T.C.

MARMARA UNIVERSITY

INSTITUTE FOR GRADUATE STUDIES IN

PURE AND APPLIED SCIENCES

OPTIMIZING THE PLACEMENT OPERATIONS OF CHIP MOUNTER MACHINES

Hüseyin Demirkale

THESIS

FOR THE DEGREE OF MASTER OF SCIENCE

IN

COMPUTER SCIENCE AND ENGINEERING

SUPERVISOR

Assist. Prof. Arzu Baloğlu

CO-SUPERVISOR

Assoc. Prof. Ekrem DUMAN

ISTANBUL 2010

T.C.

MARMARA UNIVERSITY

INSTITUTE FOR GRADUATE STUDIES IN

PURE AND APPLIED SCIENCES

OPTIMIZING THE PLACEMENT OPERATIONS OF CHIP MOUNTER MACHINES

Hüseyin Demirkale

(141100320070242)

THESIS

FOR THE DEGREE OF MASTER OF SCIENCE

IN

COMPUTER SCIENCE AND ENGINEERING

SUPERVISOR

Assist. Prof. Arzu Baloğlu

CO-SUPERVISOR

Assoc. Prof. Ekrem DUMAN

ISTANBUL 2010

ACKNOWLEDGMENT

I would like to thank my supervisor Assoc. Prof. Ekrem DUMAN and Assist. Prof. Ali Fuat ALKAYA for continual encouragement, support, advices, and extensive knowledge throughout the study. I am also grateful to Assist. Prof. Arzu BALOĞLU for her acceptance of my thesis for co-supervising.

It was a great pleasure for me to become involved in this research study with Assist. Prof. Ali Fuat ALKAYA.

Many thanks to my friends and instructors in the Computer Science and Engineering Department for their support and friendship.

The financial support of this study by the Scientific and Technological Research Council of Turkey (TÜBİTAK) through the project 108M198 is gratefully acknowledged.

June 2010 **Hüseyin Demirkale**

CONTENTS

ABSTRACT

OPTIMIZING THE PLACEMENT OPERATIONS OF CHIP MOUNTER MACHINES

The wide usage of printed circuit boards (PCBs) in numerous electronic products has placed a significant demand for PCBs. This demand directs the researchers to decrease the drawbacks of printed circuit board production where placements machines are used. Main optimization problems of placements machines are the placement sequencing problem and assignment of component types to the feeders, also called feeder configuration problem. These problems are called NP-Complete problems.

This thesis focuses on optimizing feeder configuration and placement sequencing problems on the operations of chip mounter and chip shooter machines. The similarity of these machines is that they have rotational turret, board carrier and feeder magazine which holds the components.

The research begins with literature survey of exact methods and GAMS platform with different solvers. A new generalized version of Traveling Salesman Problem (TSP) is implemented with GAMS platform which is called as the Sequence Dependent TSP (SDTSP). The nonlinear formulation of the SDTSP is explained. The results of the exact methods are examined in computational complexity and time domain. Also the results are compared in hardware requirement according to the size of the problem.

The thesis continues with exploring the artificial bee colony algorithm on chip shooter and chip mounter machines. It is very time consuming to solve problems with exact methods, for this reason, a metaheuristic is implemented and tested on the SDTSP. Furthermore, ABC is implemented to the problems of chip mounter machines and the results of ABC is compared with the exact solutions. Computational results are presented to demonstrate the effectiveness of the metaheuristics. It is shown that ABC is a promising algorithm for solving combinatorial optimization problems.

June, 2010 **Hüseyin DEMİRKALE**

ÖZET

ÇİP PARÇA YERLEŞTİRİCİ MAKİNELERİN OPTİMİZASYONU

Baskı devre kartlarının günümüzde yaygın bir biçimde kullanılması onlara olan talebi önemli ölçüde arttırmıştır. Bu da araştırmacıları, üretimde meydana gelen engellerin en aza indirgenmeye yöneltmiştir. Üretimde meydana gelen aksaklığın kaynağı dizgi makineleridir. Basılı devre kartlarının üretiminde karşılaşılan darboğaz, dizgi makinelerinden gelen iki problemden kaynaklanmaktadır. Bunlar sırasıyla, parça montaj sırasının belirlenmesi ve parçaların besleme hücrelerine bölünmesi problemleridir. Bu problemler, NP-Zor problemler olarak adlandırılmaktadırlar ve en iyi çözümleri bulan yöntemler ile birlikte çözümlere sadece küçük boyuttaki örneklerde ulaşılır.

Araştırma, çip parça yerleştirici ve çip parça saçıcı makinelerinin işlemlerini eniyilemeye odaklanmıştır. Bunu yaparken de, montaj sıralaması ve besleme konfigürasyonu problemleri ele alınmıştır. Her iki makinede döner tarete sahiptir. Problemde en iyi sonuca ulaşmak için bu iki problemin aynı anda çözülmesi gerekmektedir.

Tez kesin çözüm yöntemlerinin ve problem çözümünde kullanılan GAMS geliştirme aracı üzerinde yazın incelemesi ile başlamaktadır. Bununla alakalı olarak Sıraya Dayalı Gezgin Satıcı Probleminin (SDGSP) için doğrusal olmayan tamsayılı programlama formulasyonu verilmiştir ve GAMS geliştirme aracı üzerinde kodlanıp sonuçları incelenmiştir. Tez SDGSP probleminin sayısal analizi ve sayısal karmaşıklığının incelenmesiyle devam etmiştir. Problemin çözümünde gereken zaman, oluşan karmaşıklık ve bu karmaşıklığa bağlı olarak ihtiyaç duyulan donanımsal yapı incelenmiştir. Sayısal analiz ve sayısal karmaşıklık bölümlerinde problem çözümlerinin araştırılması sırasında çip parça yerleştirici makinelerin parça sayısı ve besleme konfigürasyonu ayrı ayrı test edilmiştir.

Tez çip parça saçıcı makinelerinin eniyileme çözümlerinin araştırılmasıyla devam etmektedir. Kesin çözüm yöntemleri ile en iyi çözümleri elde etmek çok zaman aldığından, sezgisel yöntem olan yapay arı kolonileri algoritması geliştirişmiştir. Bundan başka, çip yerleştirici makineler icin yapay arı kolonileri algoritması uygulanmıştır. Algoritmanın başarısı sayısal sonuçlarla gösterilmiştir.

Haziran, 2010 **Hüseyin DEMİRKALE**

SYMBOLS

CF1 **:** Travel cost of board carrier in chip mounter machines

c_{ij} **:** Cost of travel from city *i* to city *j*

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

d **:** Distance between two components

D_{ijp} **:** The dominating factor in C_{ijp} definition

e **:** Edge

E **:** Set of edges

G **:** Graph consisting of edges and vertices

g_{tk} **:** Group (weight category) matrix (*n x K*) indicating the group of components

H **:** Number of heads

K **:** Number of weight categories

n **:** Number of nodes

N **:** Total number of components to be placed

N_k **:** Number of components in each weight category *k*

npz **:** No pick up zone for chip mounter machines

p **:** The placement order or placement position.

R **:** Feeder number

R **:** Number of feeder slots

tt_k **:** Turret rotation time for each weight category *k*.

u **:** Feeder slot constraint

V **:** Set of vertices

v **:** Speed of board carrier

w_{ijp}**:** **:** Binary variable which takes the value of 1 if node *j* is visited in p^{th} position after node *i* is visited in $(p\text{-}1)^{th}$ position

$\mathbf{X^B}$ **:** Basic variable

x_{ip} **:** Binary variable which takes the value of 1 if node *i* is visited in position *p*.

$\mathbf{X^N}$ **:** Non-basic variable

y **:** Integer solution

y_{tr} **:** Binary variable which takes the value of 1 if component type *t* is assigned to feeder slot *r*.

ABBREVIATIONS

ABC : Artificial Bee Colony

BC : Board Carrier

BONMIN : Basic Open-Source Nonlinear Mixed Integer programming

ER : Equality Relaxation

GRG : Generalized Reduced Gradient algorithm.

IP : Integer Programming

KKT : Karush-Kuhn-Tucker

LB : Lower Bound

LP : Linear Programming

MINLP : Mixed Integer Nonlinear Programming

MIP : Mixed Integer programming

NLP : Nonlinear Programming

OR : Outer Approximation Algorithm

PCB : Printed Circuit Boards

SDTSP : Sequence Dependent Traveling Salesman Problem

SMT : Surface Mount Technology

TSP : Traveling Salesman Problem

UB : Upper Bound

FIGURES

PAGE NO

TABLES

CHAPTER I

INTRODUCTION

Printed circuit boards (PCB) are used to connect electronic components using conductive methods in wide range of industry. These boards are used in all electronic goods which are assembled with placement machines. Originally, every electronic component must be wired in order to operate. For this purpose, PCB is produced to hold the electronic components with the help of holes on it. Generally they are designed specially and make the construction of goods very easy.

When mounting the components on PCB, surface mount technology (SMT) is used. SMT is the method of constructing electronic circuits where components are mounted directly to the surface of board. Nowadays, SMT machine industry has come of age and replace the through hole technology (Manko, 1995).

Because of the wide usage of printed circuit boards and having hundreds of components on it make the importance of the assembly of PCB more important in global industry. For that reason, to have effective production of PCB, new mathematical models are developed and implemented on several machines.

In industry, there are several type of machines that are used to assemble PCBs. These machines have several parts such as rotational turret, board carrier and feeder magazine. All these parts operate concurrently in order to mount components on boards.

The machine type that we use in our thesis is chip mounter machines. Chip mounter machines have a rotational turret and board carrier (BC). Although the parts operate independently, the solution of one part effects the other.

Generally speaking, the operations of these machines yield two major problems (Duman,1998). These are allocating the components to feeder slots and determination of the placement sequence of these components. The placement sequence problem is a generalized version of TSP which is called as the sequence dependent traveling salesman (SDTSP) problem.

In a chip mounter machine, there is a stationary feeder magazine, feeder slots and board carrier. Board carrier moves in x-y directions and aligns the component hole to the next head of turret. On rotational turret, there are heads which carries the electronical components from feeder magazines to board. The turrets are capable of grabbing different types and groups of components.

The chip shooter machines are similar to the chip mounter machines. The only difference is the moveable feeder magazine. The feeder magazine of the chip shooter machines move horizontally and aligns the components to turret head.

I.1 SCOPE AND AIM

The aim of thesis study is to implement exact methods on chip mounter machines and prove that these methods are not suitable for large size of problems. The SDTSP problem is explained and implemented with exact methods using GAMS (Alkaya and Duman 2009).

In chip shooter machines, one of the most recent algorithms, artificial bee colony, (ABC), is implemented. This algorithm is motivated from the behaviour of honey bees. In that algorithm, three types of bees, scout, onlooker and employer, are analyzed and according to their movements the placement operation of chip shooter machine is developed.

Next, ABC algorithm on chip shooter machines is analyzed.

In the methodology of thesis study, first the exact methods are examined and identified. The formulation of chip mounter machine is studied (Alkaya and Duman 2009).

A literature survey about exact methods, chip mounter machines and chip shooter machines is made.

Then, the formulation of chip mounter machine was implemented and computational analyse are made.

Afterwards, ABC algorithm is implemented for chip shooter machines.

CHAPTER II

GENERAL BACKGROUND

In this chapter, we plan to give a general background and literature survey for exact methods, printed circuit boards and GAMS optimization platform. Thus, section II.1 gives a general background for exact methods. Then, in the following subsections the problems that are analyzed are given along with their solution approaches in the literature.

II.1 TRAVELING SALESMAN PROBLEM

The Traveling Salesman Problem (TSP) is one of the most widely studied IP problems. Adding new constraints to the problem yields different generalizations to the problem and each new generalization forms the basis of a new research area. Mostly known generalizations are Asymmetric TSP, Vehicle Routing Problem (VRP) and its variants. TSP is observed in many research areas. One can easily argue that there is uncountable number of studies towards solving it since it is introduced to the literature, ranging back to at least the late 1940's. The TSP is the focus of interests for many research disciplines (mostly computer scientists, mathematicians and industrial engineers) because, even after about half a century of research, the problem has not been completely solved. TSP or its variants can be applied to solve many realistic problems within our daily lives.

It can be easily stated as follows: A salesman wants to visit n distinct cities and then returns home. He wants to determine the sequence of the travel so that the overall travelling distance is minimized while visiting each city not more than once. Conceptually it seems simple, but obtaining an optimal solution requires a search in set with $n!$ feasible solutions. TSP can be represented on a graph $G=(V,E)$, where V is the set of vertices (or cities) and E is the set of edges (or links between the cities). Each edge $e \in E$ has an associated cost (or length) c_e. We define the incidence vector x as

$$x_{ij} = \begin{cases} 1 & \text{if edge between i and j is used} \\ 0 & \text{otherwise} \end{cases}$$

Notice that for a tour, at each vertex the sum of the edge variables must be two; this is called a degree constraint. This leads to the relaxation of the traveling salesman problem:

$$\min \sum c_e x_e \tag{II.1}$$

$$\text{s.t. } \sum\nolimits_{e \in \delta(v)} x_e = 2 \quad \text{for all vertices v} \tag{II.2}$$

Here, $\delta(v)$ denotes the set of all edges incident to vertex *v*. All tours are feasible in this formulation, but it also allows infeasible solutions corresponding to subtours, consisting of several unconnected loops. To force the solution to be a tour, it is necessary to include subtour elimination constraints of the form $\sum_{e \in \gamma(u)} x_e \geq 2$ for every subset $U \subseteq V$ with cardinality $2 \leq |U| \leq V/2$, where $\gamma(U)$ denotes the set of edges with exactly one endpoint in *U*. Any feasible solution to the relaxation given above which also satisfies the subtour elimination constraints must be incidence vector of a tour. One can see that the number of subtour elimination constraints is exponential in the number of cities (Mitchell, 2002).

Another formulation of TSP is done by using variables x_{ij} representing the travel of the salesman from city *i* to city *j*, and c_{ij} representing the cost of travel.

$$\min \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} c_{ij} x_{ij} \tag{II.3}$$

$$\text{s.t. } \sum_{i=1}^{n} x_{ij} = 1, \qquad j = 1, \ldots, n; i \neq j \tag{II.4}$$

$$\sum_{j=1}^{n} x_{ij} = 1, \qquad i = 1, \ldots, n; i \neq j \tag{II.5}$$

$$u_i - u_j + n x_{ij} \leq n - 1 \qquad i, j = 2,3, \ldots, n; i \neq j \tag{II.6}$$

$$x_{ij} \in \{0,1\}, \qquad i = 1, \ldots, n, \qquad j = 1, \ldots, n, \tag{II.7}$$

$$u_i \in Z^+ \qquad (Z^+ = \text{set of positive integers}) \qquad i = 1, \ldots, n, \tag{II.8}$$

In this formulation, first constraint ensures that the salesman arrives once at each city and second constraint ensures that the salesman leaves each city once. Avoidance of subtours is done by third constraint (Winston and Venkataramanan, 2003).

In general TSP, there is no restriction on the cost function. In metric TSP, all edge cost are symmetric and fulfill the triangle inequality: $c_{ij} \leq c_{ik} + c_{kj}$. In Euclidean TSP, vertices correspond to points in a *d*-dimensional space, and the cost function is the Euclidean distance. The Euclidean distance between two points $x=(x_1, x_2,\ldots,x_d)$ and $y=(y_1, y_2, \ldots, y_d)$ is

$$\sqrt{\sum_{i=1}^{d}(x_i - y_i)^2} \qquad \text{(II.9)}$$

(Alkaya, 2009).

The history of TSP is shown in Table II. 2. (Padberg and Rinaldi, 1991) In that Figure, the *Y* axis shows the number of cities that is solved in TSP and *X* axis shows the years. Table II. 2 is a table view of Table II. 2

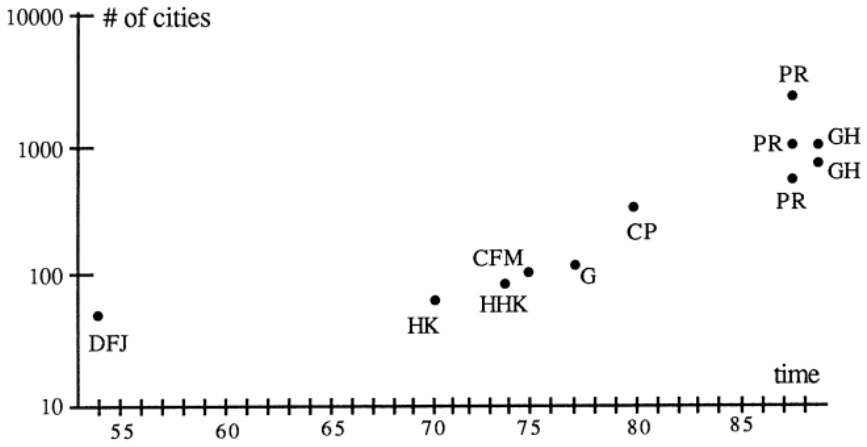

Figure II. 1 History of travelling salesman problem

Table II. 1 Literature review of TSP until 1988

Year	Referance	City	0-1 Variables
1954	DFJ: Dantzig,Fulkerson and Johnson	49	1,076
1970	HK: Held and Karp	60	1,770
1974	HHK: Helbig Hansen and Krarup	80	3,160
1975	CFM: Camerini,Fratta and Maffioli	100	4,950
1977	G: Grotschel	120	7,140
1980	CP: Crowder and Padberg	318	50,403
1987	PR: Padbergand Rinaldi	532	141,246
1987	PR: Padbergand Rinaldi	1,002	501,501
1987	PR: Padbergand Rinaldi	2,392	2,859,636
1988	GH: Grotschel and Holland	666	221,445
1988	GH: Grotschel and Holland	1,000	499,500

II.2 EXACT METHODS

In mathematics, computer science, and related subjects, an algorithm is an effective method for solving a problem using a finite sequence of instructions. Exact algorithms are guaranteed to find an optimal solution and to prove its optimality for every instance of a class of combinatorial optimization problems. In this section, the most poular exact algorithms; branch and bound, branch and cut and cutting plane algorithms are explained.

