MARMARA UNIVERSITY
INSTITUTE FOR GRADUATE STUDIES
IN PURE AND APPLIED SCIENCES

SOLUTION APPROACHES FOR SEQUENCE DEPENDENT TRAVELING SALESMAN PROBLEM

SAMET TONYALI

MASTER THESIS

Department of Computer Engineering

ADVISOR

Asst. Prof. Ali Fuat ALKAYA

ISTANBUL, 2013

MARMARA UNIVERSITY
INSTITUTE FOR GRADUATE STUDIES
IN PURE AND APPLIED SCIENCES

SOLUTION APPROACHES FOR SEQUENCE DEPENDENT TRAVELING SALESMAN PROBLEM

SAMET TONYALI

(524111001)

MASTER THESIS

Department of Computer Engineering

ADVISOR

Asst. Prof. Ali Fuat ALKAYA

ISTANBUL, 2013

PREFACE

To the most beautiful and the most lovely physician I have met so far, Pınar…

December, 2013 **Samet TONYALI**

ACKNOWLEDGMENT

I would like to thank my academic supervisor, Dr. Ali Fuat ALKAYA, for his encouragement, kindness, support, patience and valuable guidance throughout this study.

To my father, my mother and my brother, many thanks for your continuous support and understanding.

Many thanks to my colleagues in the Computer Science and Engineering Department for their sincere friendship and making my time spent at the Marmara University an enjoyable experience.

Lastly, I want to thank Hüseyin DEMİRKALE because of his contribution on GAMS coding.

I also gratefully acknowledge the support of Turkish Scientific and Technological Research Institution (TÜBİTAK) in this research, through the Master of Science Scholarship Program 2210.

December, 2013 **Samet TONYALI**

TABLE OF CONTENTS

PAGE

ÖZET

Sıraya Dayalı Seyyar Satıcı Problemi için Çözüm Yaklaşımları

Sıraya Dayalı Seyyar Satıcı Problemi (SDSSP), Seyyar Satıcı Problemi'nin (SSP) genellemesi olarak tanımlanmış bir birleşimsel eniyileme problemidir. Bu problem, Baskılı Devre Kartı (BDK) üretiminde çokça kullanılan iki önemli dizgi makine tipinin eniyilenmesi sırasında ortaya çıkmıştır. SDSSP'yi SSP'den ayıran fark şu şekilde ifade edilebilir: SDSSP'de bir noktadan diğerine gitmenin maliyeti sadece noktalar arasındaki uzaklığa değil, gezinme sırasında sonradan gelecek *k* tane noktanın hangileri olduğuna da bağlıdır.

Bu çalışmada, Benzetimli Tavlama (BT), Yapay Arı Kolonisi (YAK) ve Göç eden Kuşlar (GK) isimli üç üstsezgiselden yararlandık. Bu üstsezgiseller 10 komşu fonksiyonu ile test edildi. Yaptığımız sayısal çalışmada, beş ayrı kümeden oluşan testler yaptık. İlk olarak, her bir üstsezgisel için en iyi parametre kombinasyonunu elde ettik. Daha sonra her bir üstsezgiselin en iyi başarım gösterdiği komşu fonksiyonu belirlemek için testler yaptık. Bu testlerden sonra üstsezgisellerin başarımlarını karşılaştırmak için testler yaptık. BT, GK ve YAK'tan daha iyi başarım gösterdi. Bu testleri, problem özgün önemli bir parametrenin problemin karmaşıklığını hem çözüm kalitesi hem de çalışma zamanı açısından nasıl etkilediğini ortaya çıkarmak için yaptığımız testler takip etti ve bu parametrenin değerinin artmasıyla üstsezgisellerin başarımının düştüğünü gözlemledik. Burdan yola çıkarak SDSSP'nin SSP'den çok daha zor bir problem olduğu sonucuna vardık. Son olarak, problemin matematiksel modelini, GAMS'i kullanarak programladık ve bazı küçük ölçekli problem örnekleri için en iyi üst sınır çözümlerini elde ettik. Üstsezgisellerin aynı örnekler için elde ettiği çözümlerle karşılaştırdık ve üstsezgisellerin performansının problemin büyüklüğü arttıkça en iyi üst sınır çözümleriyle kıyaslanabilir olduğunu gözlemledik. Yaptığımız testler sonucunda, en iyi üst sınır çözümlerini elde etmek için gereken zamanın, problemin büyüklüğüyle üssel olarak arttığını gözlemledik. Bunun bir sonucu olarak, SDSSP ve benzeri problemlerin orta ve büyük ölçekli örneklerinin çözümlerini elde etmede, bu üstsezgiselleri kullanmanın iyi bir tercih olduğu sonucuna vardık.

Aralık, 2013 **Samet TONYALI**

ABSTRACT

Solution Approaches for Sequence Dependent Traveling Salesman Problem

The Sequence Dependent Traveling Salesman Problem (SDTSP) is a combinatorial optimization problem defined as a generalization of the TSP. It emerged during optimization of two kind of commonly used placement machines for production of printed circuit boards. The difference between SDTSP and TSP is that the cost incurred by transition from one point to another is dependent not only the distance between these points but also subsequent *k* points.

In this study, we applied Simulated Annealing (SA), Artificial Bee Colony (ABC) and Migrating Birds Optimization (MBO) to solve real-world and random SDTSP instances. The metaheuristics were tested with 10 neighbour functions. In our computational study, we conducted five sets of computational experiments. Firstly, we obtained best parameter value combination for each metaheuristic. Secondly, we conducted experiments so as to determine best performing neighbour function for each metaheuristic. Thirdly, metaheuristic comparison tests were conducted. As a result, SA outperformed MBO and ABC. These tests were followed by the tests conducted to reveal how a parameter peculiar to the problem affects the complexity of the problem both in terms of run time and solution quality. As this parameter value increases, performances shown by all metaheuristics decrease. Therefore, we concluded that SDTSP is much more challenging than TSP. Finally, we programmed the mathematical model of the problem by means of GAMS and we obtained the best upper-bound solutions of some small-scale problem instances. These solutions were compared with those obtained by the metaheuristics and we observed that solution quality performance of the metaheuristics are comparable to the best upper-bound solutions returned from GAMS. In addition, we observed that the time required for obtaining the best upper-bound solution of an SDTSP instance increases exponentially with the problem size. Consequently, we deduced that using these metaheuristics to obtain solutions for medium- and large-scale instances of SDTSP and similar problems is a good choice.

December, 2013 **Samet TONYALI**

SYMBOLS

k : Number of points whose turret time must be taken into account

i : Point *i*

j : Point *j*

d(i, j) : Distance between points *i* and *j*

$\boldsymbol{i_x}$: x coordinate of point *i*

$\boldsymbol{i_y}$: y coordinate of point *i*

$\boldsymbol{j_x}$: x coordinate of point *j*

$\boldsymbol{j_y}$: y coordinate of point *j*

CF : Cost Function

v : A predefined speed value

$\boldsymbol{t_{ij}}$: Time between completions of consecutive placements at points *i* and *j*

$\boldsymbol{t^0}$: Component placement time

tt/ft : Turret/Free time

N : Total number of points to be visited

p : Visiting order or visiting position

$\boldsymbol{C_{ijp}}$: Cost of travel from point *i* to point *j* when point *j* is visited in p^{th} position

$\boldsymbol{D_{ijp}}$: Dominating factor in C_{ijp} definition

$\boldsymbol{x_{ip}}$: A binary variable that shows whether point *i* is visited in p^{th} position

$\boldsymbol{w_{ijp}}$: A binary variable that shows whether point *j* is visited in p^{th} position after point *i* is visited in (*p* – 1)st position

T : Temperature

a : Temperature decrease ratio

R : Initial number of iterations for inner loop of SA

b : Expansion coefficient for *R*

CS **:** Colony size

d **:** A constant which is used to calculate *limit* value in ABC

D **:** Number of dimensions of a vector that represents a point in a problem

n **:** Number of initial solutions in MBO

m **:** Number of tours in MBO

p **:** Number of solutions to be generated from a solution in MBO

x **:** Number of solutions to be shared with the solution that follows in MBO

ABBREVIATIONS

TSP : Traveling Salesman Problem

SDTSP : Sequence Dependent Traveling Salesman Problem

NP : Non-Deterministic Polynomial Time

SA : Simulated Annealing

ABC : Artificial Bee Colony

MBO : Migrating Birds Optimization

GAMS : General Algebraic Modeling System

PCB : Printed Circuit Board

MST : Minimum Spanning Tree

TAT : Total Assembly Time

RT : Run Time

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

An optimization problem is interested in the finding the best solution among all feasible solutions. A combinatorial optimization problem is a kind of optimization problem whose feasible solutions are discrete. Most of combinatorial optimization problems cannot be solved in polynomial time by exhaustive search and solving an instance of this sort of problems requires non-deterministic polynomial time (NP). Two most common combinatorial optimization problems are traveling salesman problem (TSP) and minimum spanning tree (MST). In this thesis, we studied a new generalization of TSP. We will give details of the problem in the next section.

So far scientists have not been able to propose an algorithm that gives an optimum solution in polynomial time for this kind of problems. Instead, they developed and applied problem-specific heuristics and metaheuristics, which can be applied to all optimization problems with some small modifications.

A heuristic is not an algorithm because heuristics are not guaranteed to get an optimum solution; they are designed to give an acceptable solution in a reasonable time. However, we will use these terms interchangeably throughout this thesis.

It is agreed that one of common solution methods for the combinatorial optimization problems is to utilize metaheuristics. Metaheuristics are the heuristic methods that can be applied for various problems with small modifications. Some of the well-known metaheuristics having been used to solve combinatorial optimization problems are genetic algorithms, simulated annealing, ant colony optimization, tabu search and iterated local search. In this study, we considered three of these metaheuristics to solve a recently defined problem, sequence dependent traveling salesman problem (SDTSP): simulated annealing (SA) [4], artificial bee colony (ABC) [5] and newly proposed migrating birds optimization (MBO) [6].

Metropolis et al. [7] introduced a simple algorithm that simulates the annealing process in metallurgy. Annealing is a technique involving heating a solid to a high temperature (melting) and then controlled cooling it (freezing) very slowly to provide it to reach thermal equilibrium at each temperature. Thermal equilibrium is reached when large changes in energy do not occur at this temperature anymore [40]. This algorithm was latterly named as SA by Kirkpatrick et al. [4]. The main goal is to find a configuration that minimizes the energy of the system. In this analogy; system, configuration and energy correspond to a combinatorial optimization problem, solution generated for this problem and cost of this proposed solution, respectively.

ABC is a nature-inspired and population based optimization algorithm simulating intelligent foraging behavior of a honey bee swarm. The algorithm was proposed by Karaboga and Basturk. In their original study they used the algorithm for optimizing multimodal (a function which has two or more local optima) and multivariable functions [5].

There are three groups of bees in a honey bee hive: Employed bees, onlookers and scouts. Employed bees exploit food sources surrounding the hive. Each employed bee has her own food source. They collect nectar from the sources and transport it to the hive. When they arrive to the hive, they make waggle dance on dance floor in the hive. Onlookers waiting on the dance floor watch each dance performed by each employed bee and make an inference about where the food sources are and amount of nectar in the food sources. Then, onlookers decide food sources to be exploited. The more nectar a food source has, the more chance the food source has for being chosen by an onlooker bee. After a while, the food sources having been exploited are exhausted. In such a case, associated employed bee abandons her food source and becomes a scout. Then, it starts to explore for a new food source randomly. In the analogy, a food source represents a solution for the problem to be optimized. The amount of nectar is the fitness value of the solution.

MBO is a newly proposed, population-based neighbourhood search technique inspired from the V formation flight of the migrating birds which is proven to be an effective formation in energy minimization. The algorithm was proposed by Duman et al. [6] and was applied to solve quadratic assignment problem arising from printed circuit board assembly workshops. In the analogy, initial solutions correspond to a flock of birds. Likewise the leader bird in the flock, a leader solution is chosen and rest of the solutions is divided into two parts. Each solution generates a number of neighbour solutions. This number is a determiner value on exploration and it corresponds to the speed of the flock. The higher this value, the more detailed the flock explores its surroundings.

The reason why we have chosen ABC and MBO is that they are recently proposed and unexploited metaheuristics. Besides, all these metaheuristics rely on neighbourhood search. In our computational work, we compared performances of these metaheuristics in terms of solution quality and run time. That is, firstly, we carried out some experiments on real-world data set to fine tune parameters of the algorithms. Secondly, we run the metaheuristics with different neighbour functions on real-world data set and determined best-fit functions for each metaheuristic. Thirdly, we compared the performances of these metaheuristics on real-world and random data set. Fourthly, we run the metaheuristics on real-world data with various k values, which is a parameter peculiar to the problem (See next section for the definition of k value). Finally, we compared the solutions obtained by the metaheuristics with the best upper-bound solutions returned from the general algebraic modeling system (GAMS) in terms of run time and solution quality on small-scale problem instances. GAMS is a high-level modeling system for mathematical programming and optimization [8]. We use "best upper-bound solutions returned from GAMS" expression because GAMS does not guarantee that it will return exact solution, which is also explicitly stated in its output statement.

The remaining of the thesis is organized as follows: In Section 2, we introduced our new problem and gave the details such as its origin and some technical definitions which makes the problem clearer. Then, in Section 3, we gave algorithmic details and literature survey of the metaheuristics mentioned above while, in Section 4, we introduced neighbor functions utilized in this study. In Section 5, we elaborated the characteristics of test instances, and gave the specification of placement machines where the instances were obtained and other details about experiments. We analyzed and discussed on test results carried out both on random and real data sets in Section 6. In the last section, we presented with a summary of the study and its main conclusions.

2. SEQUENCE DEPENDENT TRAVELING SALESMAN PROBLEM

SDTSP is a recently introduced combinatorial optimization problem as a generalization of TSP [9]. The difference between the SDTSP and TSP is that the cost incurred by transition from one vertex to another depends not only the distance between these points but also on the choice of subsequent *k* vertices. Therefore, solution methods for the TSP are not guaranteed to generate optimum/near-optimum solutions for the SDTSP[1].

