will especially interest in the transportation of n×2^l different schemata to the next population in order to define an explanation of how GAs work. Then we will try to measure the effect of the "selection", "crossover" and "mutation" to these schemata.

All schemata are not created equal. Different schemata have different characteristics. There are three important properties to evaluate schemata; *"schemata order", "defining length"* and *"fitness of a schema".* The order of a schema denoted by *o(H)* is the number non "*" symbol it contains [39]. For example the following three schemata each of length 15.

H_1 = (1*10*100**1****)
H_2 = (*****100**1****)
H_3 = (1110100**1**110)

have the following orders; $o(H_1) = 7$, $o(H_2) = 4$, $o(H_3) = 11$. As you can observe the schema H_3, is the most specific one. The order of a schema is useful to calculate the survival probabilities of the schemata for mutations [37]. The defining length of the schema, denoted by $\delta(H)$ is the distance between the first and the last non "*" symbol. It defines the compactness of information contained in a schema [37]. For example $\delta(H_1) = 11 - 1 = 0$, $\delta(H_2) = 11 – 6 = 5$ and $\delta(H_3) = 15 – 1 = 14$. The defining length of a schema is useful to calculate the survival probabilities of the schema for crossovers [37]. The fitness of a schema denoted by *f(H)* is the average fitness of the strings which belong to the subset represented by schemata. The effect of "selection" on the expected number of schemata is easy to determine. Let us denote *m(H,t)* the number of strings which belong to the subset represented by schemata *H* at a time *t* within the population A. During selection a string is selected according to its fitness. More precisely a string gets selected with probability $p_i = \frac{f_i}{\Sigma f_j}$. The growth equation of number of the schemata at time t+1 can be written as follows;

$$m(H, t+1) = m(H, t)\frac{f(H)}{\bar{f}} \qquad (3.6)$$

where *f(H)* is the average fitness of the schemata H and $\bar{f}$ is the average fitness of the population A In other words the number of strings in the population grows as the ratio of the fitness of the schemata to the average fitness of the population. If we define *f(H)* as *f(H)* = $\bar{f} + c\bar{f}$ the long term effect of the selection can be formulated as follows;

$$m(H, t+1) = m(H, 0)(1+c)^t \tag{3.7}$$

This is a geometric progression equation; if c $>$ 0 "above average" schema receives an increasing number of strings in the next generation, if c $<$ 0 "below average" schema receives a decreasing number of strings in the next generation. Recall that the usage of crossovers and mutations are essential for the convergence of GAs to the optimum solution. To show the effect of a single point crossover with a random selection of a mate to the schemata let us consider the following example. Suppose that H_1 and H_2 are two different schema which both represent a string S and that we apply a single point crossover for the string S just between third and forth position of the string;

S = 1 0 1| 0 1 1 1
H_1 = * 0 *| * * * 1
H_2 = * * * | 0 1 * *

As it can be observed when we apply the single point crossover the schema H_1 will be destroyed because 0 at position 2 and 1 at position 7 will be placed in different offspring. (They are on opposite sides of the cut point.) On the other hand H_2 will not be destroyed because the two non "*" symbols placed in one side of the cut point. The order of two schemata is equal but their defining lengths are different; $\delta(H_1)=5$ and $\delta(H_2)=1$. If the crossover cut point is defined uniformly at random according to the length l = 7 -1 = 6 of possible points, we can say that H_1 is destroyed according to the probability $p_d = \delta(H_1)/(l-1)$. More generally the survival probability may be given as just below;

$$p_s \geq 1 - p_c \frac{\delta(H)}{l-1} \tag{3.8}$$

Then if we recombine the effect of selection and crossover we obtain the estimate;

$$m(H,t+1) = m(H,t)\frac{f(H)}{\bar{f}}\left[1 - p_c\frac{\delta(H)}{l-1}\right] \quad (3.9)$$

This estimation tells us about the expected number of strings matching the schemata *H* in the next generation as a function of the actual number of strings matching the schema, relative fitness of the schema, and its defining length [37].

Finally, the effect of the mutation can be adapted to the formulation as follows; the mutation is the random alteration of a single position with probability p_m. For a schema *H* to survive, all of the non"*" symbols must survive. So survival probability of a schema is equal to $(1- p_m)^{o(H.}$. For small values of p_m ($p_m << 1$) the schema survival probability may be approximated by the expression $(1- p_m).$ *o(H).* Then we can enlarge our estimation by adding the mutation survival probability;

$$m(H,t+1) = m(H,t)\frac{f(H)}{\bar{f}}\left[1 - p_c\frac{\delta(H)}{l-1} - p_m.o(H)\right] \quad (3.10)$$

The conclusion of this estimation is that short and low order, receive exponentially increasing trials in subsequent generations [38]. This conclusion is also named "Schemata Theorem" which constitutes the basic idea of GAs. An immediate result of this theorem is that GAs explore the search space by short and low order schemata which are used for information exchange during crossover [37].

3.2.4.2 Implicit Parallelism

In the previous section we have mentioned that in a population A with *n* strings of length *l* there are at least 2^l and at most $n\times 2^l$ schemata [37]. Some of these schemata are processed in a useful manner (desirably manner). Holland has shown that at least n^3 schemata are processing in a useful manner. Holland called this property "implicit parallelism" and proved this equality [50];

$$n_s = \frac{(l-l_s+1).\mathrm{n}^3}{4} \Rightarrow \mathrm{C}.\mathrm{n}^3 \quad (3.11)$$

where n_s is the number of schemata, l is the length of the string and l_s is the length of the schemata. Holland obtained this equation by restricting the population size to $2^{ls/2}$.

We conclude that the number of schemata is proportional to the cube of the population size and that despite the perturbation of long and high-order schemata, genetic algorithms process a large quantity of schemata while processing relatively small quantities of strings [38].

3.2.4.3 Building Block Hypothesis

As a result of the schemata theorem GAs work well, when short and low order, highly fit schemata recombine to form even more highly fit schemata [51]. These short and low order schemata are called *"Building Blocks"* by Goldberg [38]. On the other hand by working with these particular schemata we have reduced the complexity of our problem because we construct better and better strings from best partial solutions [38].

3.2.5 Application Areas of Genetic Algorithms

There are a variety of application areas for genetic algorithms. Some of these applications have been used in practice while others remain as research topics [39];

- ***Numerical Function Optimization:*** GAs have been used to solve difficult, discontinuous, multi model and noisy functions.

- ***Image Processing:*** With medical or satellite images, we often need to align two images of the same area taken at different times. GAs can efficiently find a set of equations which transform one image to fit onto the other.

- ***Combinatorial Optimization:*** The most widely studied combinatorial task is *"The Travelling Salesman Problem"*. The task is to find the shortest route by visiting each city exactly once. Another one is called "Bin Packing" which deals the task with determining how to fit a set of objects into a minimum number of bins. This problem has many applications in industry such as *"job shop* *scheduling", "time tabling"* etc... For solving these kinds of problem we need different coding, recombination and fitness function techniques.

- ***Design Task:*** These tasks are mix problem of combinatorial and function optimizations. Bridge structures, fire hose nozzle and neural network structure are examples of the design tasks.

- ***Machine Learning:*** There are many applications of GAs for learning systems; the usual paradigm being of a classifier system. GAs try to evolve a set of if-then-else rules to deal with some particular situation. This can be applied to the game playing, maze solving and economic modeling [39].

3.3 Solving TSP with Genetic Algorithm

In this section we give a discussion on solving the TSP by using GA. As we have mentioned before the problem is NP Hard. This implies that there is no polynomial time algorithm for the TSP unless P = NP [52]. The aim of this section is to define the settings of the GAs.

3.3.1 Chromosome Representation

In the literature there are three common vector presentations mostly used since 1980s; *"adjacency"*, *"ordinal"* and *"path"*. We will explain these representations by using an example of nine cities.