As mentioned before, traveling salesman problem is attractive topic for the researches because of its simplicity on description but difficulty on solution. In that problen type, with given cities, the aim is to find the minimum cost tour provided that each city is visited once.

Traveling salesman problem is important for researchers because the solution of it represents all combinatorial optimization problems. In other words, the solution encompasses NP complete problems.

In traveling salesman problem, where number of cities is N, there are *(N-1)!* solutions. At first glance, this calculation can be thought as easy in small size problems, but when the number of size increases, it becomes hard to find the optimum solution.

In the following, respectively, branch and bound, cutting plane and branch and cut algorithms are explained on TSP.

II.2.1 Branch and bound algorithm

In first part, branch and bound algorithms are examined. In this algorithm, with given city matrix, initially, the minimum possible tour is computed by finding the minimum edge that leaves each node. These outgoing edges or pathes may not construct a tour but one can say that, the the sum of that outgoing edges never exceeded by other tours. In other words, every node is visited only once and the summation of smallest outgoing edges is the minimum value of possible tours. And if the smallest outgoing edges constructs a tour, than that is the best tour.

Branch and bound algorithm finds the lower bound of the problem with the sum of minimum pathes that leaves the cities. And at each step, solutions are compared with that lower bound. If the current solution is less than lower bound, than algorithm continues with that node and branch with that node. In other words, the current tour is considered and included to the solution result set. Otherwise, branching with that node terminated and pruned.

In the section below, there is step by step explanation of branch and bound algorithm.

Cost matrix is defined in Table II. 2. In that examle, where there are 5 cities.

Table II. 2 Distance matrix between cities

	C1	C2	C3	C4	C5
C1	0	14	4	10	20
C2	14	0	7	8	7
C3	4	5	0	7	16
C4	11	7	9	0	2
C5	18	7	17	4	0

In the initial step of algorithm is defining the lower bound is taken as ∞. After that step, minimum pathes are calculated. As seen from Table II. 2 the minimum edge that leaves the city *C1* is 4. Likewise, The minimum path that leaves *C2* is 7, *C3* is 4, *C4* is 2 and finally *C5* is 4. By adding these values, the lower bound can be calculated as 21. Table II. 3 shows these steps. This lower bound is the initial state

of branch and bound tree. Since value, 21, is lower than ∞, adjacent cities to *C1* are included to the next step. That is to say that, pathes *C1- C2* , *C1- C3*, *C1- C4* and *C1- C5* are added to the tour.

Table II. 3 Calculating the lower bound values

0	14	4	10	20	Min → 14,4,10,20 : 4
14	0	7	8	7	Min → 14,7,8,7 : 7
4	5	0	7	16	Min → 4,5,7,16 : 4
11	7	9	0	2	Min → 11,7,9,2 : 2
18	7	17	4	0	Min → 18,7,17,4 : 4 , Total: 4 + 7 + 4 + 2 + 4 = 21

As seen from Table II. 3, lower bound is calculated as 21. And cities that are adjacent to *C1* are used for next step. First, that part from *C1-C2* is inserted to the new calculations and with the help of formulation (II.7), new lower bound is calculated.

LB (C1, C2) = (C1 - C2) + the smallest outgoing edges in unvisited nodes (II. 10)

And according to example that value is,

14 + 7 + 4 + 2 + 4 = 31 . The value of 14 comes from the distance between C1 and C2.

Likewise, the ditance between C1 and C3 is calculated as;

LB (C1, C3) = (C1 - C3) the smallest outgoing edges in unvisited nodes (II. 11)

And the result is = 4 +7 + 4 + 2 + 4 =: 21

In that way, the algorithm continues with finding a tour and keeping the lower bound value. Than, the lower bound values are compared in each step. The nodes that has higher lower bound values are pruned from the branch and bound tree. Algorithm continues with remaining nodes until it reaches the end of tree which is the optimum solution.

One of the main advantage of branch and bound algorithm is arised in runtime. In every step of algorithm lower bound values are calculated and some nodes are pruned from the branch and bound tree with the help of that lower bound. Also the

calculations of that nodes are pruned. In big problems, these advantage of branch and bound algorithm can be realized clearly.

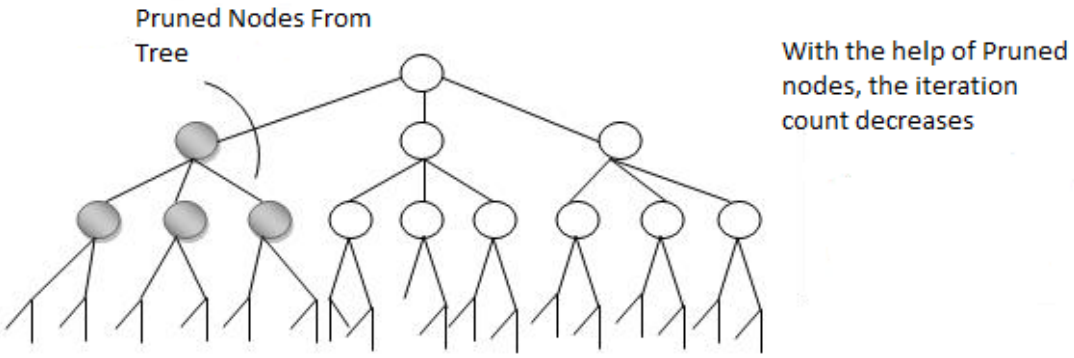

Figure II. 2 Pruned nodes

The next proposed algorithm about branch and bound is known as Little's algorithm. In that algorithm, different from the previous one, there are some constraints which are named as including or excluding constraints. According to these constraints branch and bound tree is constructed. Moreover, lower bound values are calculated with those constraints too. These lower bound values represent the smallest solution that is possible when there are tours below that node in tree. If this lower bound is higher than the best known solution that this node is pruned like first brand and bound algorithm (Little, 1963).

If the pruned nodes are closer to the root of the tree than the algorithm becomes more efficient.

As mentioned above there are including and excluding rules. These rules are given in Table II.4:

Table II. 4 Pruned nodes in branch and bound algorithm

If we exclude an edge,*C1*→*C2*, from the tour and this extraction makes it impossible for that edges to have two adjacent edges, than we must include these nodes. If two cities are included *C*1→*C2* to the tour and this happens to have more than two edges adjacent cities in tour than these cities are excluded. After the include of two nodes a cycle which is not a tour can be formed. In that situation it is excluded from the tour.

Similar to the first algorithm, in order to compute the lower bound, the smallest outgoing edges from each node are added.

In that step, with the including and excluding rules, we construct branch and bound tree and find the optimum value.

Our cost matrix is defined in Table II. 5.

Table II. 5 Distance of matrix of example problem

	C1	C2	C3	C4	C5	C6
C1	0	8	5	3	1	2
C2	8	0	4	9	2	8
C3	5	4	0	9	6	7
C4	3	9	9	0	1	1
C5	1	2	6	1	0	9
C6	2	8	7	1	9	0

Initially, algorithm starts with cities *C1* and *C2*.

In that example, i(x,y) represents including an edge and e(x,y) represents excluding an edge. For example, i(1,2) means, include the path *C1* to *C2* to the tour and e(1,2) exclude the path *C1* to *C2* to the tour.

First of all, the path from *C1* to *C2* is included to the tour. This means the distance between these cities is added to the tour, which is 8.

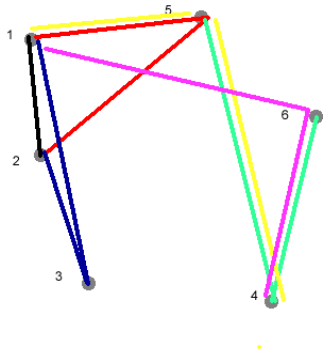

Figure II. 3 Smallest outgoing edges from every node

In the Figure II. 3 according to the cost matrix in Table II. 5 the smallest outgoing edges from every node are shown. Black line means that these edges are included to the tour. With that figure, the lower bound is calculated as;

Smallest edge in node *C1* is → 8+1 = 9. Here, 8 represents the distance between *C1* and *C2*, 1 represents the smallest outgoing edge *C1*→*C5*.

S1=Smallest outgoing edge in node *C2* → 8 + 2 = 10.

S2=Smallest outgoing edge in node *C3* → 4 + 5 = 9.

S3=Smallest outgoing edge in node *C4* → 1 + 1 = 2.

S4=Smallest outgoing edge in node *C5* → 1 + 1 = 2.

S5=Smallest outgoing edge in node *C6* → 1 + 2 = 3.

By adding these values, the lower bound 35 is calculated.

The next step in algorithm is to exclude the edge *C1*→*C2*. Likewise previous calculations;

S(1) = 3 , S(2) = 6, S(3)=9 , S(4) = 2, S(5) = 2 , S(6) = 3 and the sum is 25.

Here, the algorithm continues with lower bound 25 because, the probability of being pruned of this node from the tree is lower than the other one.

In that step, the algorithm continues with calculating the lower bound. And the rules i(*C1*,*C3*) and e(*C1*,*C3*) is applied to the current step. When we exclde the nodes 1 and 3 the lower bound is 26. On the other hand it is 28 when we include it.

i(1-3) → S(1) = 6, S(2)=6, S(3)=9 , S(4)=2, S(5)=2, S(6)=3, the sum is 28,
e(1-3) → S(1) = 3, S(2)=6, S(3)=10 , S(4)=2, S(5)=2, S(6)=3, the sum is 26. And next step continues with node 26.

Next step, continues with e(*C1*,*C4*) and i(*C1*,*C4*). In that step, with including the path *C1*→*C4* to the tour, the lower bound is calculated as 29.

When we exclude e(*C1*,*C4*), it is clear that, until that step, 3 paths are excluded from the tour which are (*C1*-*C2*) , (*C1*-*C3*) and (*C1*-*C4*) .

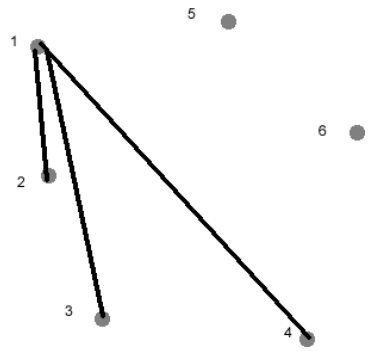

Figure II. 4 The paths that are excluded from the tours

In Figure II. 4, the excluded edges are seen. In that situation, with the help of rules, one can say that, in order to return to node *C1*, at least 2 paths are required. For that reason, paths (*C1*-*C5*) and (*C1*-*C6*) must be included to the tour.

Moreover, we must exclude path (*C5*,*C6*) from tour in order to prevent subtour.

The algorithm continuous until a tour is generated with given constraints. And with the help of the length of that tour, the nodes of branch and bound tree are pruned or branched.

Another approach on branch and bound algorithm is proposed in years 1970 from Held and Karp.(Held and Karp, 1970). In this algorithm, 1-tree is used. For a given node, a 1-Tree is a tree of {2,3,…n} +2 distinct edges connected to node 1. Initially, a node is selected in the given problem and then with remaining nodes, minimum spanning tree is created. And finally, node is connected that spanning tree.

With the help of Kruskal and Prim algorithms, a minimum spanning tree is generated with cities {2,3,4…N} and the node 1 is combined with that tree as in Figure II. 5.

In each step of the algorithm, new spanning tree is generated and according to them new cost calculations are done to find the optimal solution.

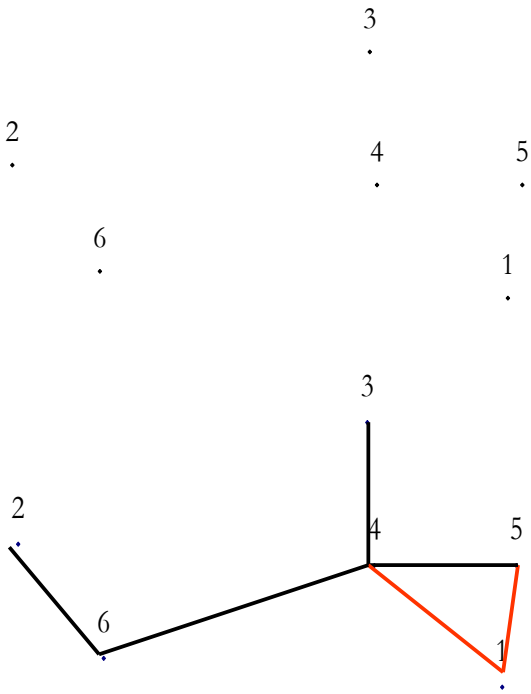

Figure II. 5 Minimum spanning tree and 1-tree relaxation

In that algorithm, the connection between the nodes are important because, with the help of them new minimum spanning trees are created. Likewise previous algorithms the algorithm continues with the help of lower bounds and 1-tree relaxation rules (Held and Karp, 1970).

The algorithm of Held and Karp gives us the lower bounds and these lower bounds help us to create new minimum spanning trees. This lower bound value also shows us the minimum tour under that node. When a tour occurs then, optimum tour is found.

II.2.2 Cutting plane methods

Cutting plane methods are exact algorithms for integer programming problems. They have proven to be very useful computationally in the last ten years, especially when combined with a branch and bound algorithm in a branch and cut framework. These methods work by solving a sequence of linear programming relaxations of the integer programming problem. The relaxations are gradually improved to give better approximations to the integer programming problem, at least in the neighborhood of the optimal solution. For hard instances that cannot be solved to optimality, cutting plane algorithms can produce approximations to the optimal solution in moderate

computation times, with guarantees on the distance to optimality (Mitchell, 2002). A survey of applications of cutting plane methods, as well as a guide to the successful implementation of a cutting plane algorithm can be found in (Jünger et al., 1995). Cutting plane algorithms for general integer programming problems were first proposed by (Gomory, 1958, Gomory, 1963). The Gomory cuts were observed to give poor performance for large scale problems and were neglected for many years. Recently, (Balas et al., 1996a, Letchford and Lodi, 1992) showed that the Gomory cuts can actually be useful. There has been interest recently in other families of cutting planes for IP problems. Two such families are lift-and-project cuts and Fenchel cuts (Balas et al., 1996b, Boyd, 1994). A recent study by (Marchand et al., 2002) is a good survey that presents cutting planes that are useful or potentially useful in solving mixed integer programs.

II.2.3 Branch-and-cut algorithms

Branch-and-cut algorithms are also exact algorithms consisting of a combination of a cutting plane method with a branch-and-bound algorithm. The method solves the LP relaxation of IP problem using simplex algorithm. When an optimal solution is obtained, and this solution has a non-integer value for a variable that is supposed to be integer, a cutting plane algorithm is used to find further linear constraints which are satisfied by all feasible integer points but violated by the current fractional solution. If such an inequality is found, it is added to the linear program, such that resolving it will yield a different solution which is hopefully "less fractional". This process is repeated until either an integer solution is found (which is then known to be optimal) or until no more cutting planes are found.

At this point, the branch and bound part of the algorithm is started. The problem is split into two versions, one with the additional constraint that the variable is greater than or equal to the next integer greater than the intermediate result, and one where this variable is less than or equal to the next lesser integer. In this way new variables are introduced in the basis according to the number of basic variables that are non-integers in the intermediate solution but which are integers according to the original constraints. The new linear programs are then solved using the simplex method and the process repeats until a solution satisfying all the integer constraints is found.

One aspect of a branch-and-cut approach that should not be overlooked is that it can be used to provide bounds. In particular, if we are minimizing but we are unable to prove optimality, a lower bound on the optimal value can be deduced from the algorithm, which can be used to provide a guarantee on the distance from optimality. Therefore, for large and/or hard problems, branch-and-cut can be used in conjunction with heuristics or metaheuristics to obtain a good (possibly optimal) solution and also to indicate how far from optimality this solution may be (Mitchell, 2002).

(Padberg and Rinaldi, 1991) introduce a branch-and-cut algorithm for solving large scale instance of TSP to optimality. (Jünger et al, 1995) is good reference to cutting plane algorithms and branch-and-cut algorithms from an implementer's point of view. Recently, (Mitchell, 2002) describes how a branch-and-cut algorithm can be tailored to a specific IP problem, and how families of general cutting planes can be used to solve a wide variety of problems.

II.3 GAMS PLATFORM

Before mention about the problem and solvers, it is good to explain the general structure of GAMS platform.

The General Algebraic Modeling System (GAMS) is high-level modeling system which can be used for mathematical modeling and optimization. This platform consists of a compiler and several solvers. These solvers are capable of handling high-level and large scale problems.