This problem emerged during optimization of two kind of commonly used placement machines for production of printed circuit boards (PCB) [2, 3]. These are chip shooter and chip mounter placement machines. In this study, we considered a specific chip mounter placement machine: TDK brand, model RX-5A. In the following subsection we give the details of operation principles of chip mounters so that the reader can easily understand the SDTSP.

2.1. Operation Principles of Chip Mounters

A chip mounter has a rotational turret consisting of multiple heads which are responsible for the pickup of components from the component magazine and placing them on the PCB. The component magazine (also called the feeder mechanism) where the component tapes are installed is a circular structure behind the machine and it is stationary. The rotational turret, which has 72 heads, collects the components to its heads (one to each head) in the placement order, from the component tapes placed in the component magazine, while rotating in clockwise direction (See Figure 2.1). Even though the machine has actually 72 heads, a few are depicted in the figure for the sake of clarity. The head reaching the 'placement location' over the board moves down and makes the placement to the precise location on the PCB which is pre-aligned with the placement position by the board carrier actions. Thus, each head makes the placement exactly at the same position and the alignment of the placement point is made by the board carrier through its simultaneous and independent movements (leading to a

1 Most part of this section was cited from [2].

Chebyshev distance measure) in the x–y plane.

These machines may handle component types of different weights (each head is equipped with three suction nozzles compatible with different weight categories). When any of the heads picks up a heavier component type, the rotation speed is reduced. For the particular machine considered, there are four discrete speed values corresponding to different component weight categories. These are 0.20, 0.23, 0.33 and 0.40 sec per rotational movement of distance of one head, ordered from the lightest to the heaviest components. On the other hand, the board carrier movement speed in both x and y directions is 120 mm per second which could practically be taken as constant (i.e. no acceleration or deceleration).

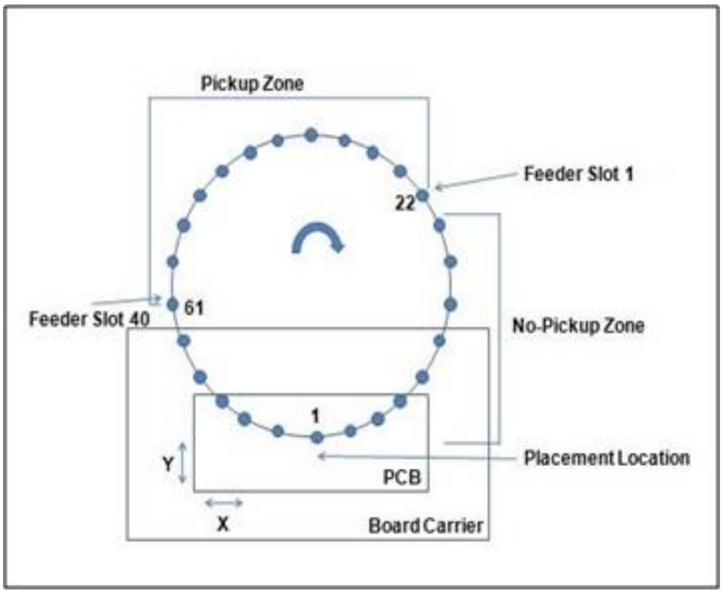


Figure 2.1 – TDK chip mounter placement machine **[10]**

The component tapes are mounted to a component magazine behind the machine and this whole system is stationary. If the placement heads are numbered from 1 to 72 in counter clockwise manner with number 1 being the placement head, up to 120 component tapes (depending on their width) can be installed to the feeder slots along heads 22 and 61 (pickup zone). The pickup times of the components from the magazine is completely determined by the order of their placement. Once the placement sequence is determined and the position of each component type in the magazine is entered to the computer, the component type that each head arriving at the placement location carries is known and fixed a priori. In other words, what component type each and every head should pick up is known. Then, during the rotational movement of the turret, the head(s) situating over the correct component tape move down and pick a component.

To summarize, a placement cycle of such a machine can be itemized as follows:

Item 1:

- turret rotates and the next placement head comes over the PCB
- board carrier aligns the new placement point under the placement head
- placement heads rotate if necessary to align the suction nozzle carrying the component to be placed

Item 2:

- the placement head moves down, makes the placement and moves up
- head(s) in the pickup zone that are above the appropriate component tapes move down and pick up components.

In the above cycle, the actions under each item are performed concurrently, and there is a sequential order between items one and two.

If the minimization of the PCB assembly time is defined as the master problem, the placement sequence and the feeder configuration are the two determinants of the assembly time. In other words, the assembly time of a PCB can be calculated as a function of the placement sequence and the feeder configuration.

Actually, what makes the master problem complicated is the varying rotational speed of the turret due to the different component weights. For the moment assume that all components to be populated on the PCB are at the same weight category (i.e. the turret has a uniform rotational speed throughout the whole assembly process). In this case, the master problem becomes equivalent to the placement sequencing problem (discussed below) and can exactly be modeled as a TSP with Chebyshev distance measure. Since Chebyshev measure is used in the considered machine, the distance between two points i and j, $d(i, j)$, is defined as $\max\{| i_x - j_x |, | i_y - j_y |\}$, where i_x and i_y represent the x and y-coordinates of point i, respectively. Cost is calculated in time units; hence it is defined as in (2.1).

$$CF(i, j) = \frac{d(i, j)}{v} \tag{2.1}$$

where v is a predefined speed value.

If t_{ij} is defined as the time between the completions of consecutive placements at points i and j, then it can be calculated by using the following definitions.

t^0= component placement time (including placement head moving down and up time),

tt = turret time (turret rotation time required for the next placement head to arrive over the PCB),

Then, t_{ij} is given in (2.2).

$$t_{ij} = t^0 + \max(CF(i, j),\ tt) \tag{2.2}$$

In many cases, the component placement time denoted by t^0 is small (in our case it is 0.05 sec) and independent of the component placement sequence, so it may be dropped from the above expression without loss of generality. The turret time denoted by tt is also known as the free time since during this time interval the board carrier is free to move to any placement point without increasing the cost (turret time and free time will be used interchangeably throughout this thesis). In other words, the points at a Chebyshev distance of free time or less can be taken as equidistant points. If all components are at the same weight category, then there would be no feeder configuration problem. That is, the positioning of the component types in the magazine would not be an issue, since they would be arriving at the placement location in the order of placement sequence regardless of their actual positions. In fact, for the assembly of a single PCB, the feeder configuration could be important, since it influences the arrival time of the first component to the placement location. However, in mass production environments, the components arrive at the placement location continuously and this duration is certainly negligible in the total production time.

Consequently, if tt is constant, the master and the placement sequencing problems become equivalent. However, in our case tt can take four possible values where its value is determined by the heaviest component type being carried by one of the heads of the turret. In other words, right after the placement of the component on head number 1 (See Figure 2.1), the rotational speed value corresponding to the heaviest component being carried by the heads numbered 2–61 will be the current tt value for the next rotational step of the turret. On the other hand, the type of components being carried by heads 2–61, depends both on the placement sequence and the positioning of these component types in the component magazine. This means that the classical TSP formulation for the PCB assembly time minimization problem cannot be valid anymore since it needs a constant distance between every pair of points. Furthermore, two sub-problems, namely the placement sequencing and feeder configuration problems arise. Since the feeder configuration problem is out of scope of this thesis we assume that a feeder configuration is given.

Given a feeder configuration, the objective of the placement sequencing problem is to find a placement sequence of the components so that the assembly process is completed in the shortest time possible. Similar to the master problem, the time between two consecutive placements is determined by t_{ij}; however in this case, tt plays an important role in the determination of t_{ij}. The dependency of tt to feeder configuration makes the placement sequencing problem dependent on the feeder configuration problem. On the other hand, as explained above, in calculating the value of tt, the information of 'if j is placed after i' is not sufficient and in addition of j we had to know the next 59 components to be placed. In such a case, the basic requirement of 'constant distances between pairs of cities' is violated and thus the placement sequencing problem cannot be modeled as a TSP, turning out to be a more difficult problem even to formulate. We give the complete final formulation of SDTSP in the next subsection; however please refer [9] so as to find out how this formulation evolves.

2.2. Formulation of SDTSP

In Table 2.1, we give the complete formulation of the SDTSP and the symbols used in the formulation are given in Table 2.2.

Table 2.1 – The complete formulation of SDTSP

$$\min \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{p=1}^{N} D_{i\,j\,p}$$

s.t.

$$D_{i\,j\,p} - w_{i\,j\,p}\; CF(i,\, j) \geq 0,\; i, j, p = 1,\ldots,N\;,$$

$$D_{i\,j\,p} - w_{i\,j\,p}\sum_{l=1}^{N}\sum_{m=0}^{k} ft_l\, x_{l\;p+m} \geq 0,\; i, j, p = 1,\ldots,N\;,$$

$$\sum_{p=1}^{N} x_{i\,p} = 1,\; i = 1,\ldots,N\;,$$

$$\sum_{i=1}^{N} x_{i\,p} = 1,\; p = 1,\ldots,N\;,$$

$$\sum_{i=1}^{N} w_{i\,j\,p} = x_{j\,p},\; j, p = 1,\ldots,N\;,$$

$$\sum_{j=1}^{N} w_{i\,j\,p} = x_{i\;p-1},\; i, p = 1,\ldots,N\;,$$

$$x_{i\,p} \in \{0,\, 1\},\; i, p = 1,\ldots,N\;,$$

$$w_{i\,j\,p} \in \{0,\, 1\},\; i, j, p = 1,\ldots,N\;,$$

$$w_{i\,j\,p} = 0,\; i = j\;,$$

$$D_{i\,j\,p} = 0,\; i = j\;,$$

$$D_{i\,j\,p} \geq 0 \textit{ and a real number}\,.$$

Table 2.2 – Legend for Table 2.1

i: point *i*
j: point *j*
N: Total number of points to be visited
p: the visiting order or visiting position
k: the number of points whose turret time must be taken into account

$D_{i\,j\,p}$: In the original formulation, $C_{i\,j\,p}$ is used. Since this definition includes nonlinearity, we introduced $D_{i\,j\,p}$ as the dominating factor in the $C_{i\,j\,p}$ definition.

$C_{i\,j\,p}$: The cost of travel from point *i* to point *j* when point *j* is visited in p^{th} position.

$$C_{i\,j\,p} = w_{i\,j\,p}\; max\left\{CF(i,\, j),\, max_{m=p}^{p+k}\left\{\sum_{l=1}^{N} ft_l\;\; x_{l\,m}\right\}\right\}.$$

$x_{i\,p} = 1,$ *if node i is visited in* p^{th} *position;*

$0,$ *Otherwise.*

$w_{i\,j\,p} = 1,$ *if node j is visited in* p^{th} *position after node i visited in* $(p-1)$*st position;*

$0,$ *Otherwise.*

2.3. An Illustrative Example

The following example illustrates the difference between the classical TSP and the SDTSP. In this example, the number of points whose free time should be taken into account is four (k = 4). The coordinates and free times (ft) of the points are given in Table 2.3 below:

Table 2.3 – Example configuration

Point	x	y	ft
A	2	7	2
B	4	6	2
C	6	7	2
D	8	5	2
E	7	5	2
F	4	5	2
G	8	1	2
H	4	1	2
J	1	5	2
K	2	5	4

While determining the cost between two points, say point i and j, firstly we calculate the Chebyshev distance between these points. Then, we compare the distance with free times of k points following point i. Maximum of these k+1 values is assigned as the final cost between two points.

If the problem had been defined as a conventional TSP the tour given in Figure 2.2 (Tour 1) would have been an optimal solution with 24 time units. However, this is a costly tour (34 time units) for SDTSP. On the other hand, although the tour given in Figure 2.3 is more costly than an optimal solution for the conventional TSP (25 time units) it will be an optimal solution for SDTSP. We can conclude that it is not guaranteed that optimal solutions for a conventional TSP are optimal for SDTSP.

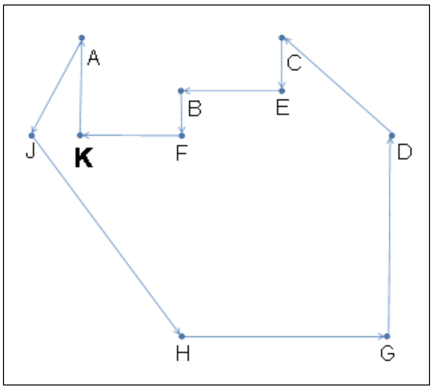


Figure 2.2 – Tour 1

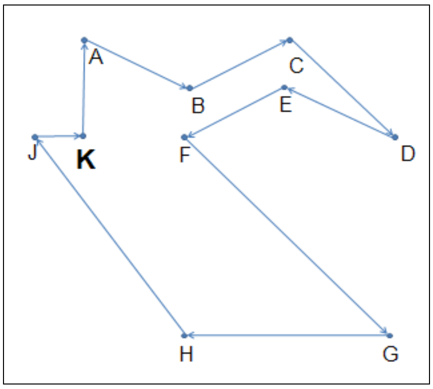


Figure 2.3 – Tour 2

3. BACKGROUND AND LITERATURE SURVEY

As we pointed out above, in this study we will compare the performances of three metaheuristics on solving SDTSP. Since SDTSP is a recently defined problem there are not so many comprehensive studies directly associated with it. In the most relevant study, Alkaya and Duman [3] focused on a particular placement machine type and formulated the feeder configuration, placement sequencing and assembly time minimization problems. They proposed three heuristics and showed that these heuristics improved previously suggested solution methods when compared to SA.

We surveyed the literature in terms of application of metaheuristics to solve generalizations of TSP and other combinatorial optimization problems. The metaheuristics to be exploited are explained and surveyed below.

