MATHEMATICAL MODELING OF THE FLYING PROBE TEST SYSTEM

MATHEMATICAL MODELING OF THE FLYING PROBE TEST SYSTEM

By

ALVIN TZU-CHIEN HSIEH.

Computing and Software

A Thesis

Submitted to the School of Graduate Studies

In Partial Fulfillment of the Requirements

For the Degree

Master of Science

McMaster University

MASTER OF SCIENCE, (August, 2010) McMaster University

(Computing and Software.) Hamilton, Ontario

TITLE: MATHEMATICAL MODELING OF THE FLYING PROBE TEST SYSTEM

AUTHOR: ALVIN TZU-CHIEN HSIEH

SUPERVISOR: Dr. Antoine Deza

NUMBER OF PAGES: ix + 73

Abstract

Electrical components are the building blocks of any electronics. These building blocks, when intelligently assembled, form circuits that behave as intended by the designer. Therefore, any errors, such as defective components and assembly errors will cause the end product to have abnormalities. Advancement in technology has led to circuits of higher complexity that require a greater quantity of components as well as a great reduction in individual component size. This translates to finer integrated circuits; more components are placed and packed in the same given area and thus, manual circuit board testing may no longer be feasible. The flying probe tester is an automated circuit board testing/verification system that uses electric probes to first stimulate a circuit and then read and verify its corresponding output values. This thesis examines the processes involved in the flying probe test system and produces a model that characterizes the current sequential method of testing test points. Furthermore, using existing techniques developed through the traveling salesman problem and linear optimization, an efficient model is developed to improve and limit the distance traveled by the probes, thus reducing the required testing time.

Acknowledgments

First and foremost, I would like to thank Dr. Antoine Deza, my supervisor, for giving me this opportunity to further my education. Thank you for your continual guidance and feedback throughout my thesis/research.

In addition, I would not have been able to complete my thesis without the extended support and knowledge of my peers, notably, Jessie Liu and Feng Xie. Your breadth of knowledge in this field helped further my understanding and background, and your assistance helped me overcome the minor obstacles I encountered during my research.

For their offer of the MITAC internship, which ultimately became my thesis topic, I would like to acknowledge Acculogic Inc. for providing the necessary materials/documents and information needed for my research. Special thanks to company VP, Farouk Eshragi, for giving me this opportunity and providing support every step of the way.

Lastly, I would like to thank my dad, mom, Melvin and Tat, for their continual support, motivation. They have encouraged me to avoid complacency and to constantly push the envelope.

Contents

List of Figures

List of Tables

Chapter 1: Introduction

1.1 Circuit Boards and Reason for Testing:

Any technology consists of electrical components that together form electric circuits. In unity these circuits (placed on a circuit board) provide the functionalities of devices and systems. In electronics, multiple circuit boards are placed/installed together to realize a full system. This is much like a desktop computer, with its motherboard, video card, sound cards, etc, each providing a unique feature of computing (figure 1-1 below). This large system of circuits and the numerous components that it is comprised of causes the testing and the troubleshooting of the end product to be time consuming and difficult in most cases. With luck, the error can be traced back to the specific circuitry responsible, much like how a computer that can no longer be powered on could be caused by a defective power supply (the circuits within it). However, even if knowing the power supply is at fault, isolating the exact defective components or circuits could be difficult and extremely time consuming.

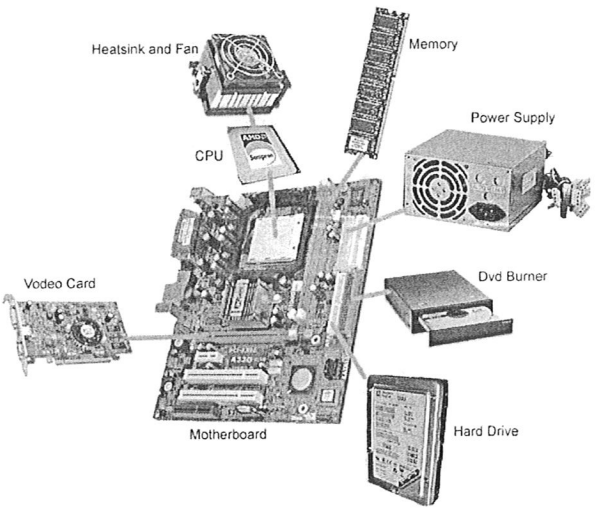


Figure 1 - 1: Different circuit boards working in unity to realize a computer system. [13]