GAMS platform has a compiler thus it has a programming language which is similar to other programming languages. One can write the model to its editor and then execute it with different solvers on different computers when it is loaded to each platform. GAMS has the capability of solving models in different types (linear Programming, nonlinear programming) and it allows to solve the same model with different algorithms and solvers. So without changing the model representation code, one can use different algorithms. As seen in Figure II.6 , in the solver list of GAMS, there are BARON, CONOPT3, MINOS, SNOPT and other solvers which are built for nonlinear programs (NLP). All these solvers can solve a model which is a NLP problem. This causes to change the solver easily and benchmark the performance of models easily in different algorithms. In other words, this yields users to find the best way to solve models.

When the formulation above is solved with nonlinear solvers in GAMS, GAMS raises errors because of discrete variables. In order to solve problems that have discrete variables, solvers needed that handle integer programming. Moreover the model above includes nonlinear functions. This brings another advantage of GAMS which generates errors to identify the problem more clearly.

Because of the structure of formulation, in GAMS, DICOPT is used as solver because it handles both nonlinear and integer programming. There are other solvers that solve our problem like BARON. However in our formulation, DICOPT is chosen as solver because the execution time and memory requirement is less than the need of BARON.

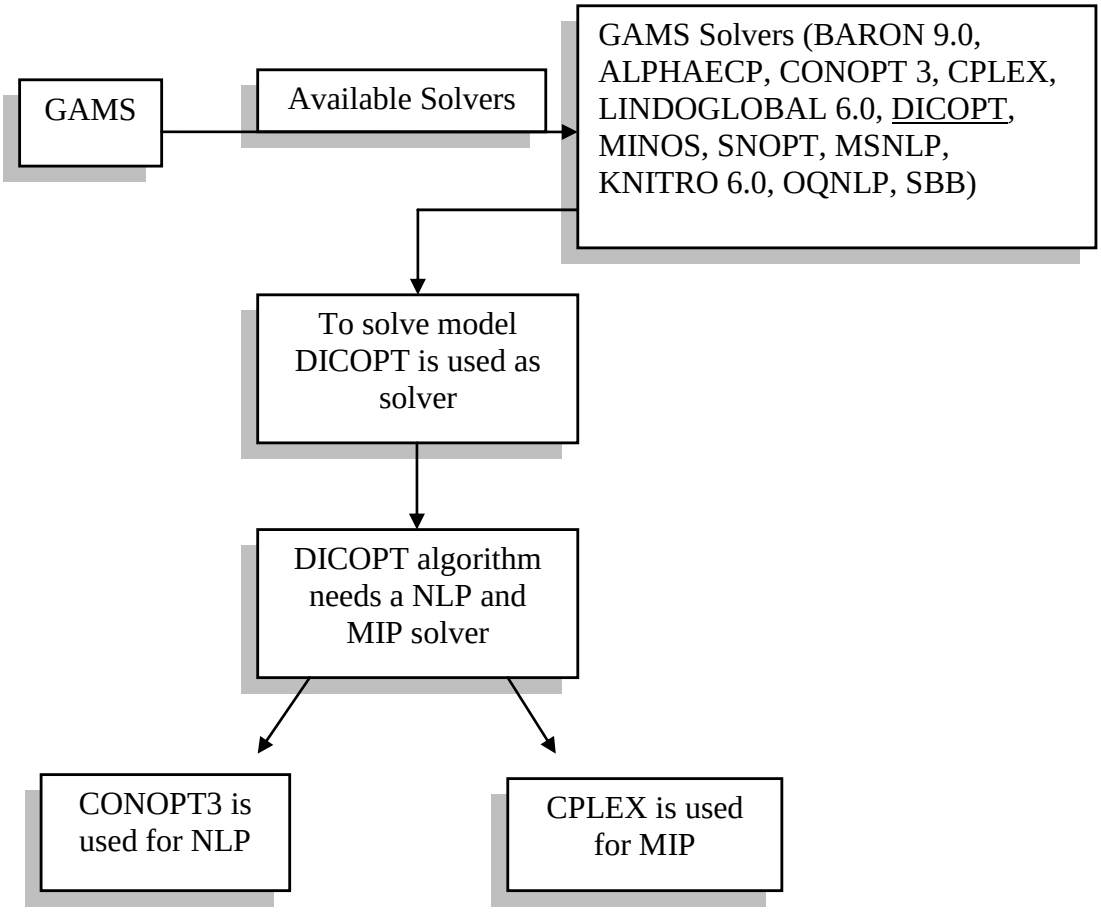

Figure II. 6 Solver strategy of our problem in GAMS

In GAMS, DICOPT is the program that solves mixed integer nonlinear programming (MINLP) problems. In other words, DICOPT solves problems which have integer, linear and nonlinear continuous variables. It is based on the extension of the outer approximation algorithm for the equality relaxation strategy (Duran and

Grossman, 1986). These algorithms solve problems with the help of NLP and MIP solvers that run under GAMS.

While solving the formulation in GAMS, three key ideas are used in DICOPT algorithm. These are;

- Outer approximation
- Equality Relaxation and
- Augmented Penalty

Duran and Grossman proposed the outer approximation algorithm to reduce the number of NLP sub problems (Duran and Grossman, 1986). So, the objective of the outer approximation algorithm is to provide polyhedral representation of the solution space of the program Figure II.8. This solution method renders the linearity in the continuous variables and enables to replace the difficult MINLP program with a mixed integer linear program. In that algorithm, outer approximations are attained by generating linearization at each step and accumulating them to get successively linear approximations of nonlinear convex functions that underestimate the objective function and overestimate the feasible region. In outer approximation algorithm nonlinear equations must be eliminated algebraically or numerically.

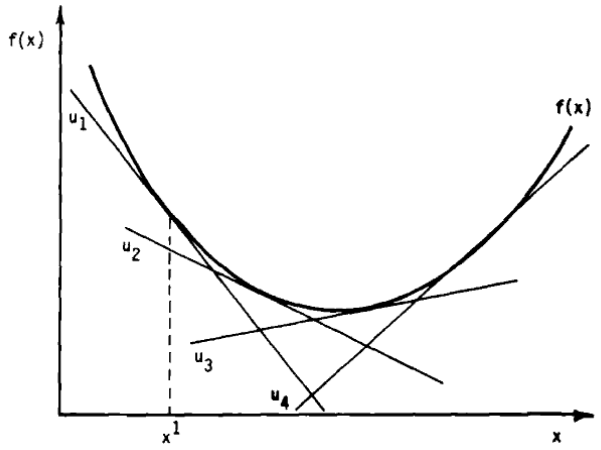

Figure II. 7 Outer Approximation of a convex function at four points

When there is a MINLP problem that contains nonlinear functions *NLP(x, y)*, one can always reformulate the problem that is linear in *y* and nonlinear in *x*. As a result a master problem (MILP) is generated. This MILP master problem includes the linear constraints from the original MINLP problem and the linear approximations to the nonlinear functions are derived in each step of solved NLP problems.

Solving the master problems gives good approximations to the original MINLP problem and, moreover, good lower bounds on the objective functions. Thus that algorithm consists of solving alternating finite sequence of nonlinear problems and relaxed versions of integer linear master problems. Below there is a that shows the steps and algorithm in that solution method.

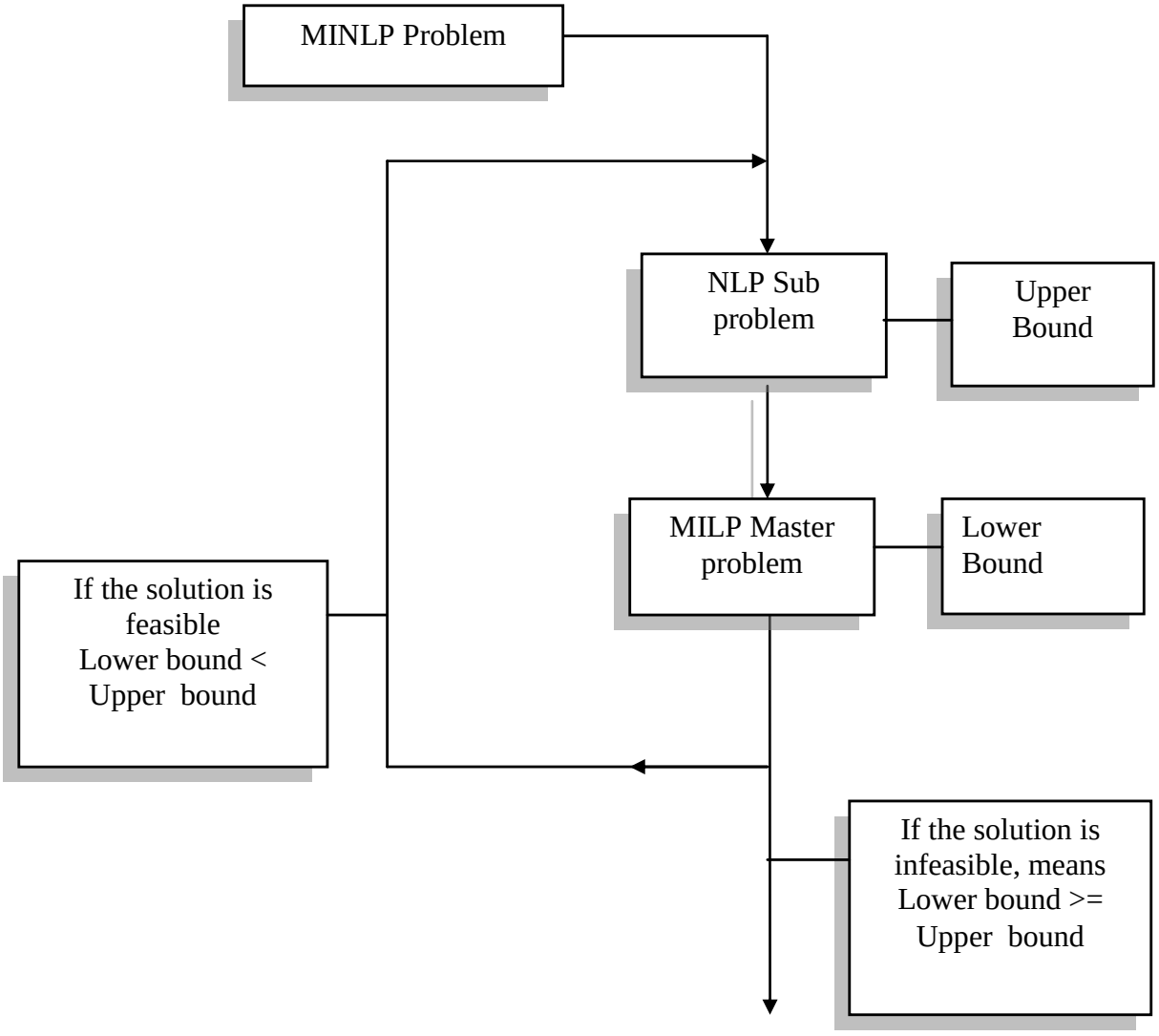

Figure II. 8 Solution steps in MINLP problems

As seen from the Figure II.8, MINLP algorithm alternates between NLP and MILP problems. First, NLP sub problem is generated for the main problem. The result of that sub problem gives the Upper bound to the original problem. After that relaxed MILP master program is generated and this gives a lower bound to the problem. If the solution is feasible which means that Lower bound < Upper Bound,

continue to solve problem with generating new NLP sub problems. If Lower bound >= Upper Bound than stop the algorithm (Figure II.8).

Because of the limitations of the outer approximation algorithm, equality relaxation (ER) algorithm was developed. That algorithm involves the solution steps of the outer approximation algorithm (Figure II.8). The difference is that, the master problem is defined in a way that nonlinear functions in the formulation can be handled explicitly.

In outer approximation algorithm in order to get the master problem, we need elimination. And with that, the original sparsity of the MINLP is preserved in the MILP problem.

Table II. 6 shows the algorithm in DICOPT.

Table II. 6 The algorithm of DICOPT

1. Solve the NLP relaxation of the MINLP program. If y(0) = y is integer, stop(“integer optimum found"). Else continue with step 2. 2. Find an integer point y(1) with an MIP master problem that features an augmented penalty function to and the minimum over the convex hull determined by the half-spaces at the solution (x(0); y(0)). 3. Fix the binary variables y = y(1) and solve the resulting NLP. Let (x(1); y(1)) be the corresponding solution. 4. Find an integer solution y(2) with a MIP master problem that corresponds to the minimization over the intersection of the convex hulls described by the half-spaces of the KKT points at y(0) and y(1). 5. Repeat steps 3 and 4 until there is an increase in the value of the NLP objective function. (Repeating step 4 means augmenting the set over which the minimization is performed with additional linearizations - i.e.half-spaces - at the new KKT point).

As seen from the outer approximation and equality relaxation algorithms, DICOPT uses NLP and MILP problems in a sequence, so, when solving a NLP problem NLP solver is used and solving MILP problem a MIP solver is used.

For that reason, to solve formulation, the two needed solvers are;

- CONOPT in NLP sub problems and

- CPLEX in MIP master problems.

While solving the NLP sub problem part of our problem, CONOPT solver is used. In GAMS, there are three versions of CONOPT. These are CONOPT1, CONOPT2 and CONOPT3. CONOPT3 is used for NLP problems because it is designed for large and sparse models and has been developed in recent years. This means the number of variables and equations can be large in models.

The algorithm used in CONOPT is based on Generalized Reduced Gradient (GRG) algorithm. This algorithm is first suggested by Abadie and Carpentier in 1969. And then the large application written in GAMS for large scale problems. Later on the details of the algorithm is found in Drud in years 1989 and 1992.

The idea behind the generalized reduced gradient method is to convert the constrained problem into unconstrained problem by using direct substitution. If this direct substitution is possible than we reduce the number of independent variables and eliminate the constraint equations. Here the methods are extended for linear constraints to apply to nonlinear constraints. As a result the active constraints are continued to satisfy as the point moves from one to another.

In GRG, our problem can be;

$$Min\ f(x) \qquad \text{(II. 12)}$$

Subject to:

$$Ax = b$$

$$x >= 0.$$

In that formulation, A is the m x n function; b is m vector ($m \leq n$) . Here the vector of x can be represented with $x = (X^B , X^N)^T$. X^B is the basic variables and X^N is the non-basic variables. From equation 1, we can write;

$$BX^B + C X^N = b \qquad \text{(II.13)}$$

And

$$X^B = B^{-1} b - B^{-1} C X^N \qquad \text{(II.14)}$$

The idea behind Reduced Gradient algorithm is to eliminate X^B with the help of (3) and consider the problem with the X^N.

The development procedure of the algorithm starts with adding necessary slack and surplus variables to the model.

After that, we define basic, non-basic variables and reduced gradient equation. And then solve the reduced gradient to iteratively to find the feasible solution (Drud, 1985).

The algorithm of GRG is shown in Table II.7:

Table II. 7 GRG Algorithm

1. Initialize and Find a feasible solution.
2. Compute the Jacobian of the constraints, J.
3. Select a set of n basic variables, xb, such that B, the sub- matrix of basic column from J, is nonsingular.
Factorize B. The remaining variables, xn, are called nonbasic.
4. Solve BT u = df=dxb for the multipliers u.
5. Compute the reduced gradient, r = df=dx JT u. r will by denition be zero for the basic variables.
6. If r projected on the bounds is small, then stop. The current point is close to optimal.
7. Select the set of superbasic variables, xs, as a subset of the nonbasic variables that probably can be changed, and nd a search direction, ds, for the superbasic variables based on rs and possibly on some second order information.
8. Perform a line search along the direction d. For each step, xs is changed in the direction ds and xb is subsequently adjusted to satisfy g(xb; xs) = b in a pseudo-Newton process using the factored B from step 3.
9. Go to 2.

To solve integer linear part of model, CPLEX is used as solver. CPLEX is selected as solver because it manages the memory efficiently. Insufficient memory is a very big problem when solving large problems. And when there is a memory problem, CPLEX solver is automatically adjusts the data structure and prevent the insufficient memory errors.

The methods that solve integer programming problems require much computation than the similar sized linear programs. For that reason, integer programming problems requires much time to solve.

When integer programming problems are solved with CPLEX, branch and cut algorithm is used for generated linear sub problems. And as cutting algorithms, it uses multiple type of cutting planes, Heuristics, Cut-off and shortcut techniques. Some cutting plane algorithms are;

- Clique cuts
- Cover cuts
- Disjunctive cuts
- Flow cover cuts
- Flow path cuts
- Gomory fractional cuts
- GUB cover cuts
- Implied bound cuts
- Mixed integer rounding (MIR) cuts
- Zero half cuts

Branch-and-cut algorithms are also exact algorithms consisting of a combination of a cutting plane method with a branch-and-bound algorithm. The method solves the LP relaxation of IP problem using simplex algorithm.

When an optimal solution is obtained, and this solution has a non-integer value for a variable that is supposed to be integer, a cutting plane algorithm is used to find further linear constraints which are satisfied by all feasible integer points but violated by the current fractional solution. If such an inequality is found, it is added to the linear program, such that resolving it will yield a different solution which is hopefully "less fractional". This process is repeated until either an integer solution is found (which is then known to be optimal) or until no more cutting planes are found.