3.1. Simulated Annealing (SA)

In metallurgy, annealing is a technique involving heating a solid to a high temperature (melting) and then controlled cooling it (freezing) very slowly to provide it to reach thermal equilibrium at each temperature. Thermal equilibrium is reached when large changes in energy do not occur at this temperature anymore [40]. The main goal is to find a configuration that minimizes the energy of the system. In this analogy; system, configuration and energy correspond to a combinatorial optimization problem, solution generated for this problem and cost of this proposed solution, respectively [4]. Metropolis et al. [7] introduced a simple algorithm that simulates this annealing process in metallurgy. This algorithm was latterly named as simulated annealing by Kirkpatrick et al. [4].

The algorithm starts with an initial solution *s* (either randomly or heuristically constructed) and with a high temperature value *T*. There are nested two loops in the remaining of the algorithm. In the inner loop, the temperature is fixed and at each iteration a new solution *s'* in *N(s)* (the set of neighbour solutions of *s*) is generated by using a neighbour function (e.g., 2-opt move, pair-wise exchange etc.). If cost of this new solution is less than that of current solution then it is accepted as current solution (for a minimization problem) for the next iteration. Otherwise, it is accepted in the expectation of escaping from encountered local optima while searching for a global optima [41] with a probability of ***exp(-ΔE/k$_B$T)*** where ***ΔE, k_B*** and ***T*** are the difference between costs of new solution and current solution, Boltzmann constant and current temperature, respectively. When an equilibrium condition is satisfied, the inner loop is terminated and then temperature is reduced by using a cooling schedule (e.g., linear cooling, geometric cooling, etc.). At the very early iterations of the outer loop, the temperature is so high and this causes the system to accept most of new solutions which increase the cost and this helps the system to get rid of a local minima. The algorithm is terminated when the current temperature is either a numerical value which is nearly zero or some other stopping criterion is satisfied. We give a pseudocode for the algorithm below in Table 3.1.

Table 3.1 – Simulated Annealing procedure

```
Procedure Simulated_Annealing
    Generate an initial solution s0 = s = s* (s* is the best solution generated so far,   s is the current solution)
    Initialize temperature T = T0 > 0
    Initialize iteration limit of inner loop, R
    solutionCounter = 0
    While solutionCounter <= solutionLimit do
        Repeat R times
            (i) Select a random neighbour s' of the current solution,  (s' ∈ N(s))
            (ii) Set Δ(C) = C(s') − C(s) // C(s) is cost of solution s
            (iii) If (Δ(C) <= 0)
                - Set s = s'
                - If C(s) < C(s*) then set s* = s
            (iv) If (Δ(C) > 0)
                - Choose a random number r uniformly from [0, 1)
                - If r < e^(-Δ(C)/kT) then set s = s' // k is Boltzmann constant, optional
            solutionCounter++
        R = R * a
        T = T / b
    return s*
```

Skiscim and Golden [11] applied Canonical Correlation Analysis Optimization (CCAO) heuristic, 2-opt algorithm and SA to solve ten randomly generated TSP instances with various sizes. Test results showed that CCAO heuristic and 2-opt algorithm outperformed SA in terms of solution quality and run time. Repeated application of 2-opt algorithm outperformed SA on nine out of ten problems. CCAO heuristic was faster than SA and produced higher quality solutions. In addition, they applied a new modified equilibrium condition and this facilitated to reduce the computation time. Johnson et al. [14, 15] applied SA to two different NP-Hard problems: the graph partitioning problem (GPP) and the graph coloring problem (GCP). They used a geometric cooling schedule, that is, the temperature decreases by a product factor. For the GPP, SA obtained final solutions which are approximately 5 percent better than those obtained by the best of traditional algorithms (Lin-Kernighan Heuristic [23]). For sparse graphs, SA was better whereas for some structured graphs Lin-Kernighan heuristic obtained better solutions. For the GCP, SA performed as good as a heuristic which is considered the best one for this problem (Johri and Matula [24]). However, computation time for SA was longer than that of this heuristic. Alkaya and Duman [3] compared the performance of SA with those of three heuristics proposed (record-to-record travel with local exchange moves, adjust first point procedure and postpone deviant) and showed that these heuristics gave comparable results.

Since a neighbour function has a great importance on the quality of new solutions to be generated for a problem, Nahar et al. [12] carried out experiments with six neighbour functions by considering perturbation and partition problems. Of these schemes; Cycle of Length 3 and Two Bond Exchange were tested with only perturbation problems whereas Pair + Single Move and Pair + Pair were tested with only partition problems. In addition, Single Move and Pairwise Exchange were tested with both problems. As a result of experiments on optimal linear arrangement problem, they concluded that a generalized heuristic cannot substitue a well-designed problem specific heuristic, that for some problems a good starting solution may increase the quality of obtained final solutions, and that selection of generation scheme is effective on whether generated solutions are scattered on search space or on a small part of search space. Ma et al. [20] tested six neighbour functions satisfying asymptotical convergence requirements. They applied SA using these functions separately to solve TSP, flow-shop scheduling problem and quadratic assignment problem. Experiments revealed that the efficiencies of solutions obtained from different functions are different from each other. They concluded that SA adopting a proper function has ability to produce high-quality solutions for different combinatorial optimization problems. Two commonly used neighbour functions by SA are reversing a sub-tour and inserting a sub-tour between two adjacent vertices. These procedures are generally carried out randomly and this may cause “good” edges to be destroyed and “bad” edges to be kept. To overcome this problem, Li et al. [22] defined a probabilistic neighbourhood model and then this model was combined with SA. This new algorithm was applied to solve TSP. Although this is a preliminary study, it showed that SA with probabilistic neighbourhood can outperform conventional SA.

Lam and Delosme [13] proposed a new cooling schedule using statistical quantities. They applied SA with this new schedule to solve TSP and cell placement problem. Results showed that the schedule gave in up to twenty four times speedup over annealing schedules proposed thus far in the literature. Abramson et al. [19] applied simulated annealing to school timetabling problem. They compared performances of six cooling schedules: geometric cooling schedule, a cooling schedule with multiple cooling rates, geometric reheating, enhanced geometric reheating, non-monotonic cooling and reheating dependent on cost function. Basic geometric cooling schedule, which is developed by van Laarhoven and Aarts [25], was used as a benchmark in this study. As a consequence of experiments conducted, they concluded that using multiple cooling rates for a given problem may result in better solutions in less time than those generated by a single cooling rate.

Martin et al. [16, 17] proposed a simulated annealing algorithm for TSP. They embedded 3-opt and Lin-Kernighan local search techniques and utilized Monte Carlo Method's capability of making large changes even at low temperatures. This new algorithm improved 3-opt by 1.6% and Lin-Kernighan by 1.3%. Cardoso et al. [18] proposed a non-equilibrium SA (NESA) algorithm. This algorithm does not wait for achieving near-equilibrium conditions to terminate inner loop. As soon as it obtains an improved solution inner loop is terminated. The new algorithm and the classical one are applied to solve TSP (400-city) and optimization of a pressure relief header network. As a result of experiments, it is revealed that the modified algorithm converged to global optima faster than the classical one. In addition, they also applied two different acceptance mechanisms: Metropolis and Glauber. The computational tests showed that the Metropolis acceptance mechanism, no matter in its original or NESA form, outperformed its Glauber variant. Ohlmann and Thomas [21] developed a new sort of SA algorithm incorporating a variable penalty method and applied this algorithm to solve TSP with Time Windows (TSPTW). This new penalty method corresponds to pressure at the other side of the analogy. Penalty multiplier in compressed annealing makes search procedure dynamic and this makes compressed annealing applicable to combinatorial optimization problems. In TSPTW, while temperature controls the probability of transition to a more costly solution, pressure controls the probability of transition to an infeasible solution. The algorithm was tested with five groups of TSPTW instances. Near-optimal solutions were obtained in a reasonable computation time. This new algorithm with variable-penalty generally outperformed SA with static-penalty.

3.2. Artificial Bee Colony (ABC)

The ABC algorithm is a nature-inspired and population based optimization algorithm simulating intelligent foraging behavior of a honey bee swarm. There are three groups of bees in a honey bee hive: Employed bees, onlookers and scouts. Employed bees exploit food sources surrounding the hive. Each employed bee has her own food source. They collect nectar from the sources and transport it to the hive. When they arrive the hive, they make waggle dance on dance floor in the hive. Onlookers waiting on the dance floor watch each dance performed by each employed bee and make an inference about where the food sources are and the amount of nectar in the food sources. Then, onlookers decide food sources to be exploited. The more nectar a food source has, the more chance the food source has for being chosen by an onlooker bee. After a while, the food sources having been exploited are exhausted. In such a case, associated employed bee abandons her food source and becomes a scout. Then, it starts to explore for a new food source randomly. In the analogy, a food source represents a solution for the problem to be optimized. The amount of nectar is the fitness value of the solution.

The algorithm (Table 3.2) is as follows: Initial solutions are generated randomly where the number of the solutions is half of the bee population. After the initialization, the algorithm steps in a loop which will last as the maximum number of cycles (MNC). The loop consists of three phases: Employed bee, onlooker and scout phases. In employed bee phase, every employed bee generates a neighbour solution of her own solution and evaluates it. Evaluation process is carried out as in (3.1) where c_i corresponds to objective function value of the solution. If the new solution has a higher fitness value than that of current solution then the current solution is assigned as the new solution. Otherwise, the new solution is omitted. Then, employed bees share this information with onlookers. In onlooker phase, they, according to fitness values of the solutions, probabilistically choose the solution to be improved. Probability values are calculated by the expression in (3.2). Roulette wheel selection is generally used during this selection process. That is, the higher fitness value a solution has, the more chance the solution has for being selected by an onlooker. Every onlooker bee also tries to improve the solution she selected and evaluates the new solution. Similar to the employed bee phase if the new solution has a higher fitness value than that of the current solution the new solution is memorized as the current solution. Otherwise, the new solution is omitted. Every solution has a *limit* value. If a solution cannot be improved further through successive number of *limit* times, this solution is abandoned and associated employed bee becomes a scout. In scout phase, if there is a scout, it seeks for a new solution randomly. In the basic ABC algorithm, at most one scout is sent to find a new solution. We can conclude that in the ABC algorithm, employed bees and onlookers execute exploitation process whereas scouts carry out exploration process.

$$fitness_i = \begin{Bmatrix} 1/(1+c_i) & if\ c_i \geq 0 \\ 1+\mathbf{abs}(c_i) & if\ c_i < 0 \end{Bmatrix} \tag{3.1}$$

$$p_i = \frac{fitness_i}{\sum_{i=1}^{NP} fitness_i} \tag{3.2}$$

This new nature-inspired, population based, swarm intelligence optimization algorithm was proposed by Karaboga and Basturk [5]. In this study, they used ABC algorithm for optimizing multimodal (a function which has two or more local optima) and multivariable functions. Then, performance of ABC was compared with those of Genetic Algorithm (GA), Particle Swarm Optimization (PSO) algorithm and Particle Swarm Inspired Evolutionary Algorithm (PS-EA). ABC outperformed the other algorithms. The results showed that this new algorithm is competitive with commonly used algorithms for optimization problems in the literature.

Table 3.2 – Artificial Bee Colony procedure

```
Procedure Artificial_Bee_Colony
    Generate an initial set consisting of CS solutions
    Calculate fitness values f of the solutions
    solutionCounter = 0
    While solutionCounter <= solutionLimit do
        For each solution s
            (i) Generate a new solution s' from s by using a neighbour function
            (ii) Calculate fitness value f(s') of the new solution
            (iii) If (f(s') >= f(s))
                - Set the new solution as current solution, s = s'
            (iv) Else
                - Increase limit counter of s by 1
            solutionCounter++
        Calculate probability values of each solution for roulette wheel selection
        Repeat CS times
            (i) Select a solution s in the set according to its probability value.
            (ii) Generate a new solution s' from s by using a neighbourcfunction
            (iii) Calculate fitness value f(s') of the new solution
            (iv) If (f(s') >= f(s))
                - Set the new solution as current solution, s = s'
                - Update probability values
            (v) Else
                - Increase limit counter of s by 1
            solutionCounter++
        For each solution s
            If(lc(s) > limit)
                - Generate a new solution s' randomly and replace it with solution s
                - solutionCounter++
    Find the solution s* with maximum fitness value in the solution set
    return s*
```

Dongli et al. [38] proposed an improved ABC algorithm based on multi-exchange neighbourhood (MNABC) in which neighbourhood of current solution is exchanged, thus search scope is enlarged and diversification of the ABC algorithm is improved. A series of experiments were performed to compare the performance of MNABC with those of the basic ABC and PSO algorithm. In average, the ABC and MNABC algorithms outperformed PSO algorithm. Compared with the basic ABC, in most of the test instances MNABC algorithm performed better. Experiments showed that the proposed algorithm has better performance, convergence rate and ability to search the solution space in detail in some numerical optimization problems.

Akay and Karaboga [43] presented modified versions of ABC algorithm and applied them to solve real-parameter optimization problems. The basic and modified versions of ABC algorithm were tested on both basic and composite functions presented at the Congress of Evolutionary Computation 2005 (CEC05), and then their performances were compared with not only those of its own versions but also those of state-of-the-art algorithms. Comparison test results showed that the classical ABC algorithm can efficiently solve basic functions whereas the modified versions can produce promising solutions on composite functions.