Adjacency Representation

This representation encodes a tour as a list of n cities [2]. If the tour leads from the city *i* to the city *j* according adjacency representation city *j* will be listed in a position *i*. For example the vector (3 8 5 2 6 4 1 9 7) represent the following tour; 1 – 3 – 5 – 6 – 4 – 2 – 8 – 9 – 7. However some adjacency lists can present an illegal tour; vector (2 4 8 1 9 3 5 7 6) leads to a subtour; 1 – 2 – 4 – 1.

Ordinal Representation

The ordinal representation encodes a tour as a list of n cities; the i-th element of the list is a number from *1* to *n-i+1*. There is some ordered list of cities C which serves as a reference point for lists [2, 37]. For example given an ordered list (reference point) C = (9 8 7 6 5 4 3 2 1), a tour A = (9 – 8 – 6 – 7 – 2 – 5 – 4 – 3 – 1) is presented as a list *l* of references *l* = (1 1 2 1 4 1 3 1 1).

The first number of list *l* is 1 so the first city of ordered city C is 9 as the first city of tour A. Remove the selected city from C and continue to the iteration until the element number of C is null. The main advantage of the ordinal presentation is that the classical *"cut-and-splice"* method works. This method means that if we split two different tours with ordinal representation and if we exchange the splitted parts between tours, we will always generate a legal tour [2].

For example, according to the reference list C = (1 2 3 4 5 6 7 8 9) the two parents; *p1* = (1 1 4 6 | 3 2 1 1 1) and *p2* = (2 2 2 1 | 1 1 1 1 1) corresponds the tours *t 1* = 1 – 2 – 6 – 9 – 5 – 4 – 3 – 7 – 8 and *t2* = 2 – 3 – 4 – 1 – 5 – 6 – 7 – 8 – 9. With the cross point marked "|" we apply *"cut-and-splice"* method and produce the offspring *o1* = (1 1 4 6 | 1 1 1 1 1) and *o2* = (2 2 2 1 | 3 2 1 1 1). These offspring correspond to the tours t3 = 1 – 2 – 6 – 9 – 3 – 4 – 5 – 7– 8 and t4 = 2 – 3 – 4 – 1 – 7 – 6 – 5 – 8 – 9.

Path Representation

The path representation is the most natural representation of a tour. For example a tour

5 – 3 – 1 – 2 – 9 – 7 – 8 – 4 – 6 is represented as (5 3 1 2 9 7 8 4 6)

3.3.2 Crossovers for TSP

In this section we will present the best known crossovers for "Path Representation";

3.3.2.1 Partially Mapped Crossover (PMX)

The PMX builds an offspring by choosing a subsequence of a tour from one parent and preserving the order and position of as many cities as possible from the other parent [2, 53]. Two random cut points are selected and cities inside this two cut points are swapped with each other. For example, the two parents with two cut point can be presented as follows;

p1 = (9 8 7 | 1 2 3 5 | 4 6) and p2 = (9 2 1 | 6 5 4 3 | 8 7)

Firstly the cities between cut points are swapped each other. (The symbol "x" can be seen as "at present unknown")

o1 = (x x x | 6 5 4 3 | x x) and o2 = (x x x | 1 2 3 5 | x x)

where 6 ↔ 1, 5 ↔ 2, 4 ↔ 3, 3 ↔ 5, are swapped. According to this match we can fill in additional cities from original parents.

o1 = (9 8 7 | 6 5 4 3 | x x) and o2 = (9 x x | 1 2 3 5 | 8 7)

The first *x* in *o1* should be *4* but *4* is replaced *3* because of mapping. The match of the *4* is *3* but there is also a conflict because *3* is also replaced *5* in the first offspring. Furthermore we search the match of *3* which is *5* and then the match of the *5* which is *2*. We replace the first *x* of the first spring by *2* because *2* has not been used yet. According this rule we fill the others *x* and obtain *2* new offspring;

o1 = (9 8 7 | 6 5 4 3 | 2 1) and o2 = (9 4 6 | 1 2 3 5 | 8 7)

3.3.2.2 Order Crossover (OX)

The OX builds an offspring by choosing a subsequence of a tour from one parent and preserving the relative order of cities from the other parents [2, 54]. From the same example 2.6.2.1 we obtain;

p1 = (9 8 7 | 1 2 3 5 | 4 6) and p2 = (9 2 1 | 6 5 4 3 | 8 7)

First of all we copy the segments between cut points to the offspring;

o1 = (x x x | 1 2 3 5 | x x) and o2 = (x x x | 6 5 4 3 | x x)

Then starting from the second cut point of one parent the cities from the other parent are copied in the same order. We select the second offspring and write the sequence of the cities; 8 – 7 – 9 – 2 – 1 – 6 – 5 – 4 – 3, by removing 1, 2, 3, 5 which are already in the first offspring we obtain; 8 – 7 – 9 – 6 – 4. Finally according this sequence we fill the unknown symbols of the first offspring starting from the second cut point.

o1 = (9 6 4 | 1 2 3 5 | 8 7) Similarly we produce the other offspring; o2 = (7 1 2 | 6 5 4 3 | 9 8)

3.3.2.3 Cycle Crossover (CX)

The CX builds an offspring in such a way that each city and its position comes from one of the parents [2, 55] This crossover works as follows;

p1 = (1 2 3 4 5 6 7 8 9) and p2 = (4 1 2 8 7 6 9 3 5)

First of all the first city of the first parent is chosen to be equal to 1.

o1 = (1 x x x x x x x x)

Since every city should be chosen from one of its parents, the next city must be city 4.
o1 = (1 x x 4 x x x x x)

Furthermore we continue until we have a cycle;

o1 = (1 x x 4 x x x x x)
o1 = (1 x x 4 x x x 8 x)

o1 = (1 x 3 x4 x x x 8 x)

o1 = (1 2 3 4 x x x 8 x)

The remaining cities are filled in from the other parent

o1 = (1 2 3 4 7 6 9 8 5)

Similarly;

o2 = (4 1 2 8 5 6 7 3 9)

CX preserves the position of cities in the parent sequence [2].

3.3.2.4 Nearest Neighborhood Crossover (NNX)

The NNX randomly selects a node as a starting point. Than a single offspring is generated by visiting the nearest unvisited node. If all neighbors of the selected city has already be used in the offspring then we choose the nearest city according to the cost matrix. Suppose that we have a distance matrix (see Table 3.2) for a STSP with 7 cities and that two parents A = (1 3 2 4 7 6 5) and B = (7 5 6 4 1 2 3). The fitness value of A and B are 32 and 31 respectively according to the distance matrix.

For a starting point we randomly select city *3* which is the first node of our offspring.

o1= (3 x x x x x x **)**

In the two parents A and B there are *3* neighbors to city *3* which are {1, 2, 7} and the distance of the neighbors to city *3* are {7, 2, 8} respectively. Then we choose city *2* as the second node of the offspring because its distance to city *3* is the minimum.

o1= (3 2 x x x x x **)**

The unvisited neighbors of city *2* are {1, 4} and their distance to city *2* are {5, 1}. Where we choose city *4* as the third node of the offspring because its distance to the city *2* is the minimum.

Table 3.2: Cost matrix for illustrative example.

Node	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1	0							
2	5	0						
3	7	2	0					
4	4	1	10	0				
5	8	4	11	4	0			
6	2	3	12	3	4	0		
7	1	4	8	2	5	8	0	
8	9	12	1	2	3	4	6	0

o1= (3 2 4 x x x x **)**

According to this rule we generate the offspring o1 = (3 2 4 7 5 6 1). The fitness value of the new offspring is 23 which is less than the fitness values of both parents.