At this point, the branch and bound part of the algorithm is started. The problem is split into two versions, one with the additional constraint that the variable is greater than or equal to the next integer greater than the intermediate result, and one where this variable is less than or equal to the next lesser integer. In this way new variables are introduced in the basis according to the number of basic variables that are non-integers in the intermediate solution but which are integers according to the original constraints. The new linear programs are then solved using the simplex

method and the process repeats until a solution satisfying all the integer constraints is found.

II.4. PRINTED CIRCUIT BOARDS

Printed circuit boards have attractive interest of researchers due to wide range of usage area. We used them in almost every part of our life. In computers, TV, radio and electronic goods are assembled with PCB.

For the PCB types and machine categories, the aim is to optimize the assembly time with the given board configuration. Thus, in this category, the main objectives are feeder configuration problem and placement sequencing problem.

Feeder configuration problem is assigning components to feeder slots and placement sequence problem is to find the order in which components will be placed on board for optimum solution. So, with two problems, our aim is to find optimum feeder configuration and placement sequence.

Likewise TSP problem, printed circuit problems are of great interest for researchers because most of them are generalization of NP-Complete problems.

In this thesis study, the main studies are concerned with chip mounter machines and chip shooter machines. We give the integer programming formulations of it and analyze placement sequence problem in GAMS platform. The machine type considered in this study is the TDK brand, model RX-5A placement machine. This machine uses surface mount technology to deploy PCBs. The machine is explained in detail in (Duman, 2007).

II.5 CHIP MOUNTER PLACEMENT MACHINE

Chip mounter machines consists of three part, one of them is rotational turret, the other is board carrier and component magazine. Rotational turret is responsible for picking up the components from the feeder mechanism and place them to the printed circuit board.

In rotational turret, there are 72 heads. When the turet head reaches the placement location, board moves in cartesian coordinate to align the head to one point. In other words, each head mounts the components in same location (Alkaya and Duman, 2009).

Chip mounter machines are capable of handling component type that have different weight category. If a turret head grabs a heavy component, the rotation

speed decreases. Thus, in our problem, we have 4 rotational speed which are 0.20, 0.23, 0.33 and 0.40s per 5 degrees. On the other hand, board carrier has the speed of 120 mm per seconds.

Component magazine is behind the feeder slots between heads 21-60 and it is stationary. When the rotational turret is over the suitable component, it grabs components and carries it to the placement location.

In next section there is a detailed simulation of chip mounter machines.

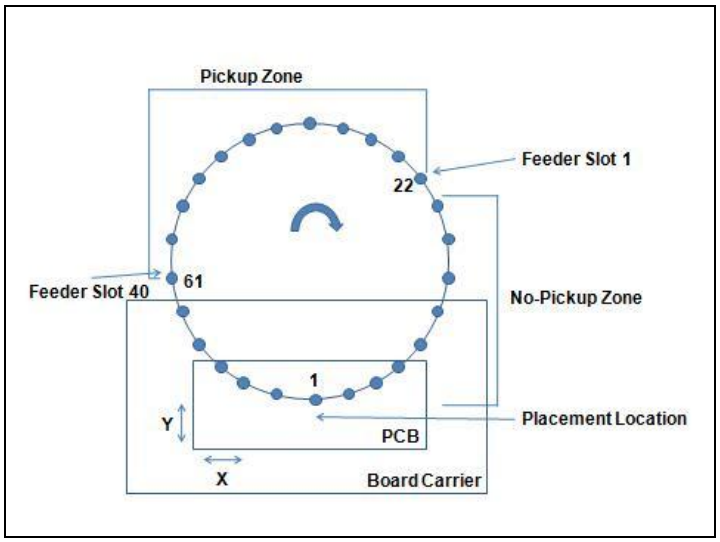

Figure II. 9 TDK chip mounter placement machine

II.6 CHIP SHOOTER PLACEMENT MACHINE

Chip shooter machines consist of three parts. A board carrier, which moves concurrently in two dimensions and carries the board. Feeder carriage that moves horizontally and consistst of type slots. And rotational turret which carries the components to the board and mounts them. These three parts have independent movements and they occur simultaneously.

In chip shooter machine, the feeder carriage moves horizontally and align the component to the turret head. Thus, in order to optimize the problem, we need to configure the order of feeder carriage.

II.7 ARTIFICIAL BEE COLONY ALGORITHM

Artificial bee colony (ABC) algorithm is one of the most recently defined algorithms in literature by Dervis Karaboğa. The main motivation of that algorithm is the movements of honey bees. In real bee colony, every individual bee have some

tasks. And the main objective of them is maximizing nectar amount. They do this issue with efficient division of labour and self-organization (Karaboga and Akay, 2009).

There are three type of bees that try to maximize nectar amount: Employeed bees, onlooker bees and scout bees. Employed bees are responsible for exploiting the nectar resources and giving the necesary information to onlooker bees. After the information comes from the employed bees, onlooker bees decide the food resource with the information that comes from employed bees. Scout bees are responsible for exploiting new food resources depending the results from employed bees and onlooker bees.

The algorithm of ABC is;

1. Initialize the food resource
2. Empoyed bees produces new food resource and exploits the better source.
3. Onlooker bees select a resource with the help of information from employed bees. Produces new food resource.
4. Find the abandoned food resource and create new food resource
5. Repeat the steps from 2 to 3 until stopping criteria is met.

In out thesis study, artificial bee colony algorithm is implemented for chip shooters machines.

CHAPTER III

THE STUDY

In the problem, the main objective is to minimize the assembly time. That is, the objective is to find the best placement sequence and feeder configuration in order to handle the process in the shortest time.

To find the minimum time, the following nonlinear integer programming is built which searches the optimum values of placement sequence and feeder configuration values.

Before definig the problems for chip mounter machine, notations are given in fisrt section (Alkaya , Duman 2009).

N: Number of components that wil be placed on board,

K: Number of weight categories,

N_k: number of components in each weight category k, k=1,2,…,K.

n: number of component types

n_k: number of component types in each weight category k, k=1,2,…,K.

$$\sum_{k=1}^{K} n_k = n$$

R: number of feeder slots (R=60, but no components are assigned to first 20 slots because they correspond to no pickup zone, npz)

H: number of heads (H=71 in our case, excluding placement head)

npz: number of heads in no pickup zone (npz=20 in our case)

ct_{it}: component type matrix (N x n) indicating the component type for each component

$$ct_{it} = \begin{cases} 1 & \text{if component } i \text{ is of component type } t \\ 0 & \text{otherwise} \end{cases} \quad \text{(III. 1)}$$

gtk: group (weight category) matrix (n x K) indicating the group for each component type

$$g_{tk} = \begin{cases} 1 & \text{if component type } t \text{ is in group } k \\ 0 & \text{otherwise} \end{cases} \quad \text{(III. 2)}$$

ttk : turret rotation time for each weight category k, (When k=1, the turret rotation speed is the maximum, that is, turret rotation time is minimum)

p: is the placement order or placement position.

In order to express the placement sequence we need to define decision variable xip which denote the assignment of node i to position p.

$$x_{ip} = \begin{cases} 1, & \text{if node i is visited in p}^{\text{th}} \text{ position} \\ 0, & \text{otherwise} \end{cases} \quad \text{(III. 3)}$$

Decision variable wijp is used for expressing the travel from node i visited in (p-1)th position to node j visited in pth position.

$$w_{ijp} = \begin{cases} 1, & \text{if node j is visited in p}^{\text{th}} \text{ position after node i was visited in (p-1)}^{\text{th}} \text{ position} \\ 0, & \text{otherwise} \end{cases} \quad \text{(III. 4)}$$

On the other hand, in order to express the feeder configuration, we need to define decision variable y_{tr} to state the status of assigning component type *t* to feeder *r*.

$$y_{tr} = \begin{cases} 1, & \text{if component type } t \text{ is stored in feeder } r \\ 0, & \text{otherwise} \end{cases} \quad \text{(III. 5)}$$

r is the feeder number and component magazine is placed behind heads 21-60 and a feeder slot corresponds to each head.

III.1 PLACEMENT SEQUENCING PROBLEM

In placement sequence problem, the aim is to find the optimum placement sequence so the assembly time is completed in minimum time. In that problem, we need to define the time between two placement operations from x to y.

In that problem, we must consider two sub problems. Travel cost and turret rotation time (ttk). First, the travel cost from x to y. We calculate the travel cost with;

$$CF1(x, y) = \frac{d(x, y)}{v} \quad \text{(III. 6)}$$

Here, *d(x, y)* is defined as $\max\{|x_1-y_1|,|x_2-y_2|\}$ where x_1, y_1, x_2 and y_2 are x-y coordinates of the components.

Turret rotation time, ttk is turret rotation time required for the head comes over next placement location. Turret rotation time changes according to the weight category of carried components. If it has a fixed speed value, than it can be treat as a classical TSP problem. But, the changeable speed make problem hard to solve. This

property of turret yields a new problem called sequence dependent traveling salesman (SDTSP) problem (Duman, 2007).

In order to define the rotation time for the turret, heaviest component that is carried must be determined. So, from head 1 to 60, the heaviest component must be find. And this heaviest component determines the current turret time in head 1.

So, the two factors that affects the assembly of PCB, C_{ijp}, in chip mounter machine is turret rotation time and travel cost.

$$C_{ijp} = w_{ijp} \max\left(CF1(i,j), \max_{m=0}^{R-1}\left(\sum_{l=1}^{N}\sum_{t=1}^{n}\sum_{k=1}^{K}\sum_{u=m+1}^{R} tt_k g_{tk} ct_{lt} x_{l,p+m} y_{tu} \right) \right) \quad \text{(III. 7)}$$

In that formulation, $CF1(x,y)$ is the cost for board alignment. And the $\max_{m=0}^{R-1}\left(\sum_{l=1}^{N}\sum_{t=1}^{n}\sum_{k=1}^{K}\sum_{u=m+1}^{R} tt_k g_{tk} ct_{lt} x_{l,p+m} y_{tu} \right)$ finds the maximum turret rotation time.

$$\min \sum_{i=1}^{N}\sum_{\substack{j=1 \\ j\neq i}}^{N}\sum_{p=1}^{N} D_{ijp} \quad \text{(III. 8)}$$

s.t.

$$D_{ijp} - w_{ijp} CF1(i,j) \geq 0 \qquad i,j,p = 1,\ldots,N \quad \text{(III. 9)}$$

$$D_{ijp} - w_{ijp} \sum_{l=1}^{N}\sum_{t=1}^{n}\sum_{k=1}^{K}\sum_{u=1}^{R} tt_k g_{tk} ct_{lt} x_{lp} y_{tu} \geq 0 \qquad i,j,p = 1,\ldots,N \quad \text{(III. 10)}$$

$$D_{ijp} - w_{ijp} \sum_{l=1}^{N}\sum_{t=1}^{n}\sum_{k=1}^{K}\sum_{u=2}^{R} tt_k g_{tk} ct_{lt} x_{l,p+1} y_{tu} \geq 0 \qquad i,j,p = 1,\ldots,N \quad \text{(III. 11)}$$

$\vdots$

$$D_{ijp} - w_{ijp} \sum_{l=1}^{N}\sum_{t=1}^{n}\sum_{k=1}^{K}\sum_{u=R}^{R} tt_k g_{tk} ct_{lt} x_{l,p+R-1} y_{tu} \geq 0 \qquad i,j,p = 1,\ldots,N \quad \text{(III. 12)}$$

$$\sum_{p=1}^{N} x_{ip} = 1, \qquad i = 1,\ldots,N, \quad \text{(III. 13)}$$

$$\sum_{i=1}^{N} x_{ip} = 1, \qquad p = 1,\ldots,N, \quad \text{(III. 14)}$$

$$\sum_{i=1}^{N} w_{ijp} = x_{jp}, \qquad j,p = 1,\ldots,N, \quad \text{(III. 15)}$$

$$\sum_{j=1}^{N} w_{ijp} = x_{i,p-1}, \qquad i,p = 1,\ldots,N, \quad \text{(III. 16)}$$

$$x_{ip} \in \{0,1\}, \qquad i,p = 1,\ldots,N, \quad \text{(III. 17)}$$

$$w_{ijp} \in \{0,1\}, \qquad i,j,p = 1,\ldots,N, \quad \text{(III. 18)}$$

In that formulation, contraint 54 determines that the required placement time is never smaller than the travel cost from i to j. Constraints 55 to 57 determines that, the placement time is never smaller than turret rotation time which is determined with heavy components. Components 58 and 59 determines that each location is occupied only one component and constraints 60, 61 determines if a component in the p^{th} and $(p-1)^{th}$ position than one component precedes in travel sequence.

The above formulation is nonlinear integer programming and the discrete variables are binary variables which appear nonlinearly in the model. And this is formulation of SDTSP.

III.2 ITERATIVE EXAMPLE OF SDTSP

In order to illustrate how the problem is formulated, an example is presented. Consider a board with 6 components, 2 groups and 3 types to be assembled by chip mounter machine. The groups of components identified by one and two with turret time values tt_1=0.20 s and tt_2=0.40 s.

The types, components and groups are shown in Table III. 1

Table III. 1 Components, their types and groups

component number (*i*)	1	2	3	4	5	6
type of component (*t*)	8	8	7	7	9	7
group of component type (*k*)	1	1	1	1	2	1

The groups and types are illustrated in Table III. 2 and Table III. 3;

Table III. 2 Types of components

ct_{it}	7	8	9
1	0	1	0
2	0	1	0
3	1	0	0
4	1	0	0
5	0	0	1
6	1	0	0

Table III. 3 Groups of types

g_{tk}	1	2
7	1	0
8	1	0
9	0	1

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. So, in order to solve the placement sequence problem, an initial feeder configuration can be as in Table III. 4. In that example, heads 1 and 2 are assumed to be in the no-pick up zone.

Table III. 4 An initial feeder configuration

y_{tr}	1	2	3	4	5
7	0	0	1	0	0
8	0	0	0	1	0
9	0	0	0	0	1

In steps 1 to 6, there is an explanation of the placement sequence problem that is based on example above. When we look at the Table III. 4 and Figure III. 1, respectively feeder 3, 4, 5 contains the types t_7, t_8, t_9. Interconnected to the model, one can say that, this initial state is not good for placement sequence problem because type 7 carries the lighter components which means heavier components allocates more time in feeders. Thus, the placement speed decreases and objective value increases. Feeder 1 and 2 are in no pick up zone that means they cannot grab components from feeders.

This is the initial state of the example.

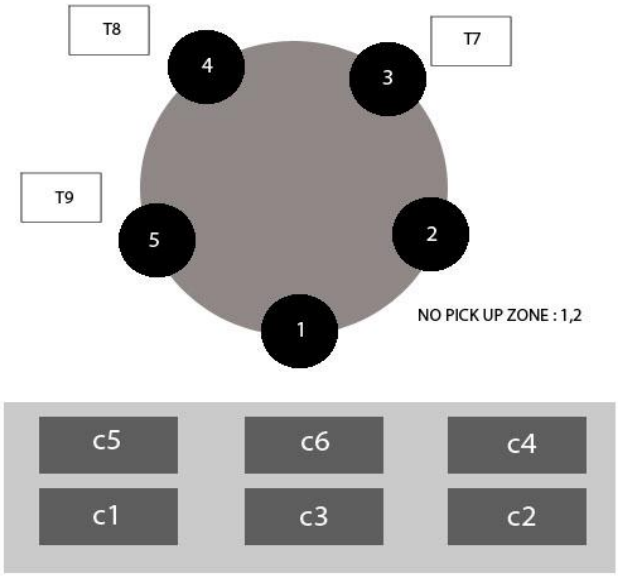

Figure III. 1 Initial state

Step 1: In first placement, component 5 will be placed on the board and it is being carried by head 1 to the placement location. This placement is not assumed. In other words, it comes from the previous placement operations. During that placement under the feeders 3 and 4, components 4 and 1 is grabbed because type of component 4 matches the type stored in slot 3 and type of component 1 matches the type stored in slot 4.

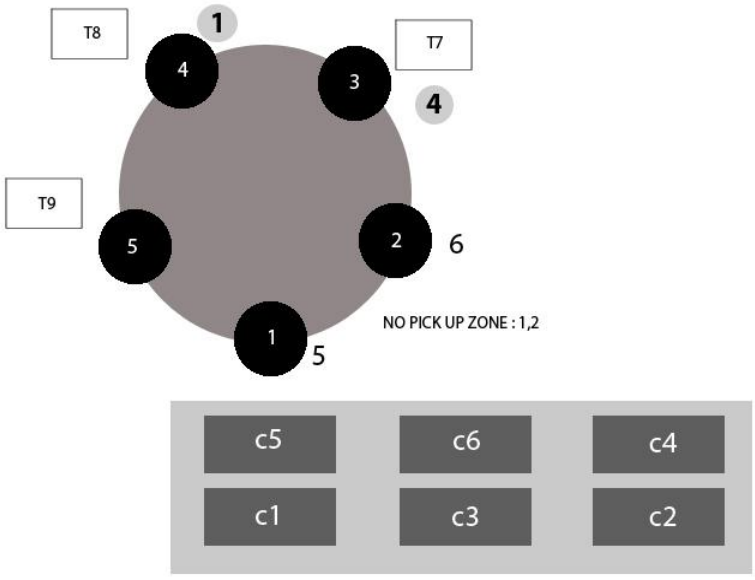

Figure III. 2 The schematic diagram of assembly step 1

This situation is shown in Figure III. 2. Here the turret rotation time is tt2 because, there are group 2 components in feeders.