Singh [26] proposed the first discrete ABC (DABC) algorithm and applied it to solve the leaf-constrained minimum spanning tree (LCMST) problem in which a minimum weighted spanning tree of an undirected, connected and weighted graph is sought among all the spanning trees that have at least l leaves. The performance of the DABC was compared with those of Ant Colony Optimization-LCMST (ACO-LCMST), Tabu Search-LCMST (TS-LCMST) and Subset-Coded GA (SCGA). The proposed DABC algorithm outperformed the other algorithms. However, average solution quality of TS-LCMST was better. With its computational efficiency, ABC-LCMST demonstrated its superiority over ACO-LCMST and SCGA. Singh and Sundar [28] proposed an ABC algorithm to solve quadratic MST problem (Q-MST) in which the aim is to minimize not only the total cost of edges in the tree but also the costs of ordered pairs of distinct edges in this tree. The performance of the proposed algorithm was compared with those of Zhou-Gen GA [29] and the algorithm proposed by Soak et al. [30] which are best heuristics known for the Q-MST problem. The proposed ABC algorithm outperformed the other algorithms in terms of solution quality. Although ZG-GA was faster than the proposed algorithm, it performed worst in terms of solution quality among all other algorithms. Test results showed that the proposed ABC algorithm is an efficient algorithm for the Q-MST problem and competitive to other algorithms proposed for Q-MST problem. Banharnsakun et al. [35] hybridized the ABC algorithm with greedy sub tour crossover method (ABC-GSX) to improve the quality of solutions generated in employed bee phase and applied it to solve TSP. Test results showed that the proposed algorithm performed better than the techniques proposed in the literature thus far in terms of solution quality and the required computational time. Pan et al. [32] proposed a DABC hybridized with a kind of iterated greedy algorithm to obtain the permutation that gives the smallest total flow time. The newly proposed DABC and hybrid discrete differential evolution (hDDE) algorithm from their previous work were tested on the well-known benchmark suite of Taillard. These algorithms were compared with the best performing algorithms in the literature in terms of solution quality and computational cost. Approximately half of best known solutions to the problems chosen from the benchmark were further improved with the proposed algorithms. Karaboga and Gorkemli [33] introduced a new ABC algorithm hybridized with greedy subtour mutation operator to solve TSP. Test results showed that the proposed algorithm gives good solutions for TSP instances considered in the study. Szeto et al. [36] proposed an enhanced version of the ABC algorithm to improve the solution quality of the original one. The enhanced version is applied to solve the capacitated vehicle routing problem (CVRP) in which *m* routes must be determined for different *m* vehicles such that every route will start and end at a depot, every customer will be visited exactly once, the total demand of a route will not exceed capacity of the vehicle on this route and the total cost of all routes will be minimized. The performance of the enhanced ABC algorithm was tested on classical benchmark instances, and then compared with those of the original ABC algorithm, existing metaheuristics and heuristics proposed for CVRP in terms of solution quality and computational cost. It was concluded that the enhanced ABC algorithm is an alternative to solve the CVRP. Nahavandi et al. [42] proposed a binary version of ABC algorithm (DisABC) using a metric of dissimilarity which is different than the one having been used in classical ABC algorithm and applied it to solve the uncapacitated facility location problem (UFLP). The performance of the proposed algorithm was tested with a set of 15 benchmark test problem instances from OR-Library and compared with those of binary DE (binDE) and PSO algorithms. DisABC outperformed these two algorithms and tests showed that it is a promising algorithm to solve a wide range of binary optimization problems. Rodriguez et al. [44] proposed an ABC algorithm to solve the maximally diverse grouping problem (ABC-MDGP) in which a set of *m* elements are divided into *n* disjoint groups so that the diversity among the elements in each group is maximized. The performance of the proposed ABC algorithm was compared with those of the state-of-the-art algorithms. Test results showed that the proposed algorithm is a highly effective and promising algorithm on MDGP.

Pacurib et al. [27] applied an improved version of ABC algorithm to solve Sudoku puzzle. The improved algorithm was tested on three types (easy, medium and hard) of Sudoku puzzles. Solutions obtained from these tests were compared with GA based Sudoku solutions. ABC based solutions were better than all other solutions. Tests and comparisons showed that the proposed ABC algorithm can be used to solve Sudoku puzzles effectively and efficiently. Pan et al. [31] applied a discrete ABC (DABC) algorithm to solve lot-streaming flow shop scheduling problem. In this discrete problem, a candidate solution was represented with a discrete job permutation and discrete operators such as insert, swap were used in order to generate a neighbour solution. A self-adaptive strategy based on insert and swap operators for generating neighbour solutions was developed. In addition, a local search was embedded to use in employed bee phase for further local intensification capability. DABC was compared with GA and hybrid discrete PSO. The test results showed the efficiency of DABC, and that DABC is a promising optimization algorithm. Zhang et al. [34] proposed an ABC algorithm in which path relinking, which is an evolutionary method, was utilized in solution construction mechanism, to solve TSP. Test results for benchmark TSP instances showed that the proposed algorithm is very efficient and competitive to the best existing algorithms in terms of solution quality. Karabulut and Tasgetiren [37] presented a DABC to solve TSP with time windows (TSPTW) in which each customer must be visited and served by a vehicle in a prespecified time window in order for a tour to become feasible. In this study, to obtain neighbour solutions destruction and construction algorithm is used and then 1-opt local search is used to further improve the solution quality. The performance of the proposed algorithm was tested on a benchmark set from the literature. Test results showed that the algorithm is very competitive to and, for some instances, better than the best performing algorithms in the literature. Duan et al. [39] proposed a DABC algorithm and applied it to solve the blocking flowshop scheduling (BFS) problem in which the total flowtime is minimized. To generate an initial population, a heuristic providing a level of both solution quality and diversity is used. Because the problem is discrete, food sources were represented with discrete job permutations and discrete operators were utilized to generate new job permutations (food sources). Test results showed that the proposed DABC algorithm is an efficient algorithm for the BFS scheduling problem.

3.3. Migrating Birds Optimization (MBO)

The MBO algorithm is a newly proposed, population-based neighbourhood search technique inspired from the V formation flight of the migrating birds which is proven to be an effective formation in energy minimization. In the analogy, initial solutions correspond to a flock of birds. Likewise the leader bird in the flock, a leader solution is chosen and the rest of the solutions is divided into two parts. Each solution generates a number of neighbour solutions. This number is a determiner value on exploration and it corresponds to the speed of the flock. The higher this value, the more detailed the flock explores its surroundings.

Another significant factor is the number of wing flaps required before the leading bird gets tired and moves backmost of one of the legs of the flock. After the leading bird gets tired, it moves backmost of either right or left leg of the flock. This corresponds to that neighborhood of the leading solution is searched in detail and it is the time for the leading solution to benefit from the "benefit mechanism". According to the amount of overlapping wing-tip spacing (WTS) air friction force posterior birds are exposed to changes. On the other side of the analogy, this corresponds to the number of best solutions which will be shared by a generator solution with the solution just behind it.

The algorithm (Table 3.3) starts with a number (n) of initial solutions corresponding to birds in a V formation. There are nested two loops in the remaining of the algorithm. The outer loop repeats while current number of solutions generated is less than maximum number of solutions to be generated whereas the inner loop repeats limit number of tours (m) times. In the inner loop, p different neighbour solutions of the leading solution are generated and their objective function values are evaluated. If there is some solution whose objective value is less than that of the leading solution, then the leading solution is replaced with this new solution. Otherwise, the leading solution remains the same. $2x$ out of these newly generated solutions are shared with solutions that follow. For each solution *s* in the flock excluding the leading solution, *p-x* neighbour solutions of *s* are generated and their objective function values are evaluated. If there is some solution *s'* whose objective value is less than that of *s*, then *s* is replaced with *s'*. Otherwise, the solutions coming from the solution in the front of *s* are checked. If there is some solution whose objective value is less than that of *s*, then *s* is replaced with this solution. Otherwise, *s* remains the same. Then, best *x* unused solutions are shared with the solution that follows. This new mechanism is called "benefit mechanism".

After the inner loop ends, the leading solution is moved to backmost of either left or right legs of the flock and one of the following solutions is assigned as new leading solution. At the end of the outer loop, we expect the best solution obtained so far is the optimal/near-optimal solution of the associated problem.

This new metaheuristic was proposed by Duman et al. [6]. They applied it to solve quadratic assignment problem instances arising from printed circuit board assembly workshops. Its performance was compared with those of metaheuristics implemented and compared in two previous studies. These are simulated annealing, tabu search, genetic algorithm, scatter search, particle swarm optimization, differential evolution and guided evolutionary simulated annealing. In this comparison, the MBO outperformed the best performed metaheuristic (simulated annealing) in the previous studies by approximately three percent on the average. In addition, MBO was tested with some benchmark problem instances obtained from QAPLIB and in most of the instances it obtained the best known solutions. As a result of these tests, it is concluded that the MBO is promising and it is a candidate to become one of the highly competitive metaheuristics. Duman and Elikucuk [45] applied MBO to solve fraud detection problem. They also proposed a new version of MBO where a different benefit mechanism is used. They tested the original MBO algorithm and its new version on real data and compared their performance with that of genetic algorithm hybridized with scatter search (GASS). Test results showed that the MBO algorithm and its new version performed significantly better than the GASS algorithm.

***Table 3.3 – Migrating Birds Optimization procedure* [6]**

```
1. Procedure Migrating_Birds_Optimization
2.      Generate n initial solutions randomly and place them in an imaginary V form
3.      solutionCounter = 0
4.      While solutionCounter <= solutionLimit do
5.          Repeat m times
6.              - Generate p neighbours of the leading solution and evaluate them
7.              - i = i + p
8.              If objective function value of any neighbour is less than that of the leading
                solution
9.                  - Replace the leading solution with it
10.             Otherwise
11.                 - The leading solution remains the same
12.             - Share best 2x neighbour solutions with the solutions that follow
13.             For each solution s in the flock excluding the leading solution
14.                 - Generate p - x neighbours of s and evaluate them.
15.                 If objective function value of any of these neighbours is less than
                    that of s
16.                     - Replace s with this neighbour
17.                 Otherwise
18.                     If objective function value of any of x shared neighbours is
                        less than that of s
19.                         - Replace s with this shared neighbour
20.                     Otherwise
21.                         - s remains the same
22.                 i = i + (p – x)
23.         Move the leading solution to backmost of one of the legs of the flock and move one of
            the solutions that follow it to the leadership
24.     Return the best solution in the flock
```

4. NEIGHBOUR FUNCTIONS

Since a solution for an SDTSP instance answers the question of in which *order* vertices should be visited to minimize the total assembly time, it can be represented as a permutation of vertex indices (See Figure 4.1).

Figure 4.1 – Solution representation (Permutation)

In this study, we tested and used ten neighbour functions.

4.1. Adjacent Interchange

One of the indices in the permutation is chosen randomly and this index is interchanged with its adjacent on the right. If the chosen index is in the last position, then it is interchanged with the index that is in the first position (Figure 4.2).

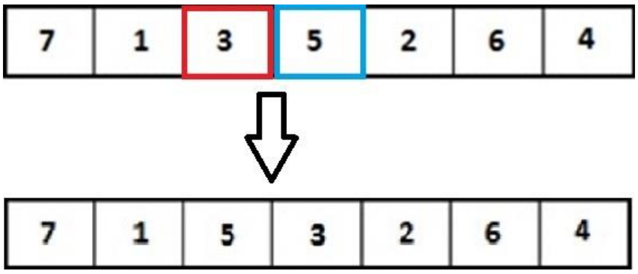


Figure 4.2 – Adjacent interchange

4.2. Random Interchange

Two distinct indices in the permutation are chosen randomly and interchanged (Figure 4.3).

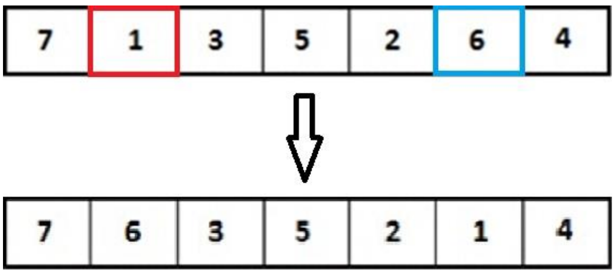


Figure 4.3 – Random interchange

4.3. Move a Single Term

One of the indices in the permutation is chosen randomly. Number of steps to be moved is chosen randomly from a predetermined set of candidate numbers, and then the index is moved rightwards by the number of steps further from the current position (Figure 4.4).

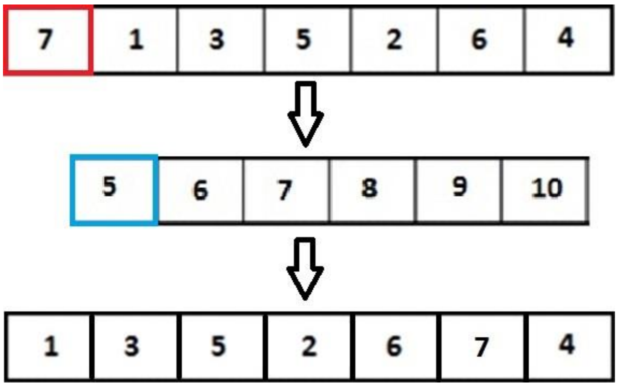


Figure 4.4 – Move a single term

4.4. Move a Subsequence

A subsequence of the permutation, whose length is determined by randomly choosing from a predetermined set of candidate numbers, is designated, and then the subsequence is moved rightwards by the number, which is generated randomly in the range of 1 and the length of the permutation, further from the current position of the beginning index of the subsequence (Figure 4.6).

4.5. Reverse a Subsequence

A subsequence of the permutation is determined as in the “Move a Subsequence” function (See Figure 4.6), and then the subsequence is reversed (Figure 4.5).

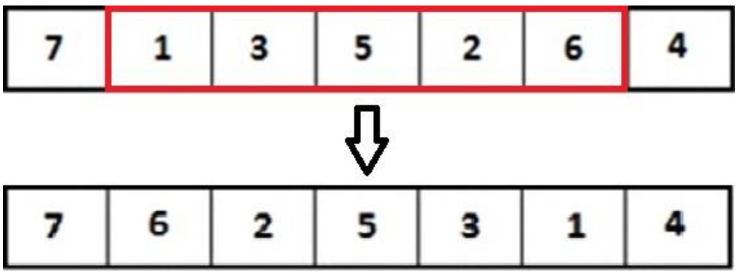


Figure 4.5 – Reverse a subsequence

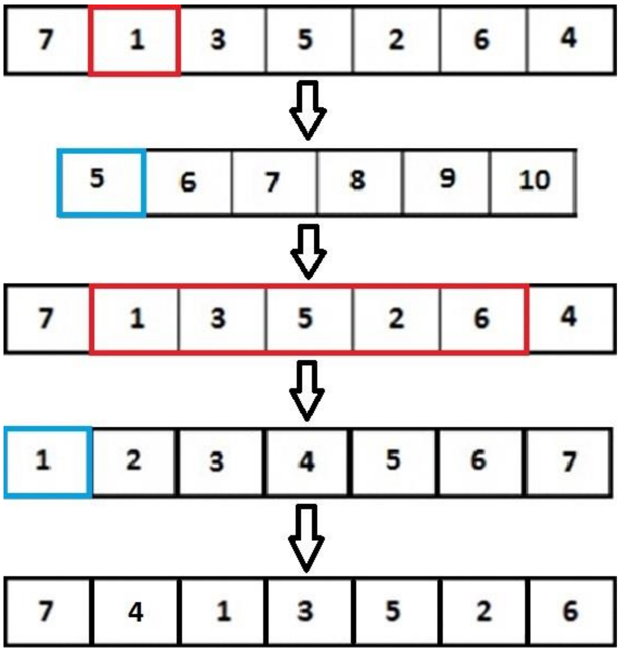


Figure 4.6 – Move a subsequence

4.6. Reverse and/or Move a Subsequence

This function consists of two different functions: reverse and move a subsequence, and reverse or move a subsequence.