3.3.2.5 Edge Assembly Crossover (EAX)

EAX is one of the most efficient and effective crossovers for TSP. The basic steps of the crossover can be quoted as follows [56, 57];

- Define a graph G_{AB} by merging the edges of two parents A and B. Each edge in G_{AB} is annotated with the parent to whom it belongs. Note that G_{AB} may contain two instances of the same edge.

- Divide the edges on G_{AB} into AB-cycles. These cycles have been generated by alternately tracing the edges of the tours A and B.

- Construct an E-set by selecting AB cycles according to a predefined rule. There are two important methods proposed in the literature;
    - E-set constructed by randomly selecting AB cycles.
    - The E-set is constructed from a single AB cycles. The intermediate solution is similar to tour-A. Children are constructed by removing a small number edges from tour A and adding the same number of edges from tour B to tour A.
- Generate intermediate tour by applying the E-set to tour A by removing tour A's edges in the E –set from tour-A and adding tour B's edges in the E-set to it.
- Modify the intermediate solution to obtain a valid tour.

An illustrative example of the EAX is given with Figure 3.5.

3.3.3 Heuristics for TSP

In this section we will discuss about tour construction and tour improvement procedures. Tour construction procedures build an optimal tour starting from the distance matrix. Tour improvement procedures start with a feasible tour and seek to improve the tour with interchanges [13].

3.3.3.1 Tour Construction Heuristics

Tour construction heuristics for the TSP starts with a partial tour of a few nodes, next selects a non tour node according to a particular criterion, inserts that node at a position in the partial tour and repeats selection-insertion moves until all nodes have been inserted to the complete tour. There are three key components of the tour construction; *"choice of an initial sub tour"*, *"the selection criterion"* and *"the insertion criterion"* [3]. According to this definition we will present 4 tour construction heuristics;

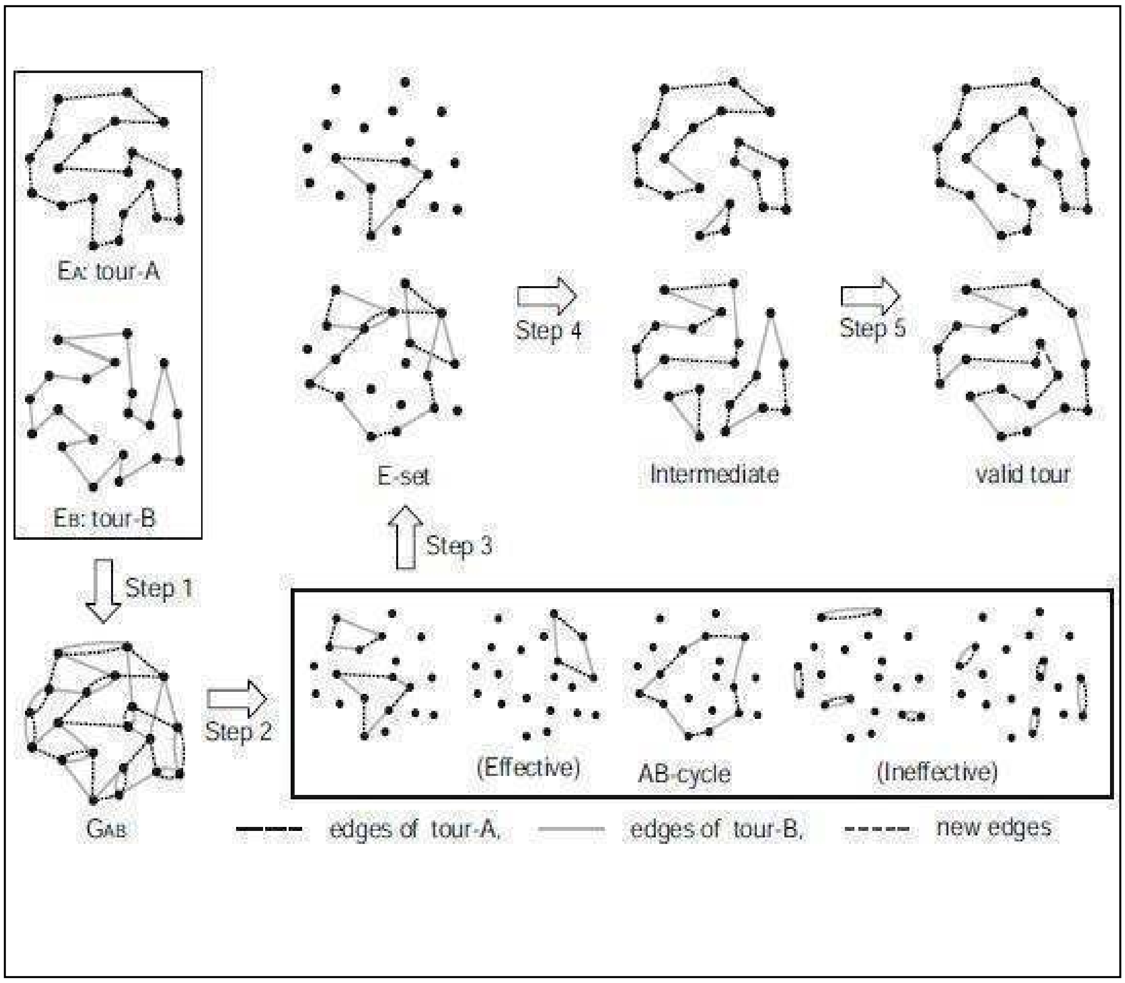

Figure 3.5: An illustrative example of the EAX.

3.3.3.1.1 Nearest Neighbor Heuristic

The aim of the Nearest Neighbor (NN) heuristic is to always visit the nearest city. Firstly we randomly select a city and then we find the nearest unvisited city. We repeat the selection procedure until there is not any unvisited city [58]. Consider the same example of the Section 2.6.4.2 and suppose that we select the city *3* as a starting point. The second city of the tour will be the city *8*, because this city is the nearest city to the city *3*. According to the NN the complete tour of the example will be (3 8 4 2 3 1 7 5 6), and the fitness value of the tour is *35*.

3.3.3.1.2 Nearest Insertion Heuristic

Nearest Insertion (NI) heuristic selects the non tour node whose distance to tour node is minimum and inserts the selected node at a position such that the increase in cost is minimized. Suppose that we have a node k which we want to replace it between nodes i and j. The variation in cost can be calculated as; $\Delta c = c_{ik} + c_{kj} - c_{ij}$

Considering the example of Section 2.6.4.2 and suppose that we select three city as a starting tour; (3 8 4) we first calculate the distance of the non tour node to tour node and we select the city *1*. After the selection, we define the position of city *1* in the tour by calculating the increase in total cost if we insert the city 1 between nodes (3, 8), (8, 4) and (4, 3) respectively. These are $\Delta c_1 = c_{31} + c_{18} - c_{38} = 7 + 9 - 1 = 15$, $\Delta c_2 = c_{81} + c_{14} - c_{84} = 4 + 9 - 2 = 11$, $\Delta c_3 = c_{41} + c_{13} - c_{43} = 4 + 7 - 10 = 1$. Since c_3 has the minimum increase we insert the city *1* between *4* and *3*. The new tour is (3 8 4 1). According to the NI the complete tour of the example will be (3 8 5 4 7 1 6 2), and the fitness value of the tour is *18*.

3.3.3.1.3 Farthest Insertion

Farthest Insertion (FrI) selects the node whose minimum distance to a cycle node is maximum and inserts the selected node at a position such that the increase in cost is minimized. Considering the example of Section 2.6.4.2 and suppose that we select three city as a starting tour (3 8 4). According to the FrI the complete tour of the example will be (3 8 5 4 6 1 7 2), and the fitness value of the tour is *20*.

3.3.3.1.4 Cheapest Insertion

Cheapest Insertion (CI) selects the node that can be inserted at the lowest increase in cost and inserts the selected node at a position such that the increase in cost is minimized. Considering the example of Section 2.6.4.2 and suppose that we select three city as a starting tour (3 8 4). According to the CI the complete tour of the example will be (3 8 5 4 6 1 7 2), and the fitness value of the tour is *16*. Figure 3.6 visualizes the difference between insertions heuristics. NI selects the node k, FI selects the node m and CI selects the node n.