Step 2: As seen in Figure III. 3, component 6 is in feeder 1 which means it is picked up during one of previous placements. Same as component 6, component 4 and 1 is grabbed in first step. And the component 6 is mounted to the board. Here turret rotation time is tt1 because all the components are group 1 components.

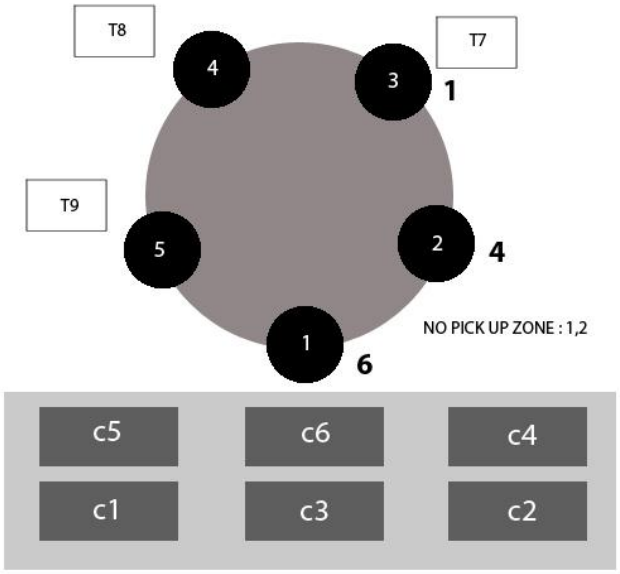

Figure III. 3 The schematic diagram of assembly step 2

Step 3: Figure III. 4 represents the placement step 3; components 4 and 1 are grabbed previously. 3, 2 and 5 is grabbed just now. Here there is group 2 component which is c5. For that reason the turret rotation time increases to tt2 again.

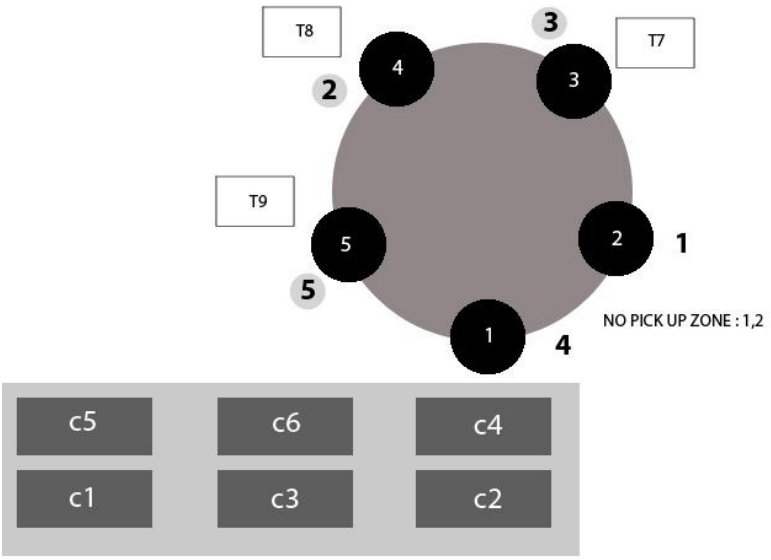

Figure III. 4 The schematic diagram of assembly step 3

Step 4: In that step, component 1 is mounted to the board. Still the feeders has group 2 component (c5) so the rotation time is still tt_2. As seen in Figure III. 5, no components are grabbed because the types under feeders will not mounted to the board. Figure III. 5 represents that step.

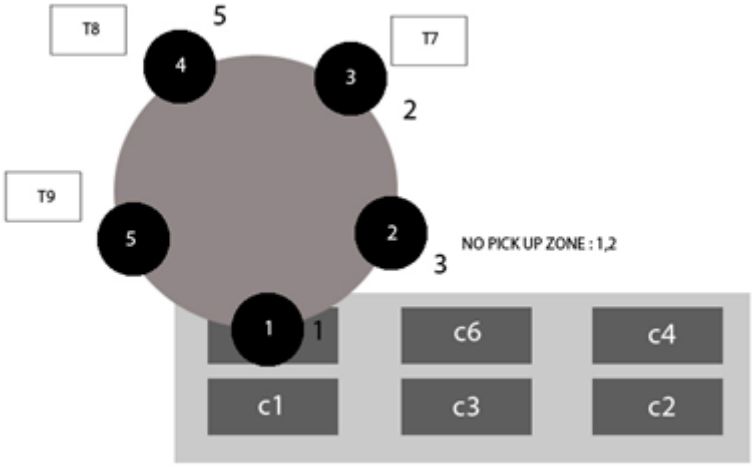

Figure III. 5 The schematic diagram of assembly step 4

Step 5: As shown in Figure III. 6, component 3 is mounted to the board and no other components are grabbed. Turret contains component 5 which is group 2 element so the turret rotation time is still tt_2.

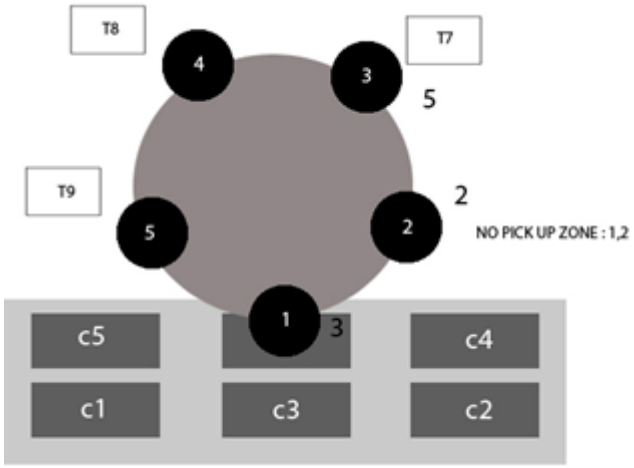

Figure III. 6 The schematic diagram of assembly step 5

Step 6: Component 2 is mounted to the board and feeder start to grabbed component 6 for next board. This means that travelling along the path restarts after a previous tour ends. Note that this set of placements restart from one after sixth placement. Turret rotation time is tt2. After the mount operation of component 2, component 5 will be mounted for next board.

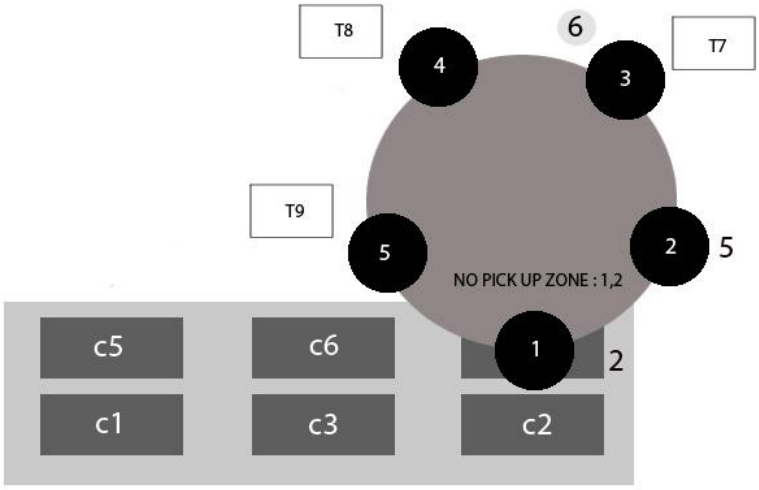

Figure III. 7 The schematic diagram of assembly step 6

When we compare that step with step 1, we can notice that in step 1 components 5 and 6 are taken from previous board assembly.

The total rotation time is

$C_{251}=max(0.40, max(0.4, 0.2, 0.2, 0.2, 0.0)) = 0.4$ s

C_{562}=*max*(0.20, *max*(0.2, 0.2, 0.2, 0.0, 0.0)) = 0.2 s

C_{643}=*max*(0.25, *max*(0.2, 0.2, 0.2, 0.2, 0.4)) = 0.4 s

C_{414}=*max*(0.20, *max*(0.2, 0.2, 0.2, 0.4, 0.0)) = 0.4 s

C_{135}=*max*(0.40, *max*(0.2, 0.2, 0.4, 0.0, 0.0)) = 0.4 s

C_{326}=*max*(0.25, *max*(0.2, 0.4, 0.2, 0.0, 0.0)) = 0.4 s and total cost is 2.2 s

From the steps above, it can be noticed that grabbing the component, mounting it and X-Y movement of the board is concurrent.

III.3 FORMULATION REPRESENTATION OF FORMULATION

In order to see the formulation, the above example can be used.

The formulation of problem is:

$$\min \sum_{i=1}^{N} \sum_{\substack{j=1 \\ j \neq i}}^{N} \sum_{p=1}^{N} D_{ijp}$$

Subject to:

For equation (III. 19)) and (III. 14);

xip_1(c1).. x(c1,p1) + x(c1,p2) + x(c1,p3) + x(c1,p4) + x(c1,p5) + x(c1,p6) = 1

xip_1(c2).. x(c2,p1) + x(c2,p2) + x(c2,p3) + x(c2,p4) + x(c2,p5) + x(c2,p6) = 1

xip_1(c3).. x(c3,p1) + x(c3,p2) + x(c3,p3) + x(c3,p4) + x(c3,p5) + x(c3,p6) = 1

xip_1(c4).. x(c4,p1) + x(c4,p2) + x(c4,p3) + x(c4,p4) + x(c4,p5) + x(c4,p6) = 1

xip_1(c5).. x(c5,p1) + x(c5,p2) + x(c5,p3) + x(c5,p4) + x(c5,p5) + x(c5,p6) = 1

xip_1(c6).. x(c6,p1) + x(c6,p2) + x(c6,p3) + x(c6,p4) + x(c6,p5) + x(c6,p6) = 1 (9)

xip_2(p1).. x(c1,p1) + x(c2,p1) + x(c3,p1) + x(c4,p1) + x(c5,p1) + x(c6,p1) = 1

xip_2(p2).. x(c1,p2) + x(c2,p2) + x(c3,p2) + x(c4,p2) + x(c5,p2) + x(c6,p2) = 1

xip_2(p3).. x(c1,p3) + x(c2,p3) + x(c3,p3) + x(c4,p3) + x(c5,p3) + x(c6,p3) = 1

xip_2(p4).. x(c1,p4) + x(c2,p4) + x(c3,p4) + x(c4,p4) + x(c5,p4) + x(c6,p4) = 1

xip_2(p5).. x(c1,p5) + x(c2,p5) + x(c3,p5) + x(c4,p5) + x(c5,p5) + x(c6,p5) = 1

xip_2(p6).. x(c1,p6) + x(c2,p6) + x(c3,p6) + x(c4,p6) + x(c5,p6) + x(c6,p6) = 1 (10)

For equation (III. 20) and (III. 16);

w(c2,c1,p1) + w(c3,c1,p1) + w(c4,c1,p1) + w(c5,c1,p1)+ w(c6,c1,p1) = x(c1,p1)

w(c2,c1,p2) + w(c3,c1,p2) + w(c4,c1,p2) + w(c5,c1,p2)+ w(c6,c1,p2) = x(c1,p2)

w(c2,c1,p3) + w(c3,c1,p3) + w(c4,c1,p3) + w(c5,c1,p3)+ w(c6,c1,p3) = x(c1,p3)

w(c2,c1,p4) + w(c3,c1,p4) + w(c4,c1,p4) + w(c5,c1,p4)+ w(c6,c1,p4) = x(c1,p4)

w(c2,c1,p5) + w(c3,c1,p5) + w(c4,c1,p5) + w(c5,c1,p5)+ w(c6,c1,p5)= x(c1,p5)

w(c2,c1,p6) + w(c3,c1,p6) + w(c4,c1,p6) + w(c5,c1,p6)+ w(c6,c1,p6) = x(c1,p6)

w(c1,c2,p1) + w(c3,c2,p1) + w(c4,c2,p1) + w(c5,c2,p1)+ w(c6,c2,p1) = x(c2,p1)

w(c1,c2,p2) + w(c3,c2,p2) + w(c4,c2,p2) + w(c5,c2,p2)+ w(c6,c2,p2) = x(c2,p2)

w(c1,c2,p3) + w(c3,c2,p3) + w(c4,c2,p3) + w(c5,c2,p3)+ w(c6,c2,p3) = x(c2,p3)

w(c1,c2,p4) + w(c3,c2,p4) + w(c4,c2,p4) + w(c5,c2,p4)+ w(c6,c2,p4) = x(c2,p4)

w(c1,c2,p5) + w(c3,c2,p5) + w(c4,c2,p5) + w(c5,c2,p5)+ w(c6,c2,p5) = x(c2,p5)(11)

REMAINING 25 ENTRIES SKIPPED

w(c1,c2,p1) + w(c1,c3,p1) + w(c1,c4,p1) + w(c1,c5,p1)+ w(c1,c6,p1) = x(c1,p6)

w(c1,c2,p2) + w(c1,c3,p2) + w(c1,c4,p2) + w(c1,c5,p2)+ w(c1,c6,p2) = x(c1,p1)

w(c1,c2,p3) + w(c1,c3,p3) + w(c1,c4,p3) + w(c1,c5,p3)+ w(c1,c6,p3) = x(c1,p2)

w(c1,c2,p4) + w(c1,c3,p4) + w(c1,c4,p4) + w(c1,c5,p4)+ w(c1,c6,p4) = x(c1,p3)

w(c1,c2,p5) + w(c1,c3,p5) + w(c1,c4,p5) + w(c1,c5,p5)+ w(c1,c6,p5) = x(c1,p4)

w(c1,c2,p6) + w(c1,c3,p6) + w(c1,c4,p6) + w(c1,c5,p6)+ w(c1,c6,p6) = x(c1,p5)

w(c2,c1,p1) + w(c2,c3,p1) + w(c2,c4,p1) + w(c2,c5,p1)+ w(c2,c6,p1) = x(c2,p6)

w(c2,c1,p2) + w(c2,c3,p2) + w(c2,c4,p2) + w(c2,c5,p2)+ w(c2,c6,p2) = x(c2,p1)

w(c2,c1,p3) + w(c2,c3,p3) + w(c2,c4,p3) + w(c2,c5,p3)+ w(c2,c6,p3) = x(c2,p2)

w(c2,c1,p4)+w(c2,c3,p4)+w(c2,c4,p4)+w(c2,c5,p4)+w(c2,c6,p4)=x(c2,p3)

w(c2,c1,p5) + w(c2,c3,p5) + w(c2,c4,p5) + w(c2,c5,p5)+ w(c2,c6,p5) = x(c2,p4) (12)

REMAINING 25 ENTRIES SKIPPED

For equation (III. 21);

dijp_wijp(c1,c2,p1).. - 10*w(c1,c2,p1) + d(c1,c2,p1) ≥ 0

dijp_wijp(c1,c2,p2).. - 10*w(c1,c2,p2) + d(c1,c2,p2) ≥ 0

dijp_wijp(c1,c2,p3).. - 10*w(c1,c2,p3) + d(c1,c2,p3) ≥ 0

dijp_wijp(c1,c2,p4).. - 10*w(c1,c2,p4) + d(c1,c2,p4) ≥ 0

dijp_wijp(c1,c2,p5).. - 10*w(c1,c2,p5) + d(c1,c2,p5) ≥0

dijp_wijp(c1,c2,p6).. - 10*w(c1,c2,p6) + d(c1,c2,p6) ≥ 0

dijp_wijp(c1,c3,p1).. - 0.4*w(c1,c3,p1) + d(c1,c3,p1) ≥ 0

dijp_wijp(c1,c3,p2).. - 0.4*w(c1,c3,p2) + d(c1,c3,p2) ≥ 0

dijp_wijp(c1,c3,p3).. - 0.4*w(c1,c3,p3) + d(c1,c3,p3) ≥ 0

dijp_wijp(c1,c3,p4).. - 0.4*w(c1,c3,p4) + d(c1,c3,p4) ≥ 0

dijp_wijp(c1,c3,p5).. - 0.4*w(c1,c3,p5) + d(c1,c3,p5) ≥ 0

dijp_wijp(c1,c3,p6).. - 0.4*w(c1,c3,p6) + d(c1,c3,p6) ≥ 0

dijp_wijp(c1,c4,p1).. - 10*w(c1,c4,p1) + d(c1,c4,p1) ≥ 0

dijp_wijp(c1,c4,p2).. - 10*w(c1,c4,p2) + d(c1,c4,p2) ≥ 0

dijp_wijp(c1,c4,p3).. - 10*w(c1,c4,p3) + d(c1,c4,p3) ≥ 0

dijp_wijp(c1,c4,p4).. - 10*w(c1,c4,p4) + d(c1,c4,p4) ≥ 0

dijp_wijp(c1,c4,p5).. - 10*w(c1,c4,p5) + d(c1,c4,p5) ≥ 0

dijp_wijp(c1,c4,p6).. - 10*w(c1,c4,p6) + d(c1,c4,p6) ≥ 0

dijp_wijp(c1,c5,p1).. - 10*w(c1,c5,p1) + d(c1,c5,p1) ≥ 0

dijp_wijp(c1,c5,p2).. - 10*w(c1,c5,p2) + d(c1,c5,p2) ≥ 0

dijp_wijp(c1,c5,p3).. - 10*w(c1,c5,p3) + d(c1,c5,p3) ≥ 0

dijp_wijp(c1,c5,p4).. - 10*w(c1,c5,p4) + d(c1,c5,p4) ≥ 0

dijp_wijp(c1,c5,p5).. - 10*w(c1,c5,p5) + d(c1,c5,p5) ≥ 0

dijp_wijp(c1,c5,p6).. - 10*w(c1,c5,p6) + d(c1,c5,p6) ≥ 0

dijp_wijp(c1,c6,p1).. - 10*w(c1,c6,p1) + d(c1,c6,p1) ≥ 0(5)

.