A subsequence of the permutation is determined as in the “Move a Subsequence” function (See Figure 4.6). Then, a random number which is in the range [0, 1) is generated. If the generated number is less than 0.5, then reversing and move a subsequence function is carried out. This function, firstly, reverses the subsequence and then moves the reversed subsequence rightwards by a random number of steps in the range of 1 and the length of the permutation further from the current position of the beginning index of the subsequence. If the generated number is equal or greater than 0.5, then the function reverses or moves the subsequence. A new random number in the range of [0, 1) is generated. If the generated number is less than 0.5, then the subsequence is reversed. Otherwise, it is moved rightwards as explained above.

4.7. Swap Subsequences

The permutation is divided into two. One subsequence for each division is determined as explained in the “Move a Subsequence” function (See Figure 4.6), and then these subsequences are swapped with each other (Figure 4.7).

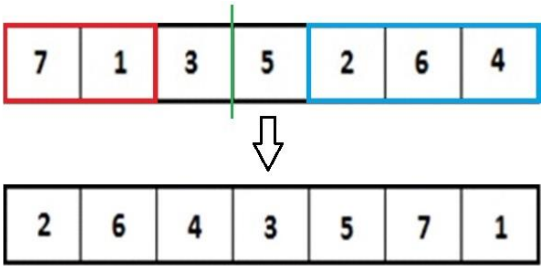


Figure 4.7 – Swap subsequences

4.8. Reverse and Swap Subsequences

The subsequences are determined as in the "Swap Subsequences" function (See Figure 4.7) and they are reversed. Then, they are swapped with each other (Figure 4.8).

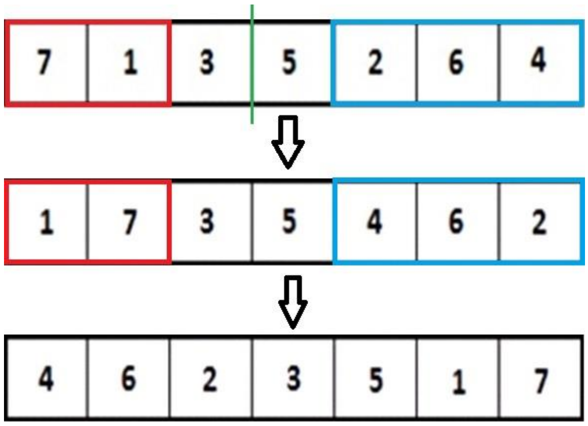


Figure 4.8 – Reverse and swap subsequences

4.9. 2-OPT

This function is the same as "Reverse a Subsequence" function (See Figure 4.5) except that 2-OPT has no subsequence length limit, that is, the subsequence may be the permutation itself (Figure 4.9).

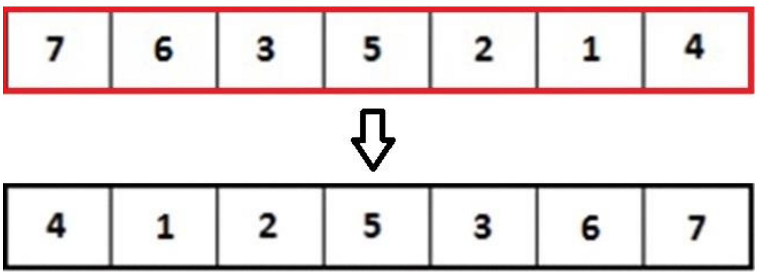


Figure 4.9 – 2-OPT

4.10.Random Remove and Reinsert

One of the indices is chosen randomly and then it is inserted into a randomly chosen position (Figure 4.10).

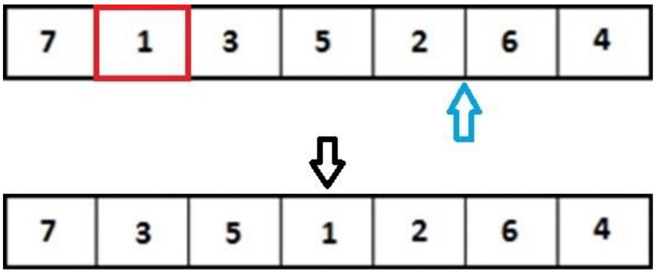


Figure 4.10 – Random remove and reinsert

5. EXPERIMENTAL SETUP

To reveal the performance of the metaheuristics we conducted five groups of tests:

- tests conducted to fine tune the parameters of the algorithms,
- tests conducted to determine the neighbour functions with which the algorithms can show their greatest performance,
- tests conducted to compare the performances of the algorithms,
- tests conducted to observe the effect of *k* value and
- tests conducted to compare the performances of the algorithms with those of GAMS.

For all tests, we used the specifications of the placement machine described in Section 2. That is, the board carrier speed is 120 mm/s in both x and y directions and the placement machine transports the next head onto mount position in 0.20 sec., 0.23 sec., 0.33 sec. and 0.40 sec. for weight groups 1, 2, 3 and 4, respectively. *k* is 20 for all tests except test four.

There are two kinds of data used for the tests: real and randomly generated data. Real data were obtained from 8 different-size real PCBs. These PCBs consist of 146, 152, 170, 261, 262, 280, 185 and 159 components, respectively. Table 5.1 gives the details regarding real PCBs.

The random data were generated by a random data generator developed in [2]. There are two kinds of model for generating the random data: structured and homogeneous model. In structured model, components in the same weight group were juxtaposed and this model resembles the real data. On the other hand, in homogeneous model, components are placed homogeneously on the board regardless of their weight category. We generated four types of PCB data each having 100, 200, 300 or 400 components. All random data include components from all weight groups, but mostly from group 1.

In our computational environment, as a result of preliminary tests we observed that the maximum problem size that GAMS can solve is 57. However, problem instances including 25 components or less can be solved in a reasonable time. Hence, in order to conduct the fifth group of tests we generated three types of PCB data each having 15, 20 or 25 components. All instances include components from all weight groups.

We compared the performances using two criteria:

1 – Total Assembly Time (TAT): The time required for a chip mounter to complete assembly of a PCB. This gives us the objective function value for a problem instance.

2 – Run Time (RT): The time required for a test case to be completed.

The tests are performed on a machine with Intel Xeon CPU E5 at 3.10 GHz. with 128 GB RAM using Windows 7 Professional OS.

In the following, we present the results of the first test conducted to fine tune the parameters of the algorithms.

Table 5.1 - Distribution of weight groups of the real data

Group	rPS11AK08-9	rPS11AK12-7	rPS11AK15-4	rPS11AK16-3	rPS11AK16-4	rPS11AK16-5	rPS11AK17N3	rPS11AK1011
Group 1	63	65	86	110	106	101	88	72
Group 2	69	78	77	127	138	155	84	75
Group 3	14	9	7	24	18	21	13	12
Group 4	0	0	0	0	0	3	0	0
Total	**146**	**152**	**170**	**261**	**262**	**280**	**185**	**159**

All these algorithms generate their initial solution(s) randomly and to make a fair comparison we impose a restriction on the number of solutions generated on the same problem instance. This number depends on the number of components a problem instance (or a PCB) consists of and is calculated by the formula given below.

$$neighbour\ solution\ limit = \frac{(number\ of\ points\ in\ the\ problem)^3}{3}$$

The metaheuristics we exploit in this study have various parameters which have to be fine-tuned. For SA, we need to fine-tune initial temperature (*T*), temperature decrease ratio (*a*), initial number of iterations for inner loop (*R*) and its expansion coefficient (b). We took these parameters and their potential values from [43]. That is, T is set to either 100 or 1000. R is set to either 5 or 20. a and b are set to either 1.1 or 1.5. Results are given in Table 5.2 where we observe that T=100, R=20, a=1.1 and b=1.1 show the best performance.

Table 5.2 – SA parameter test results

T	R	a	b	RT	TAT
100	**20**	**1.1**	**1.1**	**2190.32**	**52.98**
1000	5	1.5	1.5	2421.36	53.1
100	20	1.5	1.5	2430.85	53.36
1000	20	1.5	1.5	2335.05	53.38
1000	5	1.1	1.1	2151.01	53.4
100	5	1.1	1.1	2015.02	54.07
100	5	1.5	1.5	2448.02	54.82
1000	20	1.1	1.1	2267.09	61.35
1000	20	1.5	1.1	4.23	79.82
100	20	1.5	1.1	2.57	89.27
1000	5	1.5	1.1	1.08	107.01
100	5	1.5	1.1	0.79	117.19
100	5	1.1	1.5	2820.35	175.17
100	20	1.1	1.5	2824.8	177.95
1000	5	1.1	1.5	2810.96	180.21
1000	20	1.1	1.5	2778.07	181.39

ABC algorithm inherently needs a colony size (CS) value. This value is the number of initial solutions to be generated. It is used to calculate an important value for ABC algorithm, *limit*. *limit* is maximum number of consecutive iterations during which an ineffectual solution is used to generate new solutions. However, we need a last parameter to calculate *limit* value, d. d is used as a denominator in *limit* calculation formula as follows: $limit = \frac{(2*CS*D)}{d}$ [28] where D is the number of dimensions of a vector that represents a point in the problem. We determined potential values for CS arbitrarily but by inspiring the values used in related studies. The values we used for CS are 10, 20 and 30. The referenced study uses 3 for d. In addition to this, we wanted to investigate how 2 will affect the performance of the algorithm, so we incorporated it into the potential values for d. Results are given in Table 5.3 where we observe that CS=30 and d=2 show the best performance.

Table 5.3 – ABC parameter test results

CS	d	RT	TAT
30	**2**	**3862.41**	**113.28**
30	3	3878.92	130.08
20	2	3567.18	131.33
20	3	3580.86	145
10	2	3121.34	160.62
10	3	3016.19	177.42

MBO algorithm has many more parameters than the others: number of initial solutions (*n*), number of tours (*m*), number of neighbour solutions to be generated from a solution (*p*) and number of solutions to be shared with the following solution (*x*). To determine potential values for these parameters we utilized [38]: *n* is set to 13, 25 or 51. *m* is set to 5, 10 and 20. *p* is set to 3, 5 or 7. *x* is set to 1, 2 or 3. Moreover, *p* value has to be equal to or greater than $2*x+1$ due to the inherent design of the algorithm. Results are given in Table 5.4 where we observe that *n*=51, *m*=10, *p*=3 and *x*=1 show the best performance.

Although the best parameter value combination for MBO is (*n*=51, *m*=10, p=5, x=1) we used (51, 10, 3, 1) combination because (51, 10, 5, 1) requires 40% more time than (51, 10, 3, 1) and the solution (51, 10, 5, 1) obtained is just 0.52% better than (51, 10, 3, 1) obtained.

The first group of tests was conducted with the real data. Neighbour solutions were generated by using the random remove and reinsert function. As a result of these tests, we obtained the best parameter value combination for each algorithm to be used in the following section (Please see Table 5.5).

Table 5.4 – MBO parameter test results

n	m	p	x	RT	TAT		n	m	p	x	RT	TAT
51	10	5	1	7152.12	55.34		13	20	5	1	7226.16	56.66
51	20	7	1	8126.42	55.48		13	20	7	2	9256.48	56.73
51	**10**	**3**	**1**	**5102.71**	**55.63**		13	20	3	1	5405.93	56.75
51	5	3	1	5042.54	55.77		25	20	7	1	8570.72	56.8
51	20	7	2	8872.43	55.81		51	5	5	2	8316.16	56.82
51	10	7	2	9220.84	55.81		13	10	7	1	8319.49	56.86
51	5	7	2	9379.84	55.83		25	20	5	2	9344.82	56.89
25	5	7	1	8982.69	55.91		13	10	3	1	5384.89	56.93
51	20	5	2	8661.68	55.94		25	5	5	1	7384.95	56.97
51	5	5	1	7219.87	56		25	5	5	2	9155.53	56.97
51	10	7	1	7512.29	56.02		13	5	7	2	9116.19	57.22
51	5	7	1	8139.41	56.03		13	5	7	3	11043.92	57.24
25	10	5	2	9528.4	56.08		13	5	5	1	7158.74	57.47
51	20	7	3	8506.42	56.1		13	20	7	1	8287.3	57.69
25	5	3	1	5556.8	56.13		13	20	5	2	8946.84	57.79
25	10	7	1	7784.21	56.14		51	20	3	1	5201.95	58.02
51	5	7	3	9775.46	56.17		13	20	7	3	11087.51	58.14
51	10	5	2	8856.91	56.18		25	10	7	3	11127.52	58.19
25	20	7	2	9013.83	56.23		25	10	3	1	5432.83	58.56
25	20	3	1	5254.06	56.29		25	10	5	1	6736.94	58.68
13	10	7	3	10667.29	56.32		25	20	7	3	11635.21	58.98
51	20	5	1	7132.96	56.33		13	5	5	2	8882.38	59.28
51	10	7	3	10363.81	56.39		13	10	5	1	7291.49	59.29
25	10	7	2	9071.49	56.42		25	5	7	2	8986.68	59.34
13	10	5	2	8848.16	56.48		13	5	7	1	8282.12	59.48
13	5	3	1	5380.44	56.58		25	5	7	3	11126.31	59.79
13	10	7	2	9405.13	56.65		25	20	5	1	6889.9	59.92

Table 5.5 – The best parameter value combination for each metaheuristic

Simulated Annealing	Artificial Bee Colony	Migrating Birds Optimization
T=100, R=20, a=1.1, b=1.1	CS=30, d=2	n=51, m=10, p=3, x=1

6. RESULTS AND DISCUSSION

In this section, we present and discuss results of the tests that we described in the previous section. We carried out five groups of tests. The first group was carried out to find the best parameter value combinations for the algorithms and we gave its results in the previous section. In the next subsection, we give the results of the tests carried out to determine the best performing neighbour functions for each algorithm. Then, results of performance comparison tests are given and analyzed. It is followed by various k value tests. Finally, we obtain the best upper-bound solutions for small-scale problem instances by using GAMS and compare its performance with those of the algorithms. In the following, each figure is an average of 10 runs.