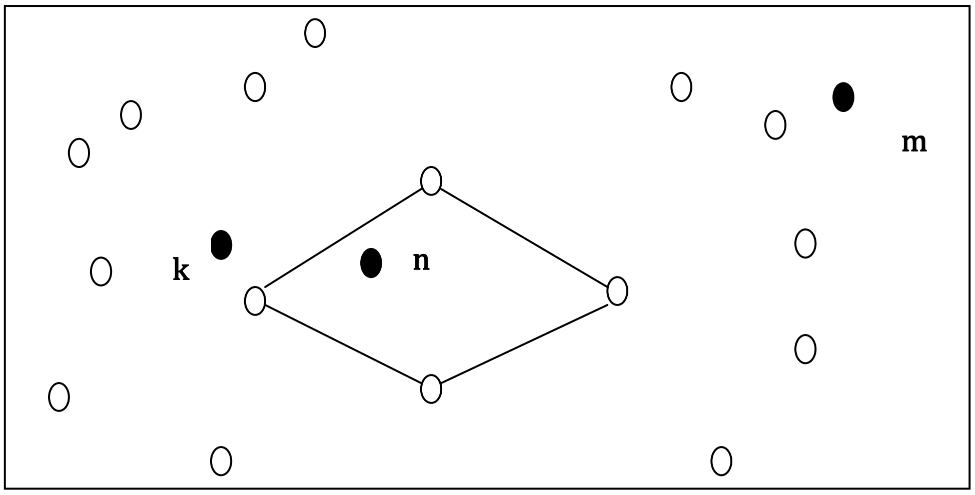

Figure 3.6: The difference between insertions heuristics.

3.3.3.2 Tour improvement heuristics

Tours generated by the construction heuristics are moderate quality and they are not satisfactory in general. The improvement heuristics are used to cover the weakness of the construction heuristics. Improvement heuristics are characterized by a certain type of basic move to alter the current tour. There are two important modifications to improve the tour. These are *Node Research* and *Edge Research*. *Node Research* consists of removing a node from the current tour and reinserting it at the best possible location. *Edge Research* consists of removing an edge from the tour and inserting at the best possible position.

3.3.3.2.1 K-opt move

K-opt move is one of the most widely used move operation. Here k is a number greater than *2*. The 2-opt move removes two edges from the tour and reconnects the two paths created. There is only one way to reconnect the two paths. Suppose that we have a tour (1 … i j ..k m… n) with n nodes, a 2-opt move will change the tour as follows (1… i k ….j m…n). The change in the cost function after the 2-opt operation is Δ = c(i,k) + c(j,m) - c(i,j) - c(k,m) (See Figure 3.7) [58, 1].

3-opt move works as 2-opt move but instead of removing 2 edges we remove three. There are eight possible 3-opt move.

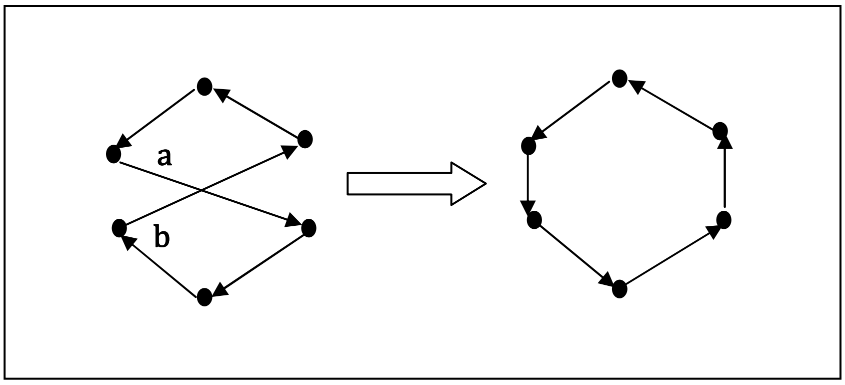

Figure 3.7: 2-opt move

3.3.3.2.2 Or-opt move

Or-opt move is a generalization of the edge insertion move where a path is removed from the tour and reinserted at the best possible location [59].

4 EVOLUTIONARY APPROACH TO SOLVE TSPPD

4.1 Mosheiov's Theorem

The basic idea of our evolutionary algorithm comes from the Mosheiov's Theorem. In his early paper Mosheiov [14] has proved the following result: Given a Hamiltonian tour ($i_1, i_2, ..., i_k, ..., i_n$,) covering all the pickup and delivery nodes but the depot, there exists at least one starting point i_k on this tour such that when the depot is inserted between i_k and i_{k+1} the resulting tour, ($i_1, i_2, ..., i_k, 0, i_{k+1}, ..., i_n$,) is feasible for the TSPPD. Using this result Mosheiov [14] has proposed a two stage Depot Insertion (DI) heuristic. In this heuristic first a Hamiltonian tour consisting of pickup and delivery points is found. Then a starting point i_k on this tour is found such that the depot is feasibly inserted right after it. Mosheiov [14] has noted that since the starting point that we will insert the depot is not necessarily unique, among all possible starting points, the one which yields the minimum tour length should be chosen.

4.2 Genetic Algorithm for TSPPD

According to the Mosheiov's proof, the stages of our approach can be detailed as follows;

Stage 1: Find a travelling salesman tour through all customers by using the GA.

```
begin
  generate initial population $P$
  improve initial population with 2-opt
  compute the fitness of each individual for $P$
      while stopping condition is not satisfied do
          for i = 1 to pop_size
```

```
            select two parents *p1* and *p2* from P according to a random selection rule
            cross *p1* and *p2* by using NNX or PMX and obtain an offspring *o1*
            improve *o1* with 2-opt
            mutate *o1* according to a predefined mutation rule
            **end for** // a new offspring population O is created
        compute the fitness of each individual for *O*
        replace the worst half of the P with the best half of the *O*
        replace the worst %10 parents by generating new parents according to the
        random generation of the initial population.
    **end while**
improve the best solution with 2-opt
**end**
```

Stage 2: Identify all feasible starting points where the depot can be inserted and choose the best one which yields the minimum tour length.

Stage 3: Use a local neighborhood search to improve the solution. (TUNING PHASE)

There are two different methods to improve the solution obtained from the second step. The first method (classical method) proposed by Gendreau et al. [15] is to use a local neighborhood search scheme. This method applied after the insertion of the depot employs the feasible arc exchange. The second method which is also used in our GA employs local search strategies by removing the depot, applying 2-exchanges operations and then reinserting the depot among the candidate positions. Given a feasible TSPPD tour (0,1,2,3,4,0) for a five node problem, when we apply the classical method on this TSPPD we obtain the following 5 neighbor solutions;

(i) (0,3,2,1,4,0) **(ii)** (0,1,3,2,4,0) **(iii)** (0,1,2,4,3,0)
(iv) (0,2,1,3,4,0) **(v)** (0,1,4,3,2,0)

Observe that we can disregard (i) and (v) because we have already checked that they are not better than the given original tour.