.

REMAINING 155 ENTRIES SKIPPED

For each feeder, constraint sets (III. 10) to (III. 12) (representing R different constraint sets) impose that placement time for component j cannot be smaller than the turret time values associated with the incoming components for placement, if they are already picked up by the heads.

For each feeder, we have 180 equation constraints.

equ1(c1,c2,p1) ..(0)*w(c1,c2,p1) + d(c1,c2,p1) + (0)*x(c1,p1) + (0)*x(c2,p1) + (0)*x(c3,p1) + (0)*x(c4,p1) + (0)*x(c5,p1) ≥ 0

equ2(c1,c2,p1)..(0)*w(c1,c2,p1) + d(c1,c2,p1) + (0)*x(c1,p2) + (0)*x(c2,p2) + (0)*x(c3,p2) + (0)*x(c4,p2) + (0)*x(c5,p2) ≥ 0

equ3(c1,c2,p1)..(0)*w(c1,c2,p1) + d(c1,c2,p1) + (0)*x(c1,p3) + (0)*x(c2,p3) + (0)*x(c3,p3) + (0)*x(c4,p3) + (0)*x(c5,p3) ≥ 0

equ4(c1,c2,p1)..(0)*w(c1,c2,p1) + d(c1,c2,p1) + (0)*x(c1,p4) + (0)*x(c2,p4) + (0)*x(c5,p4) ≥ 0

equ5(c1,c2,p1)..(0)*w(c1,c2,p1) + d(c1,c2,p1) + (0)*x(c5,p5) =G= 0 ; (LHS = 0)

REMAINING 895 ENTRIES SKIPPED

CHAPTER IV

RESULTS and DISCUSSION

In this section, we give a detailed analysis of exact methods on our problem. Moreover, the affects of component and type numbers are compared when solving the SDTSP.

Different size of problems are compared with two types of data. One of them is randomly created data and the other is uniform data. In uniform data, the solution of SDTSP is clear with a little amount of effort.

When comparing the PCB data, components, types and groups are increased orderly and the runtime, required memory complexity of them analyzed.

So we compared our model in sections,

- Complexity
- Runtime and
- Total memory requirement.

In that section, we give a solution and output of our model in GAMS and analyze the statistical data.

The statistical data is generated with DICOPT in GAMS platform. In

Table IV. 1, The NON LINEAR N-Z shows the number of nonlinear matrix entries in the model. With the help of that table we can analyze the problem size and its effects.

Additional to these values, GAMS platform gives “code length” to calculate the complexity of problem. In non-linear problems, the complexity of problem is not always same with the matrix entries to the model. For example, the complexity of , $x \times y$ is lower than the complexity of $exp(x \times y)$. But the both formulations’ matrix entries to the model is 1. For that reason, GAMS platform, results “code length” in order to define the complexity better. In our statistical data, this compexity is defined with “Level of complexity on the non-linearity”.

Optimal solution of example 1 is given in Table IV.1:

Table IV. 1 Gams statistical outputs

MODEL STATISTICS			
BLOCKS OF EQUATIONS	11	SINGLE EQUATIONS	1,165
BLOCKS OF VARIABLES	4	SINGLE VARIABLES	397
NON ZERO ELEMENTS	6, 265	NON LINEAR N-Z	4,320
DISCRETE VARIABLES	216	CODE LENGTH	

```
                S O L V E      S U M M A R Y

     MODEL   sdtsp                OBJECTIVE  obj
     TYPE    MINLP                DIRECTION  MINIMIZE
     SOLVER  DICOPT               FROM LINE  225

**** SOLVER STATUS     1 Normal Completion
**** MODEL STATUS      8 Integer Solution
**** OBJECTIVE VALUE                2.2000

 RESOURCE USAGE, LIMIT          0.812      2222.000
 ITERATION COUNT, LIMIT       493    2000000000
 EVALUATION ERRORS              0             0
```

Above solution shows us that, in 6 components and 5 feeder example, there are 11 equation blocks, 1165 single equations and 397 variables. And in 0.812 seconds, we reach the solution which is 2.2 with given feeder configuration. As known for that problem, 2.2 is not optimal solution because of the feeder configuration.

To get a better placement sequence, one idea can be changing the feeder configuration. In that manner, if we put the heavier components closer to the printed circuit board and mount them at first, better solutions can be reached. In Table IV. 2, the type of feeder slots changed and as a result a better, 1.85, result is found.

Table IV. 2 A Better feeder configuration for example 1

y_{tr}	1	2	3	4	5
7	0	0	0	0	1
8	0	0	0	1	0
9	0	0	1	0	0

In next section, the computational analysis of model is given according to size of problem.

IV.1 COMPUTATIONAL ANALYSIS

In this section, the computing complexity of the models are studied. The number of components, groups and feeders are listed and according to them the computational times are calculated and presented. Moreover, the required hardware components are listed.

IV.1.1 Computational complexity

The complexity of the model, thus the number of generated number of equations increases when the number of variables and constraints increase. In other meanings, when the number of any components or the number of feeder slots increase the problem becomes more difficult to solve and takes too much time. So, the computational complexity of the model depends on the complexity of the model. Moreover, the complexity and constraints of our model depends on the number of feeder primarily. If the feeder number increases than the equations (III. 22) to (III. 12) increase proportionally. Thus the size of model increases.

Table IV. 3 shows us the complexity and number of equations when the numbers of components are increased.

Table IV. 3 Model statistics according to the component number (Feeder Number is 5, npz=2)

Component -Feeder Number	8	12	16	20	24	35	44	50
Single Equation	2,833	9,817	23,585	46,441	80,689	252,421	503,449	740,101
Non Linear N-Z	16,128	76,032	261,120	638,400	1,324,800	5,997,600	14,984,640	24,990,000
Single Variable	961	3,313	7,937	15,601	27,073	84,526	168,433	740,101
Discrete Variable	512	1,728	4,096	8,000	13,824	42,875	85,184	125,000
Level of Complexity on the Non Linearity	101,249	472,033	1,605,121	3,906,401	8,081,281	36,318,801	90,573,825	150,920,001

Level of complexity on the non-linearity graph is illustrated in Figure IV. 1 From the figure the exponential increase according to component number is seen clearly.

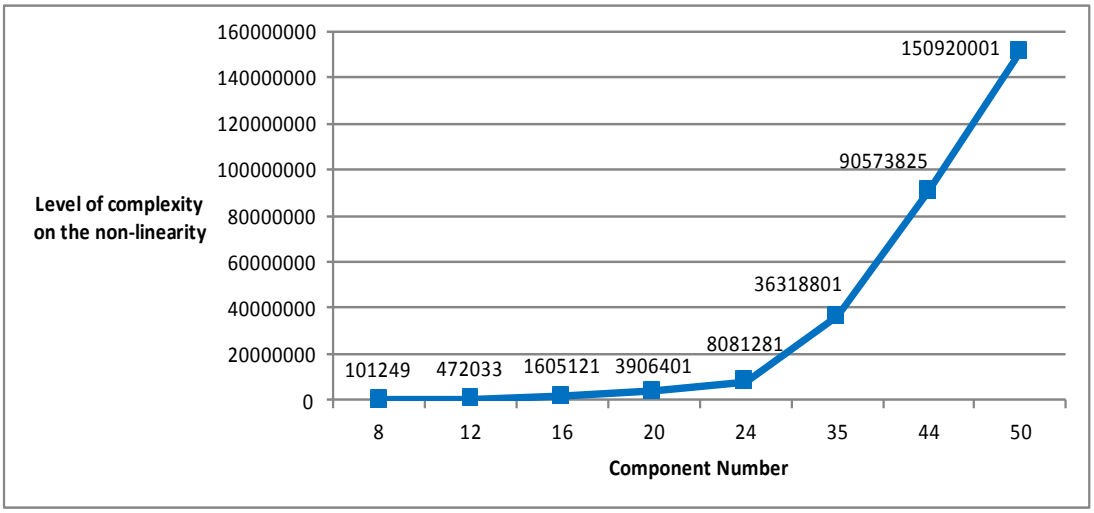

Figure IV. 1 The complexity level when the number of components between 8-50

As shown in the Table IV. 3, the complexity and the number of equations is getting higher when the number of components change. This results also more consumption on memory and hardware. Table IV. 4 shows the memory consumption of model according to component number. When the component number is 50, the required memory can be up to 16GB.

Table IV. 4 Total number of required RAM according to component number (memory sizes are approximate)

Component Number	15	20	25	35	40
RAM Consumption (mb)	700	1000	1600	3200	4300

In next test, the number of feeder is increased and analyzed how it affects the nonlinearity of the model. When this compared with the increase in component number it can be seen that the slope of the increase in feeder number is less than the increase in component number. Table IV. 5 and Figure IV. 2 illustrate the effect of feeder slot number to the model.

Table IV. 5 Model statistics according to the feeder number

Component -Feeder Number	20-5	20-6	20-7	20-8	20-9	20-10	20-16
Single Equation	46,441	54,041	61,641	69,241	76,841	84,441	130,041
NON LINEAR N-Z	638,400	782,800	813,200	851,200	881,600	934,800	1,368,000
Single Variable	15,601	15,601	15,601	15,601	15,601	15,601	15,601
Discrete Variable	8,000	8,000	8,000	8,000	8,000	8,000	8,000
Level of complexity on the non-linearity	3,906,401	4,788,000	4,985,601	5,228,801	5,426,401	5,760,801	8,451,201

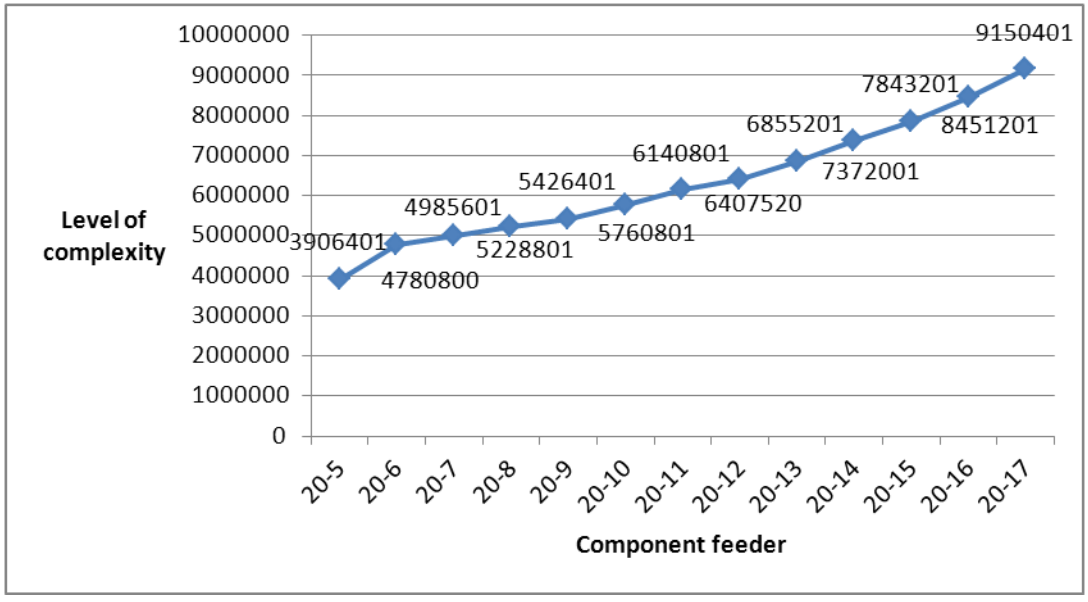

Figure IV. 2 The complexity level when the number of components between 20-5 to 20-17 component feeder

When the number of components and feeders increases independently, as seen from Table IV. 3 and Table IV. 5, there is an exponential time and complexity grow in model statistics. And in that step, both feeder and component numbers increased simultaneously. The relationship between feeder and component number is shown in Table IV. 6.

Table IV. 6 Model statistics according to increase in component and feeder number

Component Number-Feeder Slots Number	10-5	12-6	14-7	15-8	16-10	17-12	19-14	20-16
Single Equation	5,621	11,401	20,805	28,831	42,785	60,725	98,231	130,041
NON LINEAR N-Z	37,800	106,128	20,129	289,800	414,720	591,872	831,744	1,368,000
Single Variable	1,901	3,313	5,293	6,526	7,937	9,538	13,358	15,601
Discrete Variable	1,000	1,728	2,744	3,375	4,096	4,913	6,859	8,000
Level of complexity on the non-linearity	235,801	655,777	1,289,289	1,789,201	2,565,121	3,662,209	5,146,417	8,451,201

And the Figure IV.3 shows us the increase in complexity that depends on both feeder and component number.

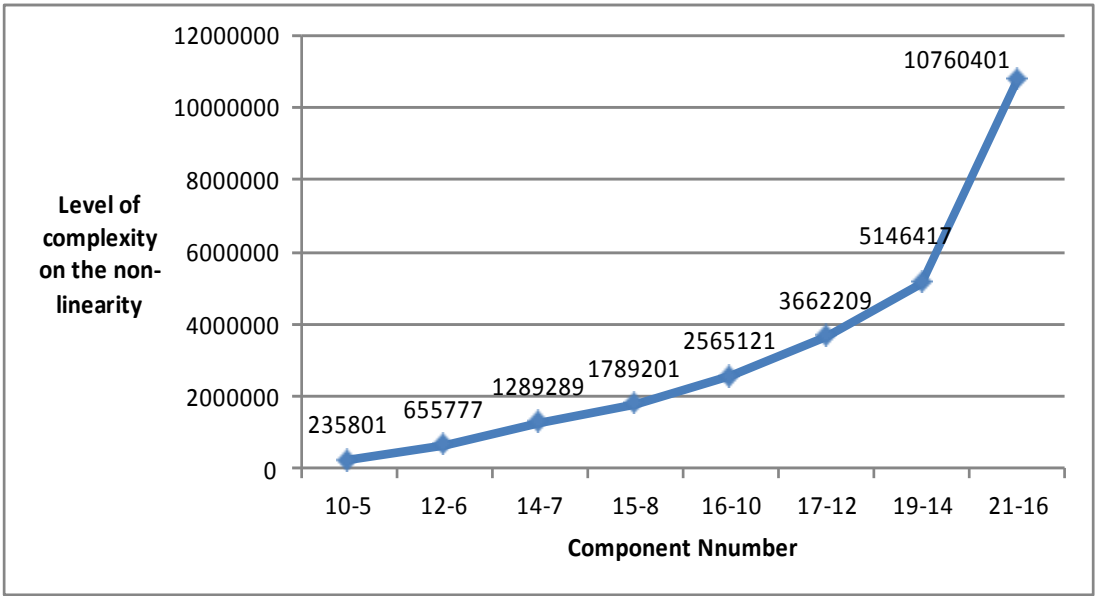

Figure IV.3 The complexity level when the number of components and feeders increase simultaneously

When Figure IV.3 is compared with Figure IV. 1 and Figure IV. 2, it can be proved that if the number of feeders and components increase simultaneously, higher grow in slope is derived .

IV.1.2 Computational Time

The above formulation is nonlinear integer programming and the discrete variables are binary variables which appear nonlinearly in the model. In that point solvers that are used must be capable of handling these both conditions.

To solve model, DICOPT is used as solver. Because of the structure of formulation, in GAMS, DICOPT is used as solver because it handles both nonlinear and integer programming. There are other solvers that solve our problem like BARON. But in our formulation, DICOPT is chosen as solver because the execution time and memory requirement is less than needed by BARON.

In problem, the other solver that is compared with DICOPT is BONMIN (Basic Open-source Nonlinear Mixed INteger programming). BONMIN is a MINLP solver that is developed with C++ programming language. In that solver, branch and bound, cutting planes and branch and cut algorithms are used. When we solve placement sequence problem with BONMIN, the required time increases considerably. Table IV. 7 shows the required times in both solvers.

Table IV. 7 Comparison of DICOPT and BONMIN solvers (Feeder configuration=5, npz=2)

Component Number	5	10	15
DICOPT Runtime (Seconds)	1.062	5.406	39.547
BONMIN Runtime(Seconds)	81.766	1,014.672	1,561.047

Although optimal solutions can be found with mathematical model, the time to solve a problem with more components, feeder slots, type and groups takes too much time. On the other hand, because of memory requirements in larger problems GAMS is unable to run the DICOPT solver. So, in our model, two drawbacks arise;

- Exponentially increase in computational time
- Exponentially increase in memory requirement.

When we change the number of components, the required time is changed as in Table IV. 8. And this time increases or decreases according to the diffuculty of the problem.

In GAMS, when the diffuculty of the problem is easy, the time required for solution is decreases. On the other hand, when the problems are hard, this time increases in big proportion.