6.1. Determining Best Performing Neighbour Functions

After selecting the best performing parameter values of the algorithms we need to determine which neighbour function should be used for each algorithm. Hence, we carried out some tests on real-world data in order to determine best-fit functions.

In Table 6.1, we present average total assembly times of the algorithms for each neighbour function.

Table 6.1 - Average of total assembly times (in seconds)

Neighbour Functions	SA	ABC	MBO
adjacent interchange	106.66	153.9	142.9
move single term	81.06	124.1	87.9
move subsequence	57.02	112.1	59.1
random interchange	54.53	115.7	60.4
random remove and reinsert	**52.09**	110.3	54.4
reverse and/or move subsequence	56.24	110.2	57.4
reverse and swap subsequences	62.49	118.7	62.8
reverse subsequence	68.11	112.8	81.9
swap subsequences	63.26	120.0	62.9
2-opt	52.77	**102.2**	**53.8**

To determine most eligible neighbour function for each algorithm, we compare the performances of the functions by using total assembly times required for all real PCB data. For all algorithms, *adjacent interchange* function shows the worst performance.

For SA, *random remove and reinsert* function shows the best performance. *2-opt* and *random interchange* functions show the second- and the third-best performance, respectively. Total assembly times obtained from tests run with *2-opt* and *random interchange* functions are 1.31% and 4.68% far away from *random remove and reinsert* function, respectively.

For ABC and MBO, *2-opt* function shows the best performance. The second- and the third-best functions are *random remove and reinsert* and *reverse and/or move subsequence* for both, but the order is different. The second-best function for ABC is *reverse and/or move subsequence* whereas it shows the third-best performance for MBO.

When compared to *2-opt* function, *reverse and/or move subsequence* function shows 7.87% worser performance while *random remove and reinsert* function shows 7.98% worser performance for ABC. For MBO, the order changes and performance of *random remove and reinsert* function gets closer to that of *2-opt* function. It is just 1.21% far away from *2-opt* function whereas *reverse and/or move subsequence* function is 6.69%.

As a consequence of these results, we prefer to use *2-opt* function for ABC and MBO, and to use *random remove and reinsert* function for SA.

When we look at Table 6.2, we see a peak on *reverse and/or move subsequence* function which is a result of that ‘reverse’ and ‘move’ operations require much more time than the others require. Because we generate uniformly distributed random numbers we may assume that half of the cases associated with the function makes “reverse” and “move” operations together. Consequently, *reverse and/or move subsequence* function requires much more time than other functions require.

When we compare the functions in terms of run times, we observe the functions that give the best three results in total assembly times comparison requires either the least run time or run time that is very close to the best run time.

Random remove and reinsert function shows the best performance for SA and ABC. MBO shows the best performance with *2-opt* function, again. In SA, the best is followed by *random interchange* and *move single term* functions, respectively. In ABC, the second- and the third-best functions are very close to each other. The second-best function is *random interchange* and the third-best function is *2-opt* function. Finally, in MBO, *random remove and reinsert* and *random interchange* functions show the second- and the third-best performances, respectively.

Table 6.2 – Average run time of neighbour functions for each metaheuristic (in seconds)

Neighbour Functions	**SA**	**ABC**	**MBO**
adjacent interchange	6707.69	8182.69	8798.86
move single term	6509.18	8176.80	8841.94
move subsequence	6751.27	8659.98	9102.35
random interchange	6463.34	8123.06	8789.51
random remove and reinsert	**6419.62**	**8074.59**	8583.80
reverse and/or move subsequence	10469.96	13639.26	13298.34
reverse and swap subsequences	6646.36	8537.49	9130.29
reverse subsequence	6590.84	8480.08	9195.00
swap subsequences	6685.66	8633.80	9301.55
2-opt	6571.07	8124.54	**8556.26**

6.2. Performance Comparison of the Metaheuristics

After determining the parameter values and the neighbour functions for each algorithm, we compare the performances of the algorithms on random data.

In Figure 6.1, we see that total assembly times increase linearly as problem size increases. For all of the problem sizes, the ABC shows the worst performance. SA and MBO show almost the same trend. SA gives slightly better results than those MBO gives.

SA has a mechanism that accepts bad solution as the algorithm progresses because a bad solution may be very close to the optimum solution in the solution space of the problem. MBO has also a similar mechanism. Some of solutions generated and not used by a solution, which are worse compared to this solution, are shared with the following solution. It is anticipated that the following solution uses these solutions to obtain a better solution. Thus, bad solutions compared to a solution are not lost and used to obtain the optimum one. However, ABC suffers lack of such a mechanism. It usually uses good solutions to generate new solutions. Therefore, it cannot utilize solutions which may convey it to optimal/near-optimal solution(s).

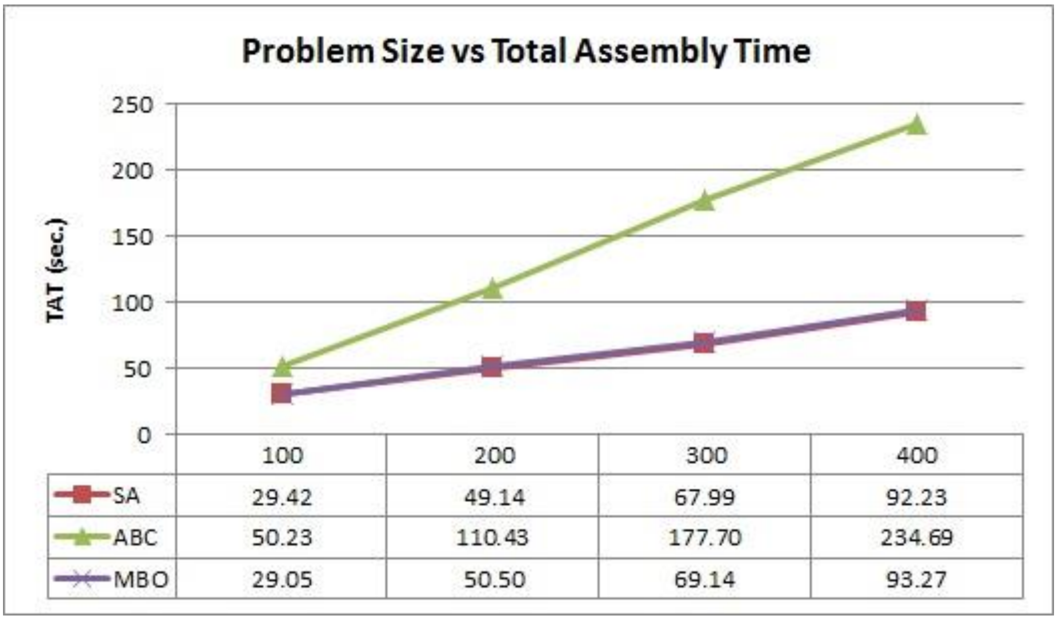


Figure 6.1 – Total assembly times of the metaheuristics

In Figure 6.2, we can see that the run times for all algorithms increase exponentially as problem size increases. All algorithms require approximately the same amount of time at each problem size.

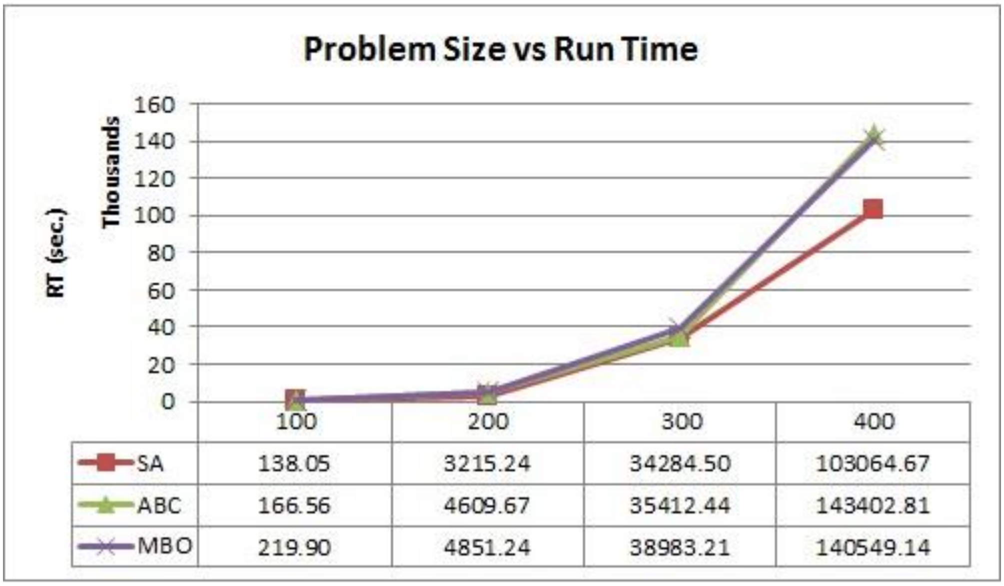
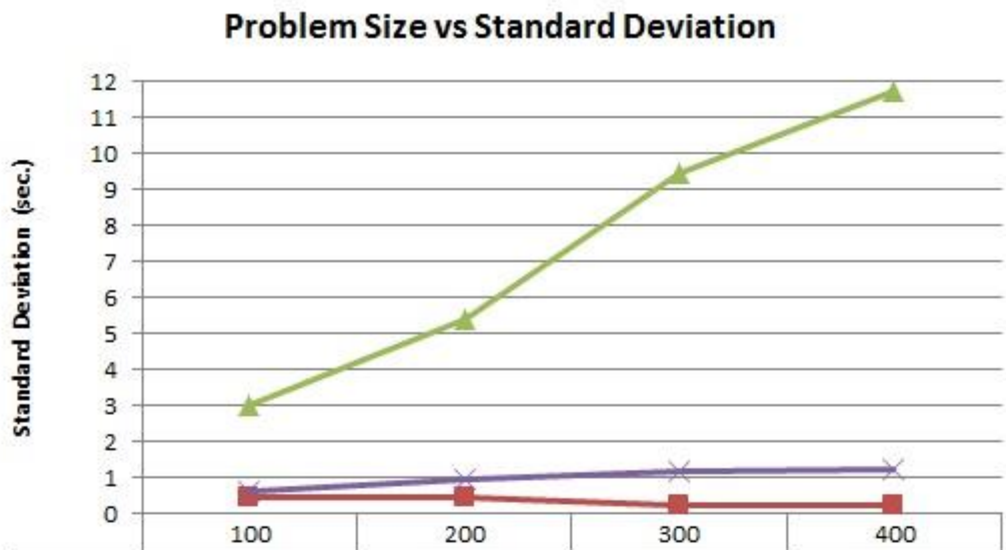


Figure 6.2 – Run times of the metaheuristics

We calculated the standard deviation between total assembly times for each algorithm and each problem size. When we look at Figure 6.3, we can say that the deviations of solutions obtained by ABC is in direct proportion to problem size. Deviations of SA and MBO are many more lower than those of ABC for each problem size. Moreover, they are almost stationary. Deviation of MBO shows a slight logarithmic increase whereas that of SA decreases as problem size increases.



	100	200	300	400
SA	0.45	0.46	0.20	0.23
ABC	3.01	5.39	9.43	11.72
MBO	0.59	0.97	1.15	1.20

Figure 6.3 – Standard deviations of total assembly times for each problem size

6.3. Performance with respect to Various *k* Values

The fourth group of tests is about the effect of various *k* values to the structure of the problem. For that purpose, we run all algorithms with *k* values in the range of [0, 20] on random data. The values in the tables below are average of the values obtained from all algorithms. Figures and tables below depict the results of the tests.

When we look at Figure 6.4 we see a steep increase during the transition from $k = 0$ to $k = 1$. The reason of this steep increase is that we take into account turret times from now on and most of the time, these values are greater than the time required for the carrier to move the board to the next position.

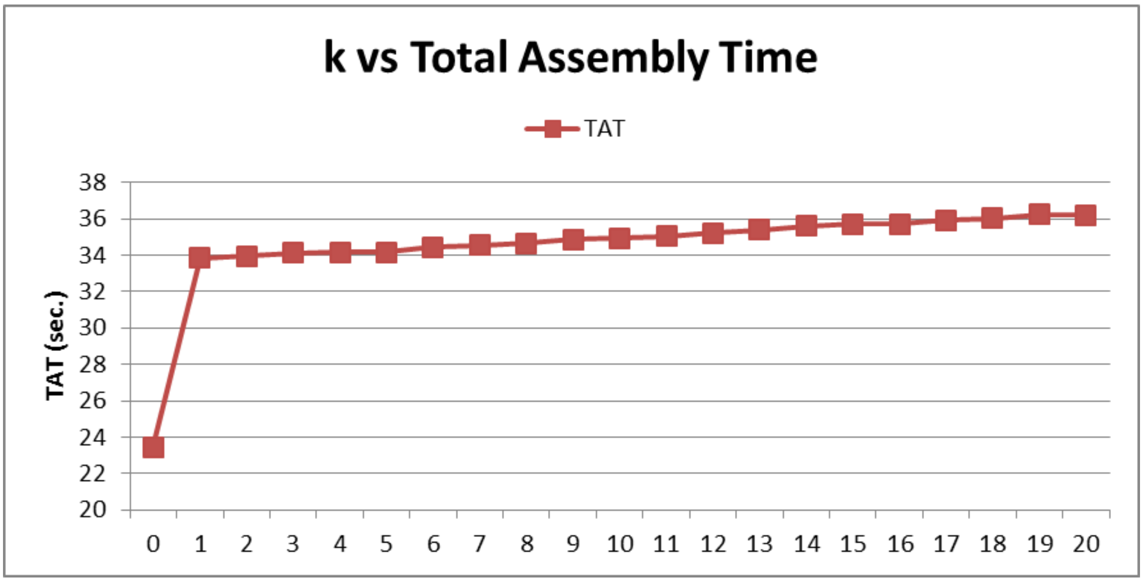


Figure 6.4 – 'k' values versus total assembly times

Figure 6.5 mimics the change in run time as *k* value increases.