When we apply the second method of local search we first disconnect the depot from the original tour (0,1,2,3,4,0) and we get the Hamiltonian tour (1,2,3,4,1). When we apply 2-exchange operation to this tour we obtain the following neighbors:

(vi) (1,2,4,3,1) **(vii)** (1,3,2,4,1)

For each of these tours and the initial tour (1,2,3,4,1) we have 4 alternative locations to reinsert the depot. Hence, we obtain the following tours;

(viii)	(0,1,3,4,2,0)	**(xii)**	(0,1,4,2,3,0)	**(xvi)**	(0,1,4,3,2,0)
(ix)	(0,1,2,4,3,0)	**(xiii)**	(0,1,3,2,4,0)	**(xvii)**	(0,2,1,4,3,0)
(x)	(0,2,1,3,4,0)	**(xiv)**	(0,2,4,1,3,0)	**(xviii)**	(0,3,2,1,4,0)
(xi)	(0,3,1,2,4,0)	**(xv)**	(0,2,3,1,4,0)	**(xix)**	(0,1,2,3,4,0)

Observe that (i) and (xviii), (ii) and (xiii), (iii) and (ix), (iv) and (x), (v) and (xvi), are the same tours. Hence, we can say that the depot removal, 2-exchange and depot reinsertion operations include all the neighbor solutions obtained with classical local search method applied to the original tour (0,1,2,3,4,0).

In Figure 4.1, we present the depot insertion move. Black (white) nodes represent the deliveries (pickups). The vehicle capacity *Q=10.* In the left hand side the solution is (0,3,2,1,4,5,6,7,8,0) with the tour length *34.05* while in the right hand side the solution is (0,6,5,4,3,2,1,8,7,0) with the tour length *23.28.* There are *8* customers indicated by numbers on nodes and their demands are given in parentheses. Positive (negative) demands correspond to pickup (delivery) requests. Observe that we delete edges (0,8), (3,0), (7,6), (1,4) and add edges (1,8), (7,0), (0,6), (3,4).

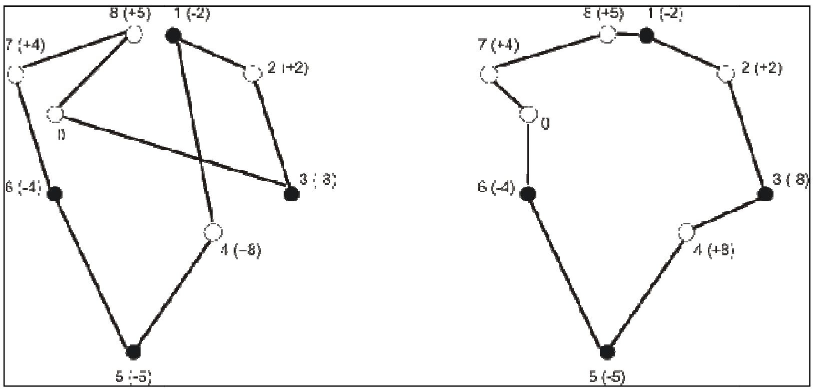

Figure 4.1: Depot Removal and Insertion.

4.3 The settings of the Genetic Algorithm

As a chromosome representation, we use a "path representation method". The fitness function of our GA is the tour length. GA converges more rapidly with smaller population but better results are obtained with larger populations. After investigating the effect of the population size on population quality and computation time, we have decided to use the population size of 30 and generate a pure random initial population and improve it by using a 2-opt. As a crossover operator we use NNX. The algorithm of our NNX can be formulated as follows;

```
begin
k=0
select a random city q as the first city of the next offspring o;
o(k) ← q
for k = 1 to N-1
find all neighbors of the o(k-1) in parents p1 and p2, respectively;
  if (all neighbors of o(k-1) in p1 and p2 have appeared in o,
      select the nearest city to the o(k-1) according to the cost matrix among all the
      unvisited cities as the next city q.
  else
      select the nearest city to the o(k-1) according to the cost matrix among all the
      unvisited neighbors of o(k-1) as the next city q.
end if
```

```
        $o(k) \leftarrow q$;
        end for
    end.
```

There are many tour improvement methods in the literature such as “k-opt”, “Or-opt”, “Lin Kernighan Procedure” etc… [14]. To accelerate the convergence of our GA we have employed 2-opt. 2-opt procedure proceeds by replacing two non adjacent edges *(b1,k1)* and *(b2,k2)* with two others *(b1,b2)* and *(k1,k2)*. The change is the cost function after the 2-opt operation is $\Delta = c(b1,b2) + c(k1,k2) - c(b1,k1) - c(b2,k2)$. In our model 2-opt is applied if and only if $\Delta \geq 0$. As a mutation operator we have used a simple 3-exchange mutation operator. [9] We randomly choose 3 cities and exchange their places in the tour. In case there are no improvements the tour is not changed.

In parent selection first of all we form a mating pool from the current population by replicating each chromosome. Then we randomly select the pairs of parents. The crossover operator generates one offspring from each pair. With newly generated offspring, we sort parents and offspring separately according to their fitness values. Then we carry the best half of offspring and parents to the next generation. We stop our GA if average fitness is exactly the same in two consecutive generations. In addition to this condition we use an iteration limit of 100 for offspring generation.

4.4 Computational Results

Our proposed GA is tested on standard instances taken from the literature. Two classes of instances that are generated by Gendreau et al. [15] are considered as our test bed. In the generation of test instances Gendreau et al. [15] have used a non-negative β parameter to indicate the percentage of the demand allocated to the pickup quantity. More specifically, given the demand quantity q_i of each node i of VRP test instance, the delivery quantity of that node is set to d_i and the pickup quantity p_i is determined according to the following rule:

$$p_i = \begin{cases} (1-\beta) * d & \text{if i is even} \\ (1+\beta) * \mathrm{d_i} & \text{if i is odd} \end{cases} \qquad \text{for } i = 1, \ldots, n$$

For each instance size, we set $\beta = 0.00, 0.05, 0.10, 0.20, \infty$. For $\beta = 0$ we have a TSP instance, and for $\beta = \infty$, the d_i and p_i values are uncorrelated. The first class of instances generated by Gendreau et al. [15] consists of 26 test problems with customer sizes varying from 6 to 261. The instances in the first class are derived from the symmetric VRP instances in the literature. The second class consists of randomly generated instances with $n = 25, 50, 75, 100, 150$ and 200. For each pair of n and β values, ten instances are generated by Gendreau et al. [15]. Note that for instances in the second class with $\beta = \infty$, d_i and p_i values are uncorrelated and generated uniformly random in $[1,100]$. Table 4.1 and Table 4.2 summarize the results of the first class of test instances. Each cell of these tables stand for the average value obtained with 26 instances on the first class. In Table 4.3 and Table 4.4 we present the results obtained with the instances on the second class, where each cell indicates the average results of ten instances for each pair of n and β values. In Table 4.1 and Table 4.3 we present the results obtained with several upper bounding approaches. The values reported are computed as $100 \times (z_{UB}/z_{TSP})$ where z_{UB} is the upper bound obtained with the corresponding algorithm and z_{TSP} is the optimum TSP solution value. In Table 4.2 and Table 4.4 we give the average CPU times. In Table 4.1, Table 4.2, Table 4.3 and Table 4.4, the first columns include β parameters. The second columns indicate the results obtained with the TS algorithm by Gendreau et al [15]. IOA columns include the results reported with the incomplete optimization algorithm by Hernández-Perez and Salazar-González [31]. The fourth columns indicate the results of the GA devised by Zhao et al [35]. The fifth columns display the results obtained with our GA. The last rows of the tables provide the averages of the corresponding columns. In order to compare the CPU time requirements of the algorithms we consider the performance evaluation and benchmarking approach proposed by Dongarra [60]. The author proposes to measure the power of a computer by its floating-point rate of execution in Mflops. Both the TS and IOA are run on an AMD 1.333 GHz PC with 649 Mflops. The GA is tested on a Pentium 1.33 GHz PC with 352 Mflops. Our experiments were performed on an Intel Pentium 2.2 GHz PC with 400 Mflops. The estimated powers of these computers, in terms of Mflops are taken from Dongarra [60]. We scale the average CPU times to the slowest computer, namely, the computer on which the TS and IOA are run. Considering the scaled CPU times reported in Table 4.5, we can say that the most efficient approach is the GA with Zhao et al., which is slightly better than our GA. However, for the instances in the second class, our

GA is the most efficient algorithm. Therefore, we can conclude that the proposed GA yields a comparable efficiency to the GA by Zhao et al. with a better accuracy. Recall that our GA performs the proposed tour improvement procedure at the final step for each Hamiltonian tour in the population and our GA outputs the best solution. For the sake of clarity, we should report that when our GA is run without the final tour improvement procedure we have obtained average bounds of *100.29* for the first class of test instances and *100.68* for the second class of test instances. However, the average bounds obtained with our EA using the tour improvement procedure are *100.04* and *100.13* for the first and second classes of test instances, respectively. This shows us the power of our tour improvement procedure, which is specially designed for the TSPPD.