Table IV. 8 shows the assembly time of printed circuit boards when the problem is easy. Easy problem can be defined as arranging the same groups and types in sequential order. In other words, the components of c_1, c_2, c_3, c_4, c_5, c_6 are designed consecutively in PCB and with just looking at the PCB, the tour can be easily defined.

Table IV. 8 According to the component number, the runtime of easy problems (Feeder configuration=5, npz=2)

Component Number	6	10	15	20	25	30
Feeder Configuration	5	5	5	5	5	5
Component Type Number	3	3	3	3	3	3
Runtime(second)	0.844	5.500	39.547	261.031	680.281	1590.844

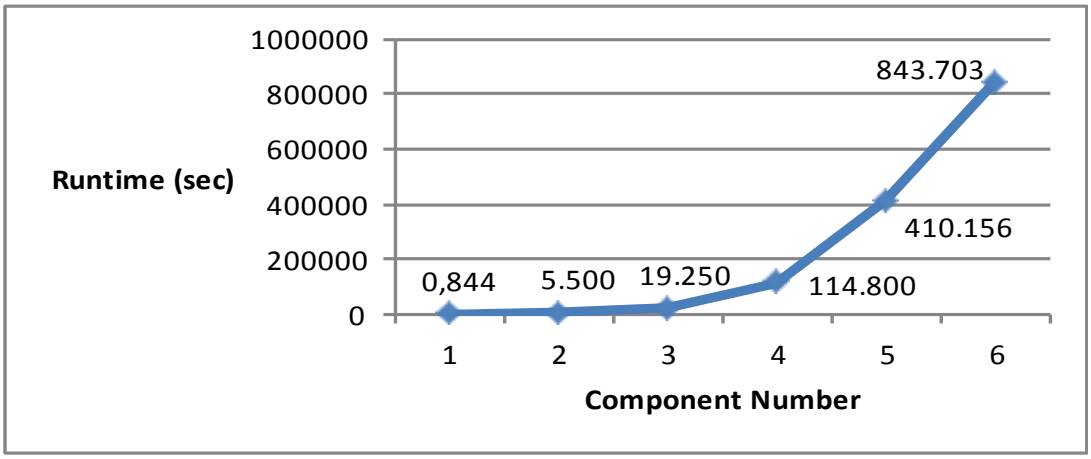

Figure IV. 4 Runtime of easy type problems according to the component number

When the coordinates of the components are defined randomly, than the problem becomes harder. Table IV. 9 shows the runtime of PCB assembly when we solve harder problems. While solving our problems, iteration number increases, thus the required time increases too.

Table IV. 9 Runtime of randomly generated problems and their optimal results (Feeder configuration=5, npz=2)

Number of Component	6	10	15	20	25	30
Feeder Configuration	5	5	5	5	5	5
Component Type Number	3	3	3	3	3	3
Runtime(second)	0.859	13	101.122	1,527.281	52,627.320	93,895.064
Optimal Solution	4.06	5.52	7.17	8.16	9.740	10.740

When the number of components increase up to 30, from Table IV. 9, we can see that the required time is 26 hours.

On the other hand, when the number of components are increased to the 46, then the required memory is insufficient to start the solver. So there is no solution after that size. Table IV.10, shows us the complexity level of the big size problems. And when that table is compared with small size problems in

Table IV. 3, the exponential increase can be seen clearly. This increase also affects the required memory size and time.

When we increase the number of components up to 50, GAMS cannot generate the input file and not solve the problem because of inadequate memory size. As seen from Table IV.10, 50 components and 5 feeder slots problem generates too much equation and has too much complexity. As a result of that, more memory is needed to solve the problem.

Table IV.10 Model statistics according to increase in component

Component number (5 feeder)	40	44	50 (not solve)
Single Equation	377,681	503,449	740,101
NON LINEAR N-Z	10,233,600	14,984,640	24,990,000
Single Variable	126,401	168,433	740,101
Level of complexity on the non-linearity	62,025,601	90,573,825	150,920,001

Although mathematical model obtain optimum solutions, it is not an efficient way to use this method because of exponentially increase in time as seen from Table IV. 9.

Table IV. 8, we solved easier problems. When the hardness of the problem increase than the elapsed time to solve a problem increases exponentially.

When more bigger test cases are compared, the slope of time grow can be seen better. Table IV. 9 the component number increased from 6 to 30 and the time increases exponentially in real data problems.

CHAPTER V

COMPUTATIONAL ANALYSES OF ABC ALGORITHM

In this chapter, extensive computational analyses of ABC algorithm are given. Firstly, performance of ABC algorithm is tested on the problem arising from the operations of chip shooter machines where the problem is solving QAP and SDTSP concurrently. The results are compared with SA and the findings are summarised. Then, performance of ABC algorithm is tested on the problems arising from the operations of chip mounter machines, where the problem is SDTSP. These results are very important because they reflect the performance of the ABC algorithm when compared with exact solutions. As the reader will see, the results are promising.

V.1 APPLICATION OF ABC ALGORITHM TO QAP AND SDTSP CONCURRENTLY

In this section, performance of ABC metaheuristic is analysed with respect to limit and cycle parameters. In that algorithm, the limit and cycle are user given parameters. If the number of cycles that a resource can not be improved for the value of limit parameter, it is considered to be exhausted and replaced with new solution. Cycle parameter is the stoping criteria of the algorithm. In this section, we also investigated the number of cycles that results in stable assembly time.

In this implementation of ABC for chip shooter machines, firstly, the foraging process of bees are initiated. In this very first step of algorithm, initial solutions are generated randomly. In the second step, employed bees are distributed to the solutions and generate new solutions. In our implementation, solution consists of two sub solutions; the feeder configuration and placement sequencing. A neighbour solution is created by pair-wise exchange in either feeder configuration or placement sequence or both. The proposed methods for generation of new solutions can be classified as equal chance, weighted chance and turn based chance.

In Equal chance generation of new solution, we used

If rand < 0.5

New neighbor of feeder

Else

New neighbor of placement sequence

in weighted chance the neighbor creation neighbors are created with,

If $r < \frac{S^2}{CS^2+S^2}$

New neighbor of feeder

Else

New neighbor of placement sequence

Where *S* is the cardinality of the set of types in placement sequence and *CS* is the cardinality of set of the components. In turn based solution creation, new placement sequence and feeder configurations are generated simultaneously.

In the next step, onlooker bees selects a solution with the help of information from employed bees. In employed and onlooker bee phase, the abandoned solutions are checked. Scout bees replace the abandoned solutions with new ones.

Onlooker bees are distributed to the solutions according to a probability. The probability is calculated via the quality (fitness) of solutions. To get better solutions, it is important to iterate over existing solutions, for that reason, in probability calculation the probability must be spread between small intervals. The calculation of probability values of each solution is (Karaboğa and Akay);

(IV.1)

$$P_i = 0.9 \times \frac{f_i}{f_{max}} + 0.1$$

In the above formulation, f_i is the fitness of current solution and f_{max} is the maximum fitness of solutions.

In this thesis, a new probability calculation method is proposed in selection step of solutions for onlooker bees. In optimizing chip shooter machines, many iterations are needed at the same solution in order to get better results. The number of bees that are sent to the solutions are not same. When they go same point, they give

more positive feedback. The new method narrows the interval of probability and sends the onlooker bees to better solutions more.

(IV.2)

$$P_{sel} = \text{random} (P_h, P_L)$$

Here, P_{sel} is selected probability, P_h, P_L are highest and lowest probability calculations from probability values of the solutions.

With the help of this method, better results are obtained because bees visit the same source more frequently.

In this section, firstly, the results of new distribution method is analyzed. Then the performance of ABC is proposed and compared with Simulated Annealing in chip shooter machines' placement operation.

For the bee colony algorithm, new distribution method is compared with the parameters as follows; the number of cycles is 5000, the limit for scout bees is 1000 and colony size is 20. Each of the experiments were repeated 10 times to see the robustness. The Figure V. 1 shows the change in PCB assembly time with new distribution of bees.

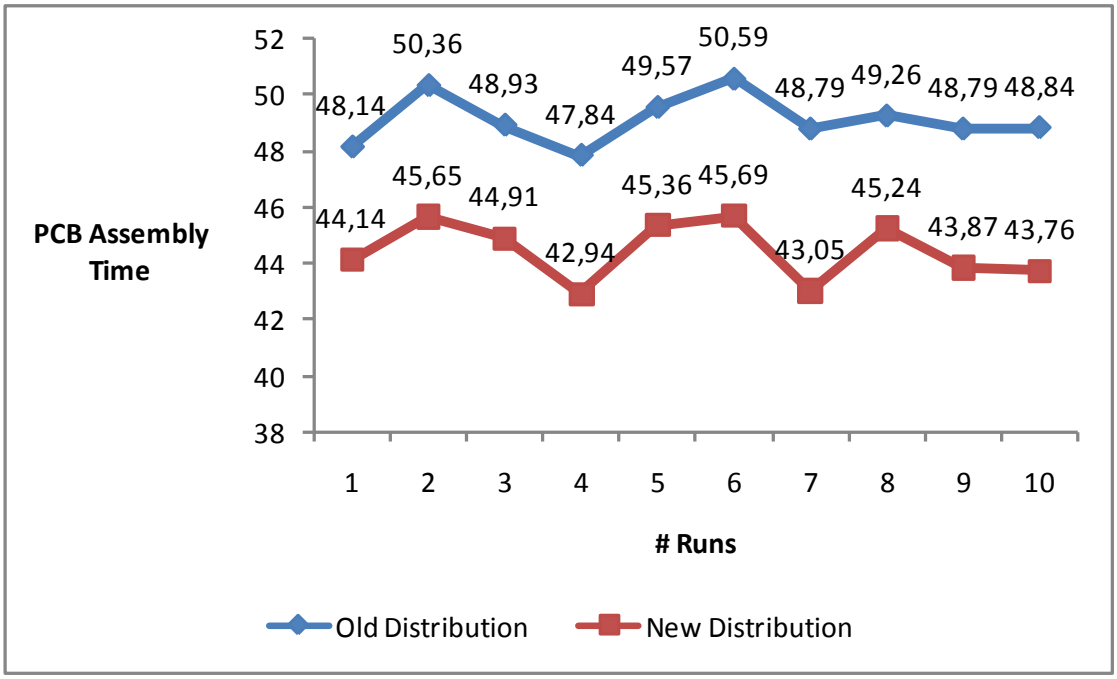

Figure V. 1 New Distribution method

In the originally proposed distribution method (old one) (Karaboğa and Akay), solution selection is done with randomly generated number between [0,1]. This

method causes to distribute onlooker bees into the all solutions. To collect the bees to the better solutions, new distribution method is proposed. As seen from Table V. 1; the mean of 10 runs of are 49.11 and 44.46, and the improvement is about 9.46%.

Table V. 1 Improvement with new distribution methodology

Mean of Solutions	Results	
	Running time	Improvement
49,11	18,16	-
44,46	17,91	9.46%

In Artificial Bee Colony algorithm, limit variable determines the abandoned solutions and replaces them with new ones with scout bees. If the limit variables are not big enough, the improvement of solutions can be left half finished and replaced with new solutions. When the limit variable is changed Table V. 2 is obtained. When the limit variable variable set to ∞, we obtain better results.

Table V. 2 Solutions with different limit variables on PS11AK08-9

Limit	Cycle	Colony Size	Results	
			Elapsed Time	Best Solution
200	5000	20	18,19	50,816
500	5000	20	17,91	47,92
1000	5000	20	17,448	45.28
∞	5000	20	17,123	44,87

The effects of iteration number on the performance of ABC algorithm is shown in Table V. 3.

Table V. 3 Change in number of iterations in ABC on PS11AK08-9

Cycle	Results	
	Time	Mean
1000	3,63	51,42
5000	17,91	44,46
10000	38,8	41,8

The other parameter that is used in Artificial Bee Colony algorithm is the size of bee colony. In our experiments, the tested colony sizes are 10, 50, 100 and 1000. Half of them are employed bees and the other half are onlooker bees (Karaboğa and Akay). Results shows that the colony size does not affect the results explicitly (Table V. 4). Therefore, colony size is not as important as the other parameters.

Table V. 4 Results with change of colony size on PS11AK08-9 with 10000 iterations and 200 limit with 10 runs

Colony size	Results
	Avg. Solution
10	50,31
20	50,62
50	50,54
100	50,03
1000	50,32

Table V. 5 Comparison with simulated annealing on problem instances

Problem Instances	Metaheuristics					
	ABC			SA		
	Avg (%)	Best (%)	Tbest (seconds)	Avg (%)	Best (%)	Tbest (seconds)
PS11AK08-9	41,8	40,4	35	38,57	36,70	7,74
PS11AK12-7	45,43	44,52	37	42,09	40,77	6,34
PS11AK15-4	44,67	43,63	39,95	45,05	43,75	3,25
PS11AK16-3	76,03	71,78	60,88	75,15	73,65	9,98
PS11AK16-4	75,32	73,38	84	74,36	72,15	11,16
PS11AK16-5	81,73	80,43	72	79,63	76,69	13,12
PS11AK17N3	54,2	49,25	4,1	49,35	47,33	13,20
PS11AK1011	43,49	42,45	42,45	42,52	40,78	6,34

To investigate the performance of simulated annealing and ABC, with given printed circuit board data in Table V. 5. We calculated average cost best found cost and time in seconds.

As can be seen from the results in Table V. 5, when the iteration count is 10000, artificial bee colony algorithm gives closer results to SA. However, when their running time are compared, SA gives better results. This is because of the fact that SA always improves the current solution and tries to find best solution from this current solution. On the other hand, in ABC algorithm, we have a user given solution number that is equal to the colony size and all these solutions are improved with the help of distrubuted bees. In other words, the iteration over same solution decreases.

In order to find the last point of the algorithm, we execute the ABC until the improvement stops to increase. Here, the limit variable is set to ∞ and the colony size is 20. In PS11AK08-9 PCB, the improvement of the algorithm can be seen in Figure V. 2.

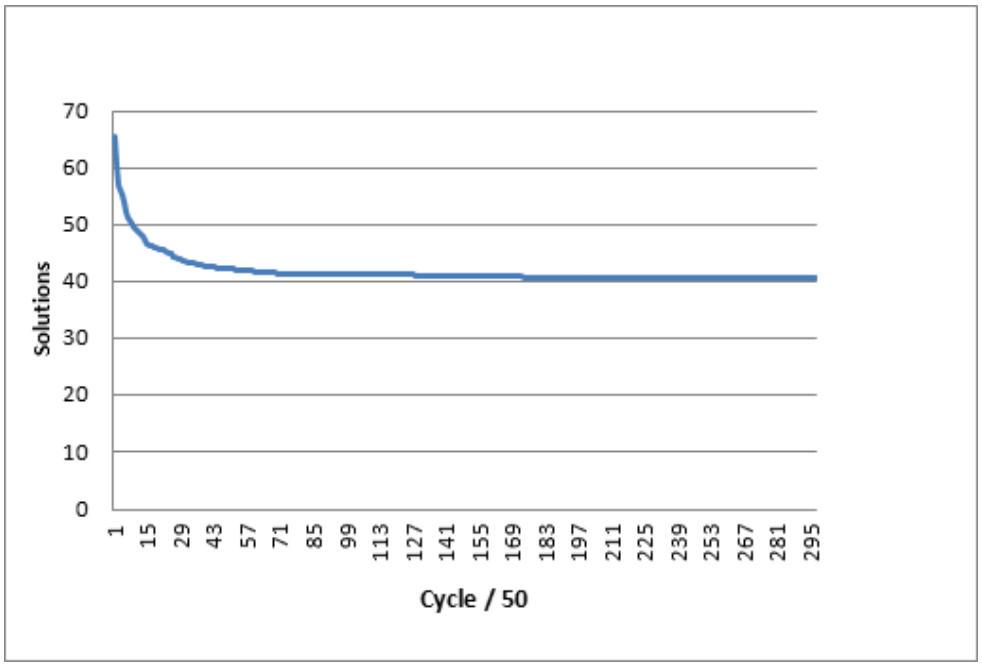

Figure V. 2 Improvement of ABC with iteration number

Figure V.2 shows the performance of algorithm in increasing number of cycle parameter. As seen from Figure V.2, the algorithm stops to improve its best solution up to 40,063 after 10000 cycles.

To get better solutions, it is important to find neighbors of the same solutions. In other words, the number of bees that are sent to solutions are changed with probability. As a result, evaluation rate over same solution decreases compared with simulating annealing. The number of iterations over same solution is changed with

the number of limit variable and probability variable of solutions. To get a better distribution of bees over the same solution space, a new methodology is proposed.

The results show that, the limit parameter which is used by scout bees affects the results of chip shooter machines. Replacing the current solution with new ones earlier, causes not to improve that solution, or not to send onlooker and employed bees. For that reason, when the limit parameters get higher, results are getting better. All the experiments were run on Windows operating system with core2 duo processor and 4 GB ram.

V.2 PERFORMANCE COMPARISON OF ABC AND SA WITH RESPECT TO EXACT SOLUTIONS ON SDTSP INSTANCES

In this section, we compared the performances of exact methods with ABC and SA. To see the performance of implemented metaheuristics , we solve 10, 15, 20, 25, 30 component size problems with the GAMS platform, ABC and SA algorithm. In 30 size problem, the exact value is 10,74 which is the taken from GAMS. With ABC algorithm, we get 12.64 with 3000 cycles, ∞ limit and 20 colony size. On the other hand, the minimum value we get with SA is 13,10 when the initial temperature is 100, number of iterations at each temperature setting is 20 and temperature decrease ratio is 1,5 (In Table V. 6).