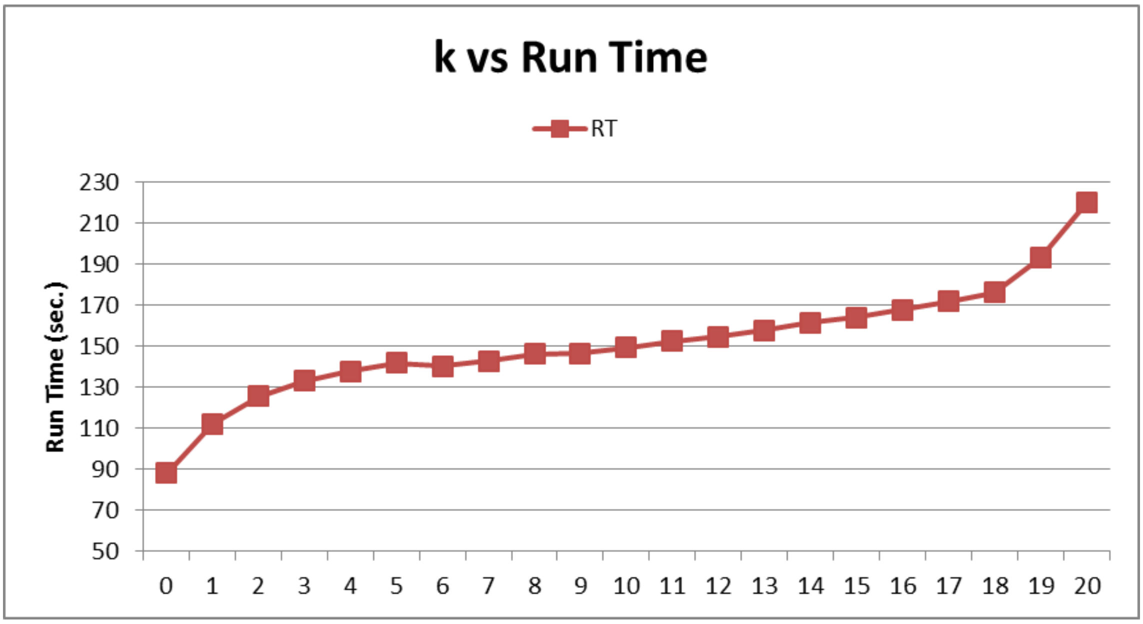


Figure 6.5 – Run times for various 'k' values

6.4. Comparison with Best Upper-Bound Solutions

In this subsection, we compare the results of the metaheuristics with the best upper-bound solutions obtained from GAMS. As we have mentioned in previous section, as a result of preliminary tests we observed that the maximum problem size that GAMS can solve is 57 where when the problem size exceeds 57 GAMS cannot even start solving the problem due to huge number of nonlinear equations. On the other hand, when the problem size is 57 we could not obtain any results after five days of computations. However, problem instances including 25 components or less can be solved in a reasonable time. Hence, in order to conduct the fifth group of tests we generated three types of PCB data each having 15, 20 or 25 components.

In Figure 6.6 we can see that the solutions obtained by the metaheuristics do not diverge as the problem size increases. Interestingly, ABC outperformed the others. In this group of tests, the mechanism which makes SA and MBO more advantageous in the previous tests became a disadvantage for them. SA and MBO, most of the time, take advantage of low quality solutions so as to obtain optimum/near-optimum solution of a problem whereas ABC give less chance to low quality solutions compared to SA and MBO. The sizes of the problem instances used for these tests are smaller compared to the previous tests. Hence, maximum number of solutions to be generated by the metaheuristics is fewer than the previous tests. As a result, SA and MBO are stuck in local minima whereas ABC benefit from high quality solutions to obtain optimal/near-optimal solution(s).

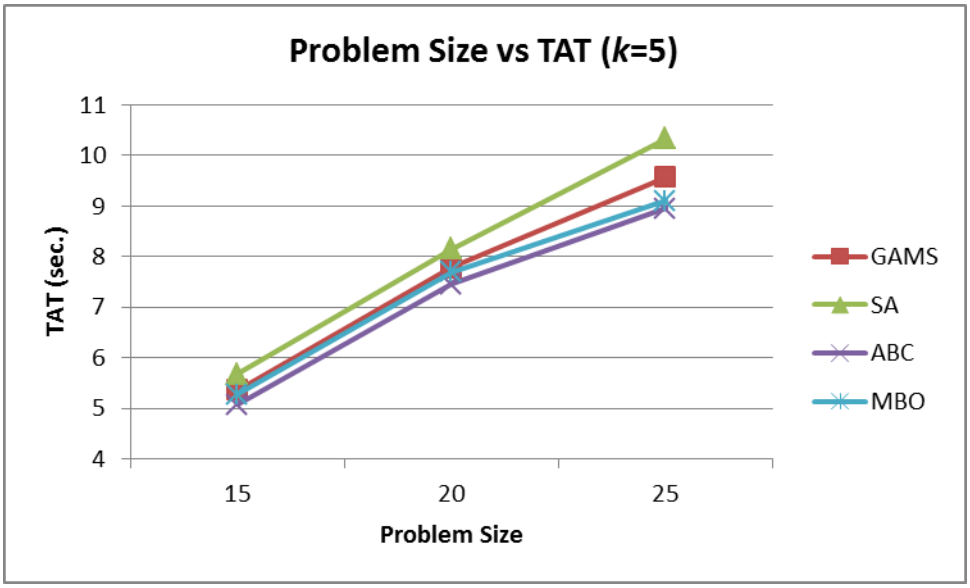


Figure 6.6 – Total assembly time comparison of metaheuristics with GAMS

When we look at Figure 6.7 we can see that the time required to obtain the best upper-bound solution of an SDTSP instance increases exponentially with the problem size because the number of variables increases exponentially and the number of constraints increases linearly as the problem size increases. On the other hand, SA requires the least time among the metaheuristics due to its primitive structure.

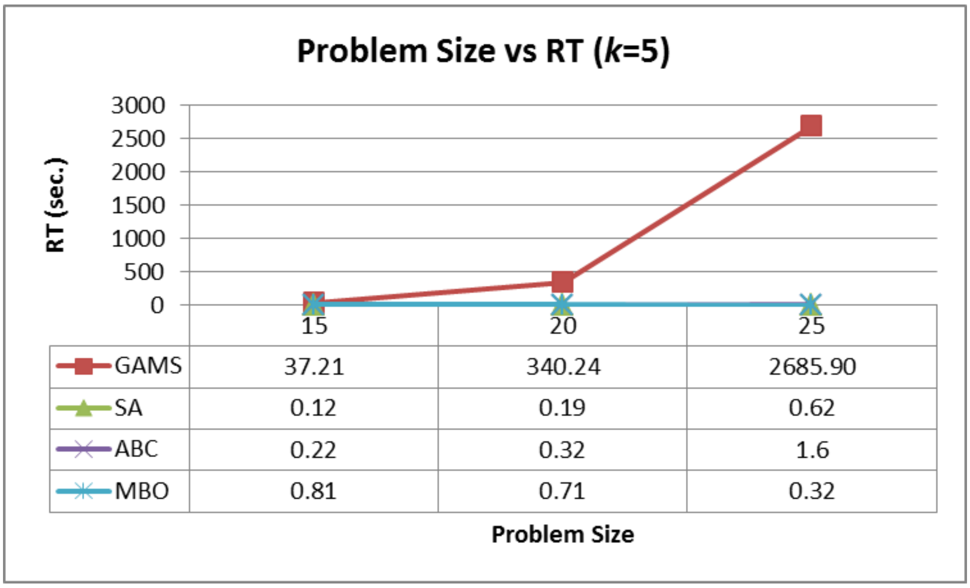


Figure 6.7 – Run time comparison of metaheuristics with GAMS

In addition to the above comparison, we investigated the effect of the value of *k* parameter. In Figure 6.8 we present the total assembly time comparison of metaheuristics and GAMS with respect to various *k* values. In this figure, we observe that there is a steep increase from *k*=0 to *k*=1. The range from *k*=1 to *k*=5 shows almost a steady trend. The ranking order for the performance of the metaheuristics is the same as we observe in Figure 6.6. When we analyze the run time figure (Figure 6.9) of those tests we see an exponential increase for GAMS whereas the metaheuristics show a steady trend. The increase in the run time is exponential as in Figure 6.7, but the reason is different. The reason of this increase is that the number of nonlinear equations increases with the value of *k* parameter.

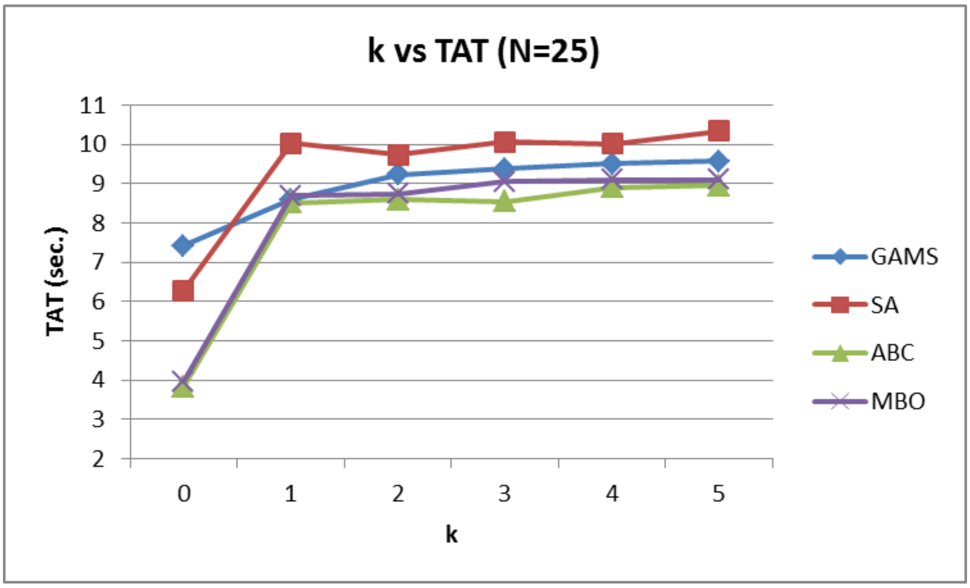
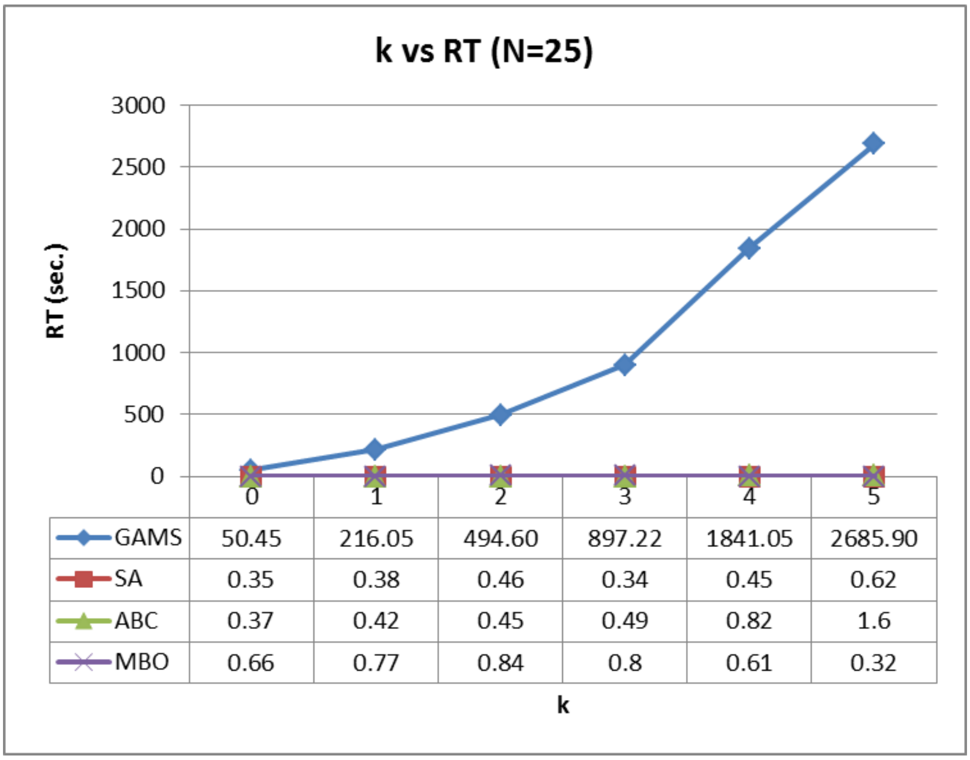


Figure 6.8 – Total assembly time comparison with respect to 'k' parameter



	0	1	2	3	4	5
GAMS	50.45	216.05	494.60	897.22	1841.05	2685.90
SA	0.35	0.38	0.46	0.34	0.45	0.62
ABC	0.37	0.42	0.45	0.49	0.82	1.6
MBO	0.66	0.77	0.84	0.8	0.61	0.32

Figure 6.9 – Run time comparison with respect to 'k' parameter

7. CONCLUSION

SDTSP is a newly introduced combinatorial optimization problem which emerged during optimization of placement machines having been used in PCB assembly. In this study, we compared performances of three metaheuristics on SDTSP instances in terms of solution quality and run time and observed the effect of various k values to the complexity of the problem. All algorithms utilized their best parameter value combination. ABC and MBO used *2-opt* neighbour function whereas SA used *random remove and reinsert* function. SA outperformed MBO and ABC by 1.33% and 140%, respectively. As you can infer from this result, the quality of solutions obtained by MBO is comparable to that obtained by SA. Moreover, small deviations of total assembly times of solutions obtained by MBO demonstrate how robust it is. Considering its run time performance also, we can say that MBO is a promising metaheuristic for combinatorial optimization problems. In addition, from the analysis of k parameter tests we can deduce that solving SDTSP even for k = 1 is much more challenging than solving classical TSP. Lastly, we obtained best upper-bound solutions for small-scale problem instances by means of GAMS. The behaviour of the metaheuristics for both various k values and increasing problem sizes were investigated by considering run time and total assembly time. We observed that the time required for obtaining the best upper-bound solution of an SDTSP instance increases exponentially with the problem size whereas the metaheuristics can find comparable solutions in a limited time. Consequently, we deduced that using these metaheuristics to obtain solutions for medium- and large-scale instances of SDTSP and similar problems is a good choice.