Table 4.1: Best bounds obtained on the first class of instances.

β	**EA**	**GA**	**TS**	**IOA**
0,00	100,046	100,05	100,51	100,05
0,05	100,028	100,04	102,45	100,61
0,10	100,038	100,08	104,34	100,72
0,20	100,058	100,06	106,16	100,90
Average	100,04	100,06	103,37	100,57

Table 4.2: Average CPU times spent on the first class of instances.

β	**EA**	**GA**	**TS**	**IOA**
0,00	1,690	1,41	3,10	0,93
0,05	1,784	1,41	2,18	0,87
0,10	1,784	1,42	2,31	1,07
0,20	1,587	1,43	2,26	1,02
Average	1,71	1,42	2,46	0,97

Table 4.3: Best bounds obtained on the second class of instances.

β	N	EA	GA	TS	IOA
0,00	25	100,00	100,00	100,00	100,00
	50	100,00	100,00	100,20	100,00
	75	100,00	100,00	100,78	100,00
	100	100,01	100,10	101,27	100,10
	150	100,10	100,33	102,37	100,34
	200	100,41	100,36	103,13	100,35
0,05	25	100,00	100,00	107,36	102,07
	50	100,00	100,00	103,95	100,17
	75	100,04	100,12	110,13	100,83
	100	100,02	100,13	107,23	100,39
	150	100,11	100,26	108,72	100,35
	200	100,39	100,57	108,57	100,69
0,10	25	100,00	100,00	108,21	101,18
	50	100,00	100,00	106,34	100,47
	75	100,15	100,15	113,28	101,32
	100	100,15	100,12	111,53	100,50
	150	100,12	100,46	110,90	100,68
	200	100,44	100,72	111,31	100,76
0,20	25	100,05	100,00	107,28	102,59
	50	100,05	100,00	106,53	100,79
	75	100,06	100,12	114,25	101,77
	100	100,12	100,20	113,09	100,96
	150	100,17	100,60	111,69	101,02
	200	100,46	100,85	113,28	100,99
oo	25	100,00	100,00	105,64	101,32
	50	100,03	100,00	110,86	102,47
	75	100,10	100,13	111,86	100,91
	100	100,13	100,15	110,34	100,87
	150	100,29	100,39	112,85	100,63
	200	100,46	100,71	113,03	100,83
Average		100,13	100,22	108,20	100,85

Table 4.4: Average CPU times spent on the second class of instances.

β	N	EA	GA	TS	IOA
0,00	25	0,04	0,25	0,16	1,10
	50	0,25	0,51	0,75	1,20
	75	0,39	0,90	1,82	1,50
	100	1,00	2,25	3,24	1,60
	150	4,15	4,25	8,35	2,70
	200	7,32	6,93	15,27	3,40
0,05	25	0,05	0,24	0,18	1,00
	50	0,25	0,50	0,60	1,20
	75	0,40	0,89	1,32	1,10
	100	0,90	2,28	2,37	1,90
	150	3,49	4,15	5,46	2,80
	200	7,87	6,71	11,15	3,00
0,10	25	0,05	0,24	0,17	1,10
	50	0,27	0,51	0,63	1,10
	75	0,39	0,95	1,42	1,40
	100	0,93	2,25	2,60	2,30
	150	3,28	4,18	6,19	3,00
	200	7,98	6,92	11,93	3,10
0,20	25	0,06	0,24	0,20	1,10
	50	0,28	0,50	0,72	1,10
	75	0,42	0,88	1,44	1,50
	100	0,91	2,26	2,61	2,00
	150	3,27	4,30	6,01	2,50
	200	7,12	6,94	11,85	3,50
oo	25	0,07	0,25	0,18	1,00
	50	0,30	0,51	0,73	1,20
	75	0,44	0,92	1,42	1,20
	100	0,94	2,29	2,59	1,70
	150	3,62	4,22	5,78	2,40
	200	7,65	6,98	11,21	3,60
Average		2,14	2,51	3,95	1,91

Table 4.5: Scaled CPU times.

Instance Set	**EA**	**GA**	**TS**	**IOA**
First Class	1,050	0,77	2,46	0,97
Second Class	1,320	1,36	3,95	1,91
Average	1,19	1,07	3,21	1,44

5. CONCLUSION

In this dissertation we have focused on TSPPD which is an extension of the famous TSP. The problem involves two sets of customers: “Delivery Customers” and “Pickup Customers” We term the central warehouse as “Depot” and all delivery and pickup services are done by a single vehicle with a given capacity Q. There are two types of goods: delivery and pickup goods. A vehicle which departs from the depot fully loaded satisfies all the customer’s needs and returns back to the depot. The objective of the TSPPD is to minimize the total travelling distance. TSPPD is NP-hard combinatorial optimization problem. To the best of our knowledge there are not many publications on TSPPD. The aim of this thesis is to propose an efficient GA for the TSPPD.

The basic idea of our GA comes from the Mosheiov’s Theorem. According to his result when we construct a Hamiltonian tour covering all pickup and delivery customers except the depot, there exists at least one node i_k on this tour such that if we insert the depot between i_k and i_{k+1} the resulting tour is feasible. Based on this theorem we have developed a GA which consists of three stages. The first stage is to solve travelling salesman tour through all customers by using the genetic algorithm. The second stage is to find best starting point to insert the depot. Finally the third stage is to apply local neighborhood search which we term “Tuning Phase”. Computational experiments are reported on the standard test instances from the literature. Two classes of instances that are generated by Gendreau et al. [4], are considered as our test bed. The first class of instances consists of 26 test problems with customer sizes varying from 6 to 261. The second class consists of randomly generated instances with n= 25, 50, 75, 100, 150 and 200. We have compared our EA with 3 different algorithms; The TS algorithm by Gendreau et al [4], the incomplete optimization algorithm by Hernández-Perez and Salazar-González [6] and the GA devised by Zhao et al [13]. According to the experimental results, we can say that the proposed GA yields promising performance in terms of both accuracy and efficiency compared to these existing algorithms. We should also report that the usage of the tuning phase clearly improves the solution obtained after the second step. This shows us the power of our tour improvement procedure, which is specially designed for the TSPPD. As a further research we suggest the adaptation of the proposed GA to One – Commodity Pickup and Delivery TSP.

REFERENCES

[1] Gutin, G., Punnen, A., “The Traveling Salesman Problem and Its Variations”, *Kluwer Academic Publishers*, (2002).

[2] Michalewich, Z., David, F., “How To Solve It: Modern Heuristics”, *Springer Editions*, 189-224, (2000).

[3] Lawler, E.L., Lenstra, J.K., Rinnooy, Kan. A.H.G., Shymoys, D.B., “Traveling Salesman Problems”, *Wiley Interscience Series in Discrete Mathematics*, 1-15 (1985).

[4] Augustine, E., “Offline and Online Variants of the Traveling Salesman Problem”, *M.S. in Systems Science, Louisiana State University*, (2001).

[5] Dantzig, G.B., Fulkerson, R., Johnson, S.M., “Solution of a Large Scales Traveling Salesman Problem”, *Operations Research 2*, 393-410, (1954).

[6] Lin, S., ”Computer Solutions of the Traveling Salesman Problem”, *Bell System Technical Journal,* 44, 2245.2269, (1965).

[7] Öncan, T., Altınel, İ.K., Laporte, G., “A comparative analysis of several asymmetric traveling salesman problem formulations”, *Computers & Operations Research* 36, 637 – 654, (2009).

[8] Baykoç, Ö.F., İşleyen, S.K., ”An Efficient Iterated Local Search Algorithm for Traveling Salesman Problem”, *TEKNOLOJİ,* 10, 96-106 (2007).

[9] Aarts, E.H.L., Lenstra, J.K., (eds.), Wiley, J., “Local Search in Combinatorial Optimization”, *London*, 215-310, (1997).

[10] Ahuja R.K, Magnanti T.L., Orlin, J.B. “Network Flows: Theory, Algorithms, and Applications.” *ISBN 1000499012, Prentice Hall: New Jersey*, (1993).

[11] Freisleben, M., “New Genetic Local Search Operators for the Traveling Salesman Problem”, *OMEGA*: *The International Journal of Management Science*, 17, 289 – 295, (1989).

[12] Bryant, K., “ Genetic Algorithms and the Traveling Salesman Problem”, *Harvey Mudd College*, (2000).

[13] Perez, H.H., “Traveling Salesman Problems with Pickups and Deliveries”, *Serie Thesis Doctorates*, Laguna, (2005).

[14] Mosheiov, G., “Traveling Salesman Problem with Pickup and Delivery”, *European Journal of Operational Research, 79*, 299-310, (1994).

[15] Gendreau, M., Laporte, G., Vigo, D., “Heuristics for Traveling Salesman Problem with Pickup and Delivery”, *Computers & Operations Research 26*, 699-714, (1999).

[16] Anily, S., Bramel, J., “Approximation Algorithms for The Capacitated Traveling Salesman Problem with Pickups and Deliveries”, *Naval Research Logistics*, 46:654–670, (1999).

[17] Anily, S., Hassin, R., “The Swapping Problem”, *Networks,* 22:419–433, (1992).

[18] Anily, S., Gendreau M., and Laporte, G., “The Swapping Problem on a Line”, *SIAM Journal on Computing,* 29(1):327–335, (1999).

[19] Gendreau, M., Hertz, A., Laporte G., “An Approximation Algorithm for the Traveling Salesman Problem with Backhauls”, *Operations Research,* 45:639–641, (1997).

[20] Frederickson, G.N., Hecht, M.S., Kim, C.E.,”Approximation Algorithms for Some Routing Problems”, *SIAM Journal on Computing,* 7:178 – 193, (1978).

[21] Hernandez, P., Salazar, G., “The Multi-Commodity One-to-one Pickup-and-Delivery Traveling Salesman Problem”, *European Journal of Operational Research, 196,* 987–995, (2009).

[22] Savelsbergh, U., Sol, M., “The General Pickup and Delivery Problem.*”, Transportation Science,* (29):17–29, (1995).

[23] Pankratz, G., “A Grouping Genetic Algorithm for the Pickup and Delivery Problem with Time Windows”, *OR Spectrum,* 27: 21–41, (2005).

[24] Bent, R., Hentenryck, P.V., “A Two-stage Hybrid Algorithm for Pickup and Delivery Vehicle Routing Problems with Time Windows”, *Computers & Operations Research,* 33 875–893, (2006).

[25] Cordeau, J. F., Dell'Amico, M., Iori, M., “Branch-and-cut for the Pickup and Delivery Traveling Salesman Problem with FIFO Loading”, *Computers & Operations Research, www.elsevier.com/locate/cor,* (2009).

[26] Berbeglia, G., Cordeau, J.F., Laporte, G., “Dynamic Pickup and Delivery Problems”, *European Journal of Operational Research,* 202, 8–15 (2010).

[27] Berbeglia, G., Cordeau, J. F., Gribkovskaia I., Laporte G., “Static Pickup and Delivery Problems: A Classification Scheme and Survey” *TOP,* 15: 45–47, (2007).

[28] Baldacci, R., Hadjiconstantinou, E., Mingozzi, A., “An Exact Algorithm for the Traveling Salesman Problem with Deliveries and Collections”, *Networks 42, 26–41*, (2004).

[29] Anily, S., E.Mosheiov, G., “Traveling Salesman Problem with Delivery and Backhauls”, *Operations Research Letters 16*, 11-18, (1994).

[30] Renaud, J., Boctor, F.F., Ouenniche, J., “A Heuristic for the Pickup-and-Delivery Traveling Salesman Problem”, *Computers & Operations Research,* 27, 905-916 (2000).

[31] Hernandez, P., Salazar, G., “Heuristics for the One Commodity Pickup-and-Delivery Traveling Salesman Problem”, *Transportation Science*, 38, 245-255, (2004).

[32] Hernandez, P., Salazar, G., “A Branch-and-cut Algorithm for a Traveling Salesman Problem with Pickup and Delivery”, *Discrete Applied Mathematics* 145, 126-139 (2004).

[33] Hernandez, P. Salazar, G., “The One-commodity Pickup-and-Delivery Travelling Salesman Problem: Inequalities and Algorithms”, *Networks,* 50 (4), 258-272 (2007).

[34] Hernandez, P., Rodríguez, M.I., Salazar G.,”GRASP/VND Heuristic for the One-Commodity Pickup-and-Delivery Traveling Salesman Problem”, *Computers & Operations Research* 36, 1639-1645, (2009).

[35] Zhao, F., Sun, J.S., Li, S.J., Liu, W.M., “A Hybrid Genetic Algorithm for the Traveling Salesman Problem with Pickup and Delivery”, *International Journal of Automation and Computing,* 06 (1), 97-102, (2009).

[36] Zhao, F., Sun, J.S., Li, S.J., Liu, W.M., “Genetic Algorithm for the One-Commodity Pickup and Delivery Traveling Salesman Problem”, *Computers and Industrial Engineering,* 56, 1642-1648, (2008).

[37] Michalewich Z., “Genetic Algorithms + Data structures = Evaluations Programs”*, Springer Editions*, (1999).

[38] Goldberg, D.E.,”Genetic Algorithms in Search, Optimization, and Machine Learning”, *Addison Wesley Longman Inc.,* 1-57, (1989).

[39] Beasley, D., Bull, D.R., Martin, R.R.,”An Overview of Genetic Algorithms; Part 1, Fundamentals”, *University Computing 15(2),* 58-69 (1993).

[40] Reeves, R.C., Rowe, J.E., “Genetic Algorithms – Principles and Perspectives”, *Kluwer Academic Publishers*, 2-91, (2002).

[41] Janikow, C.J., Michalewich, Z., “An Experimental Comparison of Binary and Floating Point Representations in Genetic Algorithm”, *The Fourth International Conference on Genetic Algorithms*, 31-36, (1991).

[42] Beasley, D., Bull D.R., Martin, R.R.,”An Overview of Genetic Algorithms; Part 2, Fundamentals”, *University Computing 15(4),* 170-181 (1993).

[43] Goldberg, D.R.,” Optimal Initial Size For Binary – Coded Genetic Algorithms “, *TSGA Report 85001, University of Alabama, Tuscaloosa* (1985).

[44] Goldberg, D.R.,” Sizing Populations for Serial and Parallel Genetic Algorithms”, *3rd International Conference on Genetic Algorithms, Morgan Kaufmann, San Mateo*, CA, 70-79 (1989).

[45] Reeves, C.R.,”Modern Heuristic Techniques for Combinatorial Problems”, *Blackwell Scientific Publications, Oxford, UK*, (1993).

[46] Schmitt L.J., Amini, M.M., “Performance Characteristics of Alternative Genetic Algorithmic Approaches to the Travelling Salesman Problem Using Path Representation: An Empirical Study”, *European Journal of Operational Research,* 108 (3) 551-570, (1998).

[47] Ahuja R.K., Orlin J. B., "Developing Fitter GAs", *INFORMS Journal on Computing,* 9, 251 – 253, (1997).

[48] Whitley, D., "The GENITOR Algorithm and Selection Pressure: Why Rank-based Allocation of Reproductive Trials is Best", *Los Altos, CA,* 116-121, (1989).

[49] Goldberg, D.E., Deb, K., "A Comparative Analysis of Selection Schemes Used in Genetic Algorithm", *Foundation of Genetic Algorithms, CA,* 69-93, (1989).

[50] Holland, J.H., "Adaptation in Natural and Artificial Systems", *University of Michigan Press, Ann Arbor*, (1975).

[51] Forrest, S., Mitchell, M., "Relative Building-Block Fitness and the Building-Block Hypothesis", *Foundations of Genetic Algorithms 2, CA,* (1993).

[52] Garey, M.R., Johnson, D.S., "Computers and Intractability: A Guide to the Theory of NP-Completeness", *W. H. Freeman & Co. New York, NY, USA,* (1979).

[53] Goldberg, D.E., Lingle, R., "Alleles, Loci and the TSP", *First International Conference on Genetic Algorithms, Lawrence Erlbaum Associates, Hillsdale, NJ* (1985).

[54] Davis, L., "Applying Adaptive Algorithms to Epistatic Domains", *Proceedings of the International Joint Conference on Artificial Intelligence,* 162-164, (1995).

[55] Oliver, I.M., Smith, D.J., Holland, J.R.C., "A Study of Permutation Crossover Operators on the Traveling Salesman Problem", *2nd International Conference on Genetic Algorithms, Lawrence Erlbaum Associates, Hillsdale, NJ,* 224-230, (1985).

[56] Nagata, Y., "Fast EAX Algorithm Considering Population Diversity for Traveling Salesman Problems", *EvoCOP 2006, LNCS 3906,* 171-182, (2006).

[57] Nagata, Y., “New EAX Crossover for Large Instances”, *PPSN IX,, LNCS 4193,* 372-381, (2006).

[58] Nilsson, C., “Heuristics for the Traveling Salesman Problem”, PhD Thesis *Linkoping University,* (2002).

[59] Or, I., “Traveling salesman-type combinatorial problems and their relation to the logistics of regional blood banking.” *PhD thesis, Northwestern University, Evanston,* (1976).

[60] Dongarra, J.J., “Performance of Various Computers using Standard Linear Equations Software” (*Linpack Benchmark Report), Technical Report CS-89-85, University of Tennessee, Computer Science Department, USA* (2009).

APPENDIX: C++ Coding

```
#include <iostream>
using namespace std;
#include <fstream>
#include <math.h>
#include <time.h>
#include "Header_1.h"
int main(){
int i,j,N,p,h;
double x,c,l,a;
double bas,son,best,time,skor,toplamtime,toplamskor,besttime,D,beta;
char dos[50];
char atla[3000];
double **K=new double*[M];
for(i=0;i<M;i++)        K[i]=new double[N+2];
double **DIS=new double*[N+2];
for(i=0;i<N+2;i++) DIS[i]= new double[N+2];
double **D=new double*[N+2];  for(i=0;i<N+2;i++) D[i]= new double[2];
double **B=new double*[N+2];  for(i=0;i<N+2;i++) B[i]= new double[2];
double **INS=new double*[N+4];   for(i=0;i<N+2;i++) INS[i]= new double[N+4];
double **IN=new double*[N+2];  for(i=0;i<N+2;i++) IN[i]= new double[N+2];
double **X=new double*[N+1];  for(i=0;i<N+1;i++) X[i]= new double[5];
int *BEST=new int [N+4];
double *U=new double [SDD];
int *V=new int [SDD];
double *UU=new double [SDD];
int *VV=new int [SDD];
```

```
D=0;
skor=1000000000;
besttime=100000000;
toplamtime=0;
toplamskor=0;
best=0;
int q;
for ( q=0;q<SDD;q++ ) {
for ( h=0;h<T;h++) {
bas=cpu_time();
GENETIK( N,M,O,BEST,IN,best,q); //
best= UZ(N,BEST,IN);
TURINSERT( N,BEST,X,INS,c,l);
POSTTWOP(N,BEST,X,INS,c,l);
best= UZ(N+1,BEST,INS);
U[q]=best;
V[q]=q+1;
son=cpu_time();
time = son - bas;
UU[q]=time;
VV[q]=q+1;
if (best<skor){
skor=best;
        }
if (time<besttime){
        besttime=time;
        }
        } // end of for T
        }
heapsort(UU,VV,0,SDD-1);
heapsort(U,V,0,SDD-1);
toplamtime=0;
toplamskor=0;
```

```
for (q=0;q<Q;q++){
        toplamtime=toplamtime+UU[q];
        toplamskor=toplamskor+U[q];
        }
toplamtime=toplamtime/(Q);
toplamskor=toplamskor/(Q);
skor=U[0];
besttime=UU[0];
} // p dongusunun sonu
return 0;
}

void GENETIK( int nn, int mm, int oo,  int *&BEST, double **&INS, double best, double
sdd){
int i,j,k,t,q,a,s,l;
srand(sdd);
int **P=new int*[mm];  //populasyon matrisi
for(i=0;i<mm;i++)     P[i]=new int[nn+2];
int *Y=new int[nn+2];
double *U=new double[nn+2];
double *UU=new double[mm+2];
int *VV=new int[mm+2];
POP(nn,mm,P,k,sdd);

for (i=0;i<mm;i++){
for (j=0;j<nn;j++){
                Y[j]=P[i][j]
 }
 TWOP1(nn,Y,INS);
for (j=0;j<nn;j++){
P[i][j]=Y[j];
}
UU[i] = UZ(nn,Y,INS);
```

```
VV[i] = i+1;
}
heapsort(UU,VV,0,mm-1);

i=VV[0];

for (j=0;j<nn;j++){
BEST[j]=P[i-1][j];
}
best=UU[0];
for (t=0;t<oo;t++){
cout<<"hata4";
ESNX(nn,mm,P,UU,VV,INS,i);
for (i=0;i<mm;i++){
for (j=0;j<nn;j++){
Y[j]=P[i][j];
        }
TWOP1(nn,Y,INS);
MUTASYON(nn,Y,INS,3);
        UU[i] = UZ(nn,Y,INS);
        VV[i] = i+1;
for (j=0;j<nn;j++){
                P[i][j]=Y[j];
        }
}
heapsort(UU,VV,0,mm-1);

  if((UU[0]<best)){
                i=VV[0];
                for (j=0;j<nn;j++){
                BEST[j]=P[i-1][j];
                }
                best=UU[0];
}
}
for (i=mm-3;i<mm;i++){
                q = VV[i];
                for(j=0;j<nn;j++){
                U[j]=0;
                }
                for (k=0;k<nn;k++){
                s=0;
                l=rand()%(nn-k)+1;
                for (a=0;a<nn;a++){
                if (U[a]==0) s=s+1;
                if (s==l)
                {
                          U[a]=1;
                          P[q-1][k]=a+1;
                          break;
                }
                }
                }
                }
for (i=0;i<mm;i++){
for (j=0;j<nn;j++){
                Y[j]=P[i][j];

        }
UU[i] = UZ(nn,Y,INS);
VV[i] = i+1;
}
heapsort(UU,VV,0,mm-1);
}
TWOP2(nn,BEST,INS);
delete [] Y;
delete [] UU;
delete [] VV;
delete [] U;
for (i=0;i<mm;i++) delete [] P[i];
delete [] P;
}
```

BIOGRAPHICAL SKETCH

Volkan ÇINAR was born in Ağrı, TURKEY on July 4, 1982. He graduated from Galatasaray High School in 2001. He received his B.S. degree in Industrial Engineering in 2005 from Galatasaray University, İstanbul, TURKEY.