To avoid testing mindlessly at the component level, circuit board designers generally place test points on various parts of the board. These test points allows testing at a circuit level (however small or large), treating it as a black box. By giving some voltage (or current) input to the circuit, v_{in} in figure 1-2, verification can be made against the output, v, to see if the circuit is working. To read electric values of components and test points, external test probes must make contact with them (figure 1-3). Since circuits together form bigger circuits, testing will begin at a general level to specific components. This is reason as to why testing is time consuming.

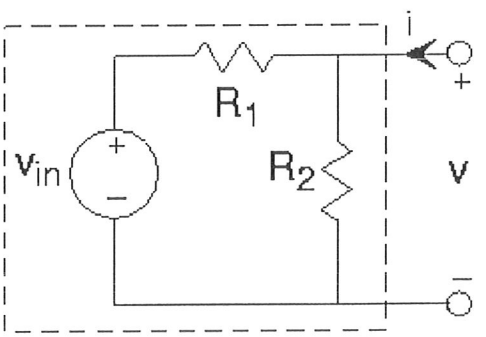


Figure 1 - 2: Black Box Testing [12]

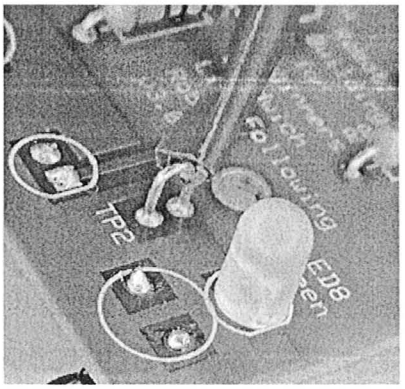


Figure 1 -3: Reading the values of a test point (TP2) to read its values [5].

Testing is done either at the production stage or trouble shooting stage. The decision of how often to test and how thoroughly lies with the manufacture or tester. Excessive testing may drain resources and may not be feasible given the cost it takes to test is often more expensive than replacing a circuit board all together. However, any early undetected error at the production stage will escalate, increase the amount of time spent on debugging the end product, and potentially losing consumer confidence.

1.2 Finding an Efficient Test Path:

Testing circuits today have shifted to using automated systems. The research of this thesis is based on the work done during an internship at a company in the automated testing instrumentation market. The instrument that was studied was a flying probe tester, which uses electric probes to make contact with test points, to generate inputs or read outputs. The instrument itself is part of the flying probe tester system, which also includes a computer to provide the means of control for the user. In short, the tester tests components and test points that are specified by the user, but the path taken by the probes is of a sequential order. This testing sequence is not an efficient method of testing, as the probes may potentially be required

to travel the total length of the board, back and forth, frequently. This thesis deals with modeling the flying probe test system and providing an algorithm/model that is able to find an efficient path of testing, that is superior to the original sequential path. The term efficient in this sense refers to a shorter distance traveled by the probes during the duration of a complete set of tests. The intricacies of the flying probe test system will be introduced in the proceeding chapter.

Chapter 2: The Flying Probe Test System

The flying probe test system consists of the main testing unit, as well as a computer unit that runs the software that controls the overall system (figure 2-1). On the hardware side of the system, a circuit board is suspended on a conveyer belt, supported only by the two edges that rest slightly on the belt (figure 2-2). The conveyer belt slides the board further into the machine while being sensed by cameras/sensors. Once the correct board position is determined by cameras, the testing commences. Tests are able to check for defective components as well as any misplaced ones, as they would produce different output values than the expected ones. However, it is up to the technician to specify how many components to test on a circuit board. Also, there are numerous types electrical components, each with their unique electrical characteristics. Testing them requires different techniques and approaches.