When we test our problem with 30 components, the proximity of the ABC to the exact value is 19,55%, on the other hand, with 15 size problem this proximity decreases to 10,5 %.

Table V. 6 Comparison with exact methods and metaheuristics

# Components	10	15	20	25	30
Exact Solution	5,52	7,17	8,16	9,74	10,74
ABC	5,98	7,93	9,33	11,06	12,75
SA	5,98	7,93	9,33	11,06	13,1

When we compare them in time domain, we get the table below. When the number of components increase up to 30, from Table V.7, it can be seen that the

required time is 26 hours. On the other hand, the required time is about 1,49 secons with SA and 13,91 seconds with ABC algorithm.

Table V. 7 Computational time with Exact Methods and Metaheuristics

# Components	10	15	20	25	30
GAMS(Runtime)	13	101,122	1527,281	52627,32	93895,064
SA(Runtime)	0,018	0,06	0,2	0,38	1,49
ABC(Runtime)	2,06	8,05	9,75	11,7	13,91

We can say that ABC and SA give reasonable results in very small amounts of run time when compared with exact solutions and its running time. Thus, it is shown that ABC is a promising algorithm for solving combinatorial optimization problems.

CHAPTER VI

CONCLUDING REMARKS and RECOMMENDATIONS

In almost every electronic device, printed circuit boards are used. This usage of PCB, attracts the researchers because of the NP-Completeness inherent to the problems they yield. In this thesis, in order to optimize the production of PCBs some methodologies are developed and implemented. In general, PCBs have lots of components on it. For that reason, the production time increases dramatically, and that affects the cost of production.

In this thesis study, a new generalization of TSP, SDTSP is implemented with GAMS platform and computational analyse are reported. Afterwards, artificial bee colony algorithm is implemented for optimizing the operations of chip shooter machines and results are analysed and reported.

The tests are performed on a machine with Intel Xeon CPU at 2.0 GHz with 4 GB RAM using Windows 2003 Server Operating system.

For the chip mounter machine, placement sequence problem is formulated as mathematical model and solutions are analyzed using GAMS statistical data. Moreover, computational complexity and required hardware is reported.

The significant contribution of this thesis is;

1. With GAMS, optimal solutions are found and it was proved that the computational complexity and time grows exponentially when the number of components of any type and feeder slots increase. When the number of components are not so high, with exact methods, we reach the optimal solution in reasonable time. But when the number of component is too high, there is a dramatical increase in complexity or, nonlinear matrix entries of problem.
2. Although the optimal solution is found in small size models with GAMS, the required time and memory requirements to solve a real problem become too much. When we increase the number of components up to 30, the required time is about 26 hours. This dramatically increase in time prevents us to use

exact methods for solving large SDTSP instances. In other words, exact methods guarantee to find optimal solution of problems, but when the size of problem increases, they become meaningless to solve with.

3. Artificial bee colony algorithm is implemented for chip shooter machines. ABC algorithm tries to find a good solutions with the help of movements of honey bees. And the result of ABC algorithm shows that in short time better results can be found by heuristic methods.

REFERENCES

[1] Akrotirianakis I.; Maros I.; Rustem, B.: "An outer approximation based branch and cut algorithm for convex 0-1 MINLP problems", Optimization Methods and Software,*16(1-4)* **(2001)** 21-47

[2] Albiach, J.; Sanchis, J. M.; Soler, D.: "An asymmetric TSP with time windows and with time-dependent travel times and costs: An exact solution through a graph transformation", *European Journal of Operational Research*, 189 **(2008)** 789–802.

[3] Alkaya, A.F.; Duman, E.: "A Literature Survey of the Operation Optimization in Chip Shooter Placement Machines", *Proceedings of PICMET'09*, Portland, OR, USA, August 2-6 **(2009)** 3296-3306.

[4] Alkaya, A.F.: "Optimizing The Operations Of Electronic Component Placement Machines*", Doctoral Thesis*, Marmara University Institute for Graduate Studies in Pure and Applied Sciences, İstanbul, Türkiye **(2009)** 77-79.

[5] Alkaya, A.F.; Duman, E.: "Programming and optimizing the operations of a placement machine" **(2009).**

[6] Alkaya, A.F.; Duman, E.; Eyler, M.A: "Assembly time minimization for an electronic component placement machine", *WSEAS Transactions on Computers*, 7 **(2008)** 326-340.

[7] Applegate, D.; Bixby, R.; Chvátal, V.; Cook, W.: "Finding tours in the TSP" *Forschungs Institut für Diskrete Mathematik Report*, 99885, Computational and Applied Mathematics **(1999)**

[8] Applegate, D.L., Bixby, R.E., Chvatal, V., Cook, W.J., "The Traveling Salesman Problem: A Computational Study", *Princeton Series in Applied Mathematics*, **(2006)**.

[9] Babin, G.; Deneault, S.; Laporte, G.: "Improvements to the Or-opt heuristic for the symmetric travelling salesman problem", *Journal of the Operational Research Society*, 58 **(2007)** 402-407.

[10] Balas, E.; Ceria, S.;Cornuejols G.; Natraj, N.: "Gomory Cuts revisited", *Operations Research Letters*, 19 **(1996)** 1-9.

[11] Balas, E.; Ceria, S.; Cornuejols, G.: "Mixed 0-1 programming by lift-and-project in a branch-and-cut framework", *Management Science*, 42 **(1996b)** 1129-1246.

[12] Bigras, L.-P.; Gamache, M.; Savard, G.: "The Time-Dependent Traveling Salesman Problem and Single Machine Scheduling Problems with Sequence Dependent Setup Times", *Discrete Optimization*, 5 **(2008)** 685-699.

[13] Borchersa B.; John E.: "A computational comparison of branch and bound and outer approximation algorithms for 0–1 mixed integer nonlinear programs",*Computer and Operations Research*, 24(8) **(1997)** 699–701

[14] Boyd, E.A.: "Fenchel Cutting Planes for Integer Programs", *Operations Research*, 42 **(1994)** 53-64.

[15] Clausen, J.: "Branch and Bound Algorithms - Principles and Examples", *Department of Computer Science, University of Copenhagen, Universitetsparken* **(1999)** 2-26.

[16] Croes, G.: "A Method for Solving Traveling-Salesman Problems", *Operations Research*, 6 **(1958)** 791-812.

[17] Dantzig, G.; Fulkerson, R.; Johnson, S.: "Solution of a Large Scale Traveling Salesman problem", *Operations Research 2* **(1954)** 393-410.

[18] Dantzig, G.B.; Ramser J.H.: "The truck dispatching problem", *Management Science*, 6 **(1959)** 80-91.

[19] Drud A.: "A GRG Code for Large Sparse Dynamic Nonlinear Optimization Problems", *Mathematical Programming* 31 **(1985)** 153-191.

[20] Duman E.: "Modelling the operations of a component placement machine with rotational turret and stationary component magazine", *Journal of the Operational Research Society*, 58**(2007)** 317-325.

[21] Duman, E.; Or, I.: "The quadratic assignment problem in the context of the printed circuit board assembly process", *Computers and Operations Research*, 34 **(2007)** 163-179.

[22] Dumont, J.; Robichaud V.: “Introduction to GAMS Software A Manual for CGE Modelers” **(2000).**

[23] Duran. M.A.; Grossman I.E.: “An outer approximation algorithm for a class of mixed-integer nonlinear programs”, *Mathematical Programming, 36* **(1986)** 307–339.

[24] Fletcher R.; Leyffer S.:“Solving mixed integer nonlinear programs by outer approximation”, *Mathematical Programming, 66* **(1996)** 327-349

[25] Fox, K.; Gavish, B.; Graves, S.C.: “An n-Constraint Formulation of the (Time Dependent) Traveling Salesman Problem”, *Operations Research,* 28 **(1980)** 1019–1021.

[26] Gary, M.R.; Johnson, D.S.: "Computers and Intractability: A Guide to the Theory of NP-Completeness", *W.H. Freeman and Company*, **(1979)**.

[27] Gendrau, M.; Laporte, G.; Musaraganyi, C.; Taillard, E.D.: “A tabu search heuristic for the heterogeneous fleet vehicle routing problem”, *Computers and Operations Research,* 26 **(1999)** 1153-1173.

[28] Geoffrion A.M.: “Generalized benders decomposition”,*JOTA*, 10(4) **(1972)** 237-260.

[29] Golden, B.L.; Assad, A.A.; Levy L.; Gheysens, F.G.: “The Fleet Size and Mix Vehicle Routing Problem”, *Computers and Operations Research*, 11 **(1984)** 49-66.

[30] Gomory, R.E., "Outline of an algorithm for integer solutions to linear programs", *Bulletin of the American Mathematical Society*, 64 **(1958)** 275-278.

[31] Gomory, R.E.: "An algorithm for integer solutions to linear programs", *Recent Advances in Mathematical Programming*, R.L. Graves, P.Wolfe eds. McGraw-Hill, New York, **(1963)** 269-302.

[32] Grötschel, M.; Holland, O.: "Solution of large-scale traveling salesman problems", *Mathematical Programming*, 51(2) **(1991)** 141-202.

[33] Gutin, G.; Punnen, A.: "The Traveling Salesman Problem and its Variants", *Kluwer Academic Publishers*, **(2002)** 169-207.

[34] Haghani, A.; Jung, S.: "A dynamic vehicle routing problem with time-dependent travel times", *Computers and Operations Research*, 32**(2005)** 2959–2986.

[35] Hansen K.; Kraup J.: "Improvement of the Held-Karp algorithm for the symetric traveling salesman problem", *Mathematical Programming* 7 **(1982)** 87-96.

[36] Held, M.; Karp, R.M.: "The Traveling Salesman problem and minimum spanning trees", *Operations Research,* 18 **(1970)** 1138-1162.

[37] Ho, W.: "Component Sequencing and Feeder Arrangement for PCB Assembly Machines: Integration, models, solutions", *Doctoral Thesis,* The Hong Kong Polytechnic University **(2004)** 69-162

[38] Ichoua, S.; Gendreau, M.; Potvin J.Y.: "Vehicle dispatching with time-dependent travel times", *European Journal of Operational Research,* 144 **(2003)** 379–396.

[39] John E. M.: "Branch and cut algorithms for Combinatorial Optimization Problems", *Oxford University Press,* **(2002)** 65-77.

[40] Jon L.; Raffensperger, J.F.: "Using AMPL for teaching the TSP",*INFORMS Transactions on Education,* 7 **(2006)** 37-69.

[41] Jünger M.; Reinelt G.; Thiene S.: "Provably Good Solutions for the Traveling Salesman Problem", Preprint 94-31, IWR Heidelberg, **(1994)**

[42] Jünger, M.; Reinelt, G.; Thienel, S.: "Practical problem solving with cutting plane algorithms in combinatorial optimization", *Combinatorial Optimization: DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, AMS, **(1995)** 111-152.

[43] Karaboğa, D.; Akay, B.: "Artificial Bee Colony(ABC) Algorithm on Neural Networks" **(2009).**

[44] Kocis G. R.; Grossmann I.E.: "Relaxation Strategy for the Structural Optimization of Process Flow-sheets", *Industrial and Engineering Chemistry Research,* 26 **(1987)** 1869-1880

[45] Lawler L.E.; Lenstra J.K.; RinnooyKan A.H.G.; Shmoys D.B.: **"**The Traveling Salesman Problem", John Wiley & Sons, **(1985)**.

[46] Letchford, A.N.; Lodi, A.: "Strengthening Chavatal-Gomory Cuts and Gomory fractional cuts", *Operations Research Letters*, 30(2) **(2002)** 74-82.

[47] Li, F.; Golden, B.; Wasil, E.: "A record-to-record travel algorithm for solving the heterogeneous fleet vehicle routing problem", *Computers and Operations Research*, 34 **(2007)** 2734-2742.

[48] Lin, S.: "Computer solutions of the traveling salesman problem", *Bell System Technical Journal*, 44 **(1965)** 2245–2269.

[49] Malandraki, C.; Daskin, M.S.: "Time Dependent Vehicle Routing Problems: Formulations, Properties and Heuristic Algorithms", *Transportation Science*, 26 **(1992)** 185-200.

[50] Marchand, H.; Martin, A.; Weismantel, R.; Wolsey, L.: "Cutting planes in integer and mixed integer programming", *Discrete Applied Mathematics*, 123 **(2002)** 397-446.

[51] Martin, G.T.: "Solving the traveling salesman problem by integer linear programming", *CEIR*, **(1966)**

[52] Miliotis P.: "Using cutting planes to solve the symmetric traveling salesman problem", *Mathematical Programming 5* **(1978)** 177-188

[53] Mitchell, J.E.: "Branch-and-Cut Algorithms for Combinatorial Optimization Problems", *Handbook of Applied Optimization*, Oxford University Press, **(2002)** 65-77.

[54] Or, I.: "Traveling Salesman Type Combinatorial Problems and Their Relation to the Logistics of Blood Banking", *Northwestern University*, Unpublished PhD Thesis, **(1976)**.

[55] Padberg, M.; Rinaldi, G.: "A branch-and-cut algorithm for the resolution large-scale symmetric traveling salesman problem", *SIAM Review*, 33 **(1991)** 60-100.

[56] Padberg, M.; Rinaldi, G.: "Optimization of a 532 city symmetric traveling salesman problem by branch and cut", *Operations Research Letters*, 6 **(1987)** 1-7.

[57] Picard, J.C.; Queyranne, M.: "The Time-Dependent Traveling Salesman Problem and Its Application to the Tardiness Problem in One-Machine Scheduling", *Operations Research*, 26 **(1978)** 86-110.

[58] Schneider, J.: "The time-dependent traveling salesman problem", Physica A: Statistical Mechanics and its Applications, 314 (1-4) **(2002)** 151-155.

[59] Silih S.; Zula T.; Kravanja Z.; Kravanja S.: "MINLP Optimization of Mechanical Structures", *University of Maribor, Faculty of Civil Engineering* **(2000)**

[60] Stewart, W.R.: "A computationally efficient heuristic for the traveling salesman problem", *Proceedings of the 13th Annual Meeting of Southeastern TIMS*, Myrtle Beach, SC, USA, **(1977)** 75–83.

[61] Taillard E.D.: "A heuristic Column Generation Method for the Heterogeneous Fleet VRP", *RAIRO*, 33 **(1999)** 1-14.

[62] Tarantilis, C.D.; Kiranoudis, C.T.;Vassiliadis, V.S.: "A list based threshold accepting metaheuristic for the heterogeneous fixed fleet vehicle routing problem", *Journal of the Operational Research Society*, 54 **(2003)** 65-71.

[63] Tarantilis, C.D.; Kiranoudis, C.T.; Vassiliadis, V.S.:"A threshold accepting metaheuristic for the heterogeneous fixed fleet vehicle routing problem", *European Journal of Operational Research*, 152 **(2004)** 148-158.

[64] Tereshko, V.; Loengarov, A.: "Collective Decision-Making in Honey Bee Foraging Dynamics", *Computing and Information Systems Journal*, 9(3) **(2005)**

[65] Viswanathan, J.; Grossmann.I.E.: "A Combined Penalty Function and Outer Approximation Method for MINLP Optimization," *Computers and Chemical Engineering* 14, 769 **(1990)** 307-309.

[66] Wiel, R.J.V.; Sahinidis, N.V.: "Heuristic Bounds and Test Problem Generation for the Time Dependent Traveling Salesman Problem", *Transportation Science*, 29 **(1995)** 167-183.

[67] Wiel, R.J.V.; Sahinidis, N.V.: "An exact solution approach for the time-dependent traveling-salesman problem", *Naval Research Logistics*, 43 **(1996)** 797-820.

[68] Winston, W.L.: Operations Research: Applications and Algorithms, *Duxbury Press*, **(1994)** 639-720.

[69] Winston, W.L.; Venkataramanan, M.: "Introduction to Mathematical Programming: Operations Research", *Brooks/Cole-Thomson Learning*, California, **(2003)** 653-738**.**

[70] Wong L.; Low M.; Chong C.: "Bee Colony Optimization with Local Search for Traveling Salesman Problem" Singapore Institute of Manufacturing Technology 6 **(2007)** 1-7

CURRICULUM VITAE

Personal Information

Name, Surname	Hüseyin Demirkale
Address	Sahrayıcedit Mah. Müminderesi Cad. Arı Apartmanı No:6 D:8 Kadıköy/İSTANBUL
Phone	0216 368 75 23
E-mail	hdemirkale@gmail.com
Date of Birth	03.11.1984

Education

2002 – 2007	B.S. Computer Science and Engineering Marmara University Engineering Faculty, Department of Computer Science and Engineering Istanbul, Turkey GPA: 3.20/4.00
1997 – 2002	Trabzon Tevfik Serdar Anatolian High School, Istanbul, Turkey GPA: 4.95/5.00

Work Experience

Havelsan Center of Flight Simulator , Ankara , TURKEY
Project

Network Message control software using UDP sockets .
Graphical User Interface design with QT gui library

Interca software center, İstanbul, TURKEY
Project

Administrator panel for www.yeditepeli.com

Koçsistem Information and Communication Services, İstanbul, TURKEY

Java Developer in SAP systems