A small improvement in operation time of placement machines makes the time required for return of investment shorter. Therefore, we anticipate this study will contribute to production sector directly or indirectly. In addition to industrial contributions, the undertaken study contributed to the related literature by being presented in *The 9th International Conference on Optimization: Techniques and Applications, Taiwan, Dec. 12-16, 2013.*

8. REFERENCES

[1] Rutenbar, R.A. (1989) Simulated Annealing Algorithms: An Overview. IEEE Circuits and Devices Magazine, 8755-3996 89 0100-001981 00.

[2] Duman, E. (2007a) Modeling the Operations of a Component Placement Machine with Rotational Turret and Stationary Component Magazine. Journal of the Operational Research Society, 58, 317-325.

[3] Alkaya, A.F., Duman, E. (2013) An Application of the Sequence Dependent Traveling Salesman Problem in Printed Circuit Board Assembly. IEEE Transactions on Components, Packaging and Manufacturing Technology, Vol. 3, No. 6.

[4] Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P. (1983) Optimization by Simulated Annealing. Science, (13), 220 (4598), doi:10.1126/science.220.4598.671, 671-680.

[5] Karaboga, D., Basturk, B. (2007) A Powerful and Efficient Algorithm for Numerical Function Optimization: Artificial Bee Colony (ABC) Algorithm. Journal of Global Optimization, 39, (3), 459-171.

[6] Duman, E., Uysal, M., Alkaya, A.F. (2012) Migrating Birds Optimization: A New Metaheuristic Approach and its Performance on Quadratic Assignment Problem. Information Sciences, 217, 65-77.

[7] Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A., Teller, E. (1953) Equation of State Calculations by Fast Computing Machines. Journal of Chemical Physics, 21, (6), 1087-1092.

[8] http://www.gams.com , 12th of December, 2013

[9] Alkaya, A.F., Duman, E. (2010) A New Generalization of the Traveling Salesman Problem. Applied Computational Mathematics, 9, (2), 162-175.

[10] Alkaya, A.F. (2009) Optimizing the Operations of Electronic Component Placement Machines. Unpublished PhD Thesis, Marmara University Institute for Graduate Studies in Pure and Applied Sciences, Istanbul, Turkey.

[11] Skiscim, C.C., Golden, B.L. (1983) Optimization by Simulated Annealing: A Preliminary Computational Study for TSP. IEEE Proceedings of the Winter Simulation Conference.

[12] Nahar, S., Sahni, S., Shragowitz, E. (1986) Simulated Annealing and

Combinatorial Optimization. IEEE 23rd Design Automation Conference, Paper 16.1, 293-296.

[13] Lam, J., Delosme, J.M. (1988) An Efficient Simulated Annealing Schedule: Derivation. Technical Report 8816, Yale Electrical Engineering Department, New Haven, CT.

[14] Johnson, D.S. (1990) Local Optimization and the Traveling Salesman Problem. Proceeding ICALP '90 Proceedings of the 17th International Colloquium on Automata, Languages and Programming, 446-461.

[15] Johnson, D.S., Aragon, C.R., McGeoch, L.A., Schevon, C. (1991) Optimization by Simulated Annealing: An Experimental Evaluation; Part II, Graph Coloring and Number Partitioning. doi:10.1287/opre.39.3.378, Vol. 39, No. 3, 378-406.

[16] Martin, O., Otto, S.W., Felten, E.W. (1991) Large-Step Markov Chains for the Traveling Salesman Problem. Complex Systems, 5, 299-326.

[17] Martin, O., Otto, S.W., Felten, E.W. (1992) Large-Step Markov Chains for TSP Incorporating Local Search Heuristics. Operations Research Letters, 11, (4), 219-224.

[18] Cardoso, M.F., Salcedo, R.L., de Azevedo, S.F. (1994) Nonequilibrium Simulated Annealing: A Faster Approach to Combinatorial Minimization. Industrial and Engineering Chemistry Research, 33, 1908-1918.

[19] Abramson, D., Krishnamoorthy, M., Dang, H. (1999) Simulated Annealing Cooling Schedules for the School Timetabling Problem. Asia-Pacific Journal of Operational Research, Vol. 16, No. 1, 1-22.

[20] Ma, J., Tian, P., Zhang, D.M. (1999) Application of the Simulated Annealing Algorithm to the Combinatorial Optimization Problem with Permutation Property: An Investigation of Generation Mechanism. European Journal of Operational Research, 118, 81-94.

[21] Ohlmann, J.W., Thomas, B.W. (2007) A Compressed-Annealing Heuristic for the Traveling Salesman Problem with Time Windows. INFORMS Journal on Computing, Vol. 19, No. 1, 80-90.

[22] Li, Y., Zhou, A., Zhang, G. (2011) Simulated Annealing with Probabilistic Neighbourhood for Traveling Salesman Problems. IEEE Seventh International Conference on Natural Computation, 1565-1569.

[23] Lin, S., Kernighan, B.W. (1973) An Effective Heuristic Algorithm for the Traveling-Salesman Problem. Operations Research, 21, (2), Doi:10.1287/opre.21.2.498, 498-516.

[24] Johri, A., Matula, D. (1982) Probabilistic Bounds and Heuristic Algorithms for Coloring Large Random Graphs. Technical Report 82-CSE-6, Southern Methodist University, Dallas, Texas, USA.

[25] van Laarhoven, P.J., Aarts, E.H. (1987) Simulated Annealing: Theory and Applications, Springer.

[26] Singh, A. (2009) An Artificial Bee Colony Algorithm for the Leaf-Constrained Minimum Spanning Tree Problem. Applied Soft Computing, 9, 625-631.

[27] Pacurib, J.A., Seno, G.M.M., Yusiong, J.P.T. (2009) Solving Sudoku Puzzles using Improved Artificial Bee Colony Algorithm. 4th International Conference on Innovative Computing, Information and Control (ICICIC), 885-888.

[28] Sundar, S., Singh, A. (2010) A Swarm Intelligence Approach to the Quadratic Minimum Spanning Tree Problem. Information Sciences, 180, 3182-3191.

[29] Zhou, G., Gen, M. (1998) An Effective Genetic Algorithm Approach to the Quadratic Minimum Spanning Tree Problem. Computers and Operations Research, 25, 229-237.

[30] Soak, S.-M., Corne, D.W., Ahn, B.-H. (2006) The Edge-Window-Decoder Representation for Tree-Based Problems. IEEE Transactions on Evolutionary Computation, 10, 124-144.

[31] Pan, Q., Tasgetiren, M.F., Suganthan, P.N., Chua, T.J. (2011) A Discrete Artificial Bee Colony Algorithm for the Lot-Streaming Flow Shop Scheduling Problem. Information Sciences, 181, 2455-2468.

[32] Pan, Q., Tasgetiren, M.F., Suganthan, P.N., Chen, A.H.L. (2011) A Discrete Artificial Bee Colony Algorithm for the Total Flowtime Minimization in Permutation Flow Shops. Information Sciences, 181, 3459-3475.

[33] Karaboga, D., Gorkemli, B. (2011) A Combinatorial Artificial Bee Colony Algorithm for Traveling Salesman Problem. International Symposium on Innovations in Intelligent Systems and Applications.

[34] Zhang, X., Bai, Q., Yun, X. (2011) A New Hybrid Artificial Bee Colony Algorithm for the Traveling Salesman Problem. IEEE 3rd International

Conference on Communication Software and Networks (ICCSN).

[35] Banharnsakun, A., Achalakul, T., Sirinaovakul, B. (2010) ABC-GSX: A Hybrid Method for Solving the Traveling Salesman Problem. Second World Congress on Nature and Biologically Inspired Computing, 7-12.

[36] Szeto, W.Y., Wu, Y., Ho, S.C. (2011) An Artificial Bee Colony Algorithm for the Capacitated Vehicle Routing Problem. European Journal of Operational Research, 215, 126-135.

[37] Karabulut, K., Tasgetiren, M.F. (2012) A Discrete Artificial Bee Colony Algorithm for the Traveling Salesman Problem with Time Windows. IEEE World Congress on Computational Intelligence (WCCI).

[38] Dongli, Z., Ximping, G., Yinggan, T., Yong, T. (2012) An Artificial Bee Colony Optimization Algorithm based on Multi-Exchange Neighbourhood. 4th International Conference on Computational and Information Sciences, 211-214.

[39] Duan, J., Han, Y., Yang, Y., Zhang, M., Yun, B. (2012) Minimizing the Total Flowtime Flowshop with Blocking using a Discrete Artificial Bee Colony. Advanced Intelligent Computing Theories and Applications, Lecture Notes in Computer Science, Vol. 6839, 91-97.

[40] Malek, M., Guruswamy, M., Pandya, M., Owens, H. (1989) Serial and Parallel Simulated Annealing and Tabu Search Algorithms for the Traveling Salesman Problem. Annals of Operations Research, 21, 59-84.

[41] Henderson, D., Jacobson, S.H., Johnson, A.W. (2003) Chapter 10: The Theory and Practice of Simulated Annealing. Kluwer Academic Publishers.

[42] Nahavandi, N., Kashan, M.H., Kashan, A.H. (2012) DisABC: A new Artificial Bee Colony Algorithm for Binary Optimization. Applied Soft Computing, 12, 342-352.

[43] Akay, B., Karaboga, D. (2012) A Modified Artificial Bee Colony Algorithm for Real-Parameter Optimization. Information Sciences, 192, 120-142.

[44] Rodriguez, F.J., Lozano, M., Garcia-Martinez, C., Gonzalez-Barrera, J.D. (2013) An Artificial Bee Colony Algorithm for the Maximally Diverse Grouping Problem. Information Sciences, 230, 283-196.

[45] Duman, E., Elikucuk, I. (2013) Solving Credit Card Fraud Detection Problem by the New Metaheuristics Migrating Birds Optimization. IWANN, Part II, LNCS

7903, 62-71.

RESUME

SAMET TONYALI

Marmara University, Goztepe Campus

Engineering Faculty, Computer Science and Engineering Dept.

Room: MB453, PK:34722, Kadikoy, Istanbul, Turkiye

Phone (Cell)+90-537-591-1868 (W)+90-216-348-0292 ext 253

e-mail: samet.tonyali@marmara.edu.tr, samettonyali29@gmail.com

Web: http://mimoza.marmara.edu.tr/~samet.tonyali/

EDUCATION

M.S., Computer Science Marmara University, Faculty of Engineering, Istanbul, Turkey, Ongoing. *Thesis Topic:* Solution Approaches for Sequence Dependent Traveling Salesman Problem, *Advisor*: Assist. Prof. Ali Fuat ALKAYA. GPA: 3.86

B.S., Computer Science Marmara University, Istanbul, Turkey, 2011. GPA: 3.64

WORK EXPERIENCE

2012-... *Teaching Assistant*, Marmara University, Computer Science and Engineering Dept, Istanbul, Turkey

2009-2011 *Student Assistant*, Computer Science and Engineering Dept., Marmara University, Istanbul, Turkey.

Summer 2010 *Trainee,* Computer Science Dept., Ozyegin University, Istanbul, Turkey.

Summer 2009 *Trainee,* Computer Science and Engineering Dept., Marmara University, Istanbul, Turkey.

PROFESSIONAL SKILLS

ALES (2012): 94,09; 74,367; 92.333 (Q, V, EW scores, resp.)

KPDS (2012): 77,5; ÜDS (2012): 88,75

IELTS (2013): Listening: 6.5 Reading: 6.5 Writing: 5.5 Speaking: 6.5 Overall Band Score: 6.5

GRE (2013): Verbal Reasoning: 147 Quantitative Reasoning: 164 Analytical Writing: 3.0

Experienced in LibreOffice (Writer, Calc, Impress)

Effective Use of Java and C programming languages in different platforms (Linux and Windows) and with several IDEs (Eclipse and NetBeans)

PUBLICATIONS

Tonyali, S.; Alkaya, A.F.; Agaoglu, M.: "Neighbour Function Based Solution Approaches for Sequence Dependent Traveling Salesman Problem", *The 9th International Conference on Optimization: Techniques and Appllications (ICOTA 9),* pp. 117-118, Taipei, Dec. 12-16, (2013).

RESEARCH INTERESTS

Combinatorial Optimization, Machine Learning, Artificial Intelligence, Networks and Security

RESEARCH PROJECTS

Solution Approaches for Sequence Dependent Traveling Salesman Problem, BAPKO FEN-C-YLP-150513-0185

Application and Performance Comparison of Metaheuristics for a Generalization of the Traveling Salesman Problem, BAPKO FEN-A-150513- 0172

FOREIGN LANGUAGES

English (fluent)

AWARDS AND HONORS

Bsc. Graduation 2nd among 60 graduates

TUBITAK Msc. Scholarship

In the first 5000 (4021) among 1.5 million test-takers in the University Entrance Exam 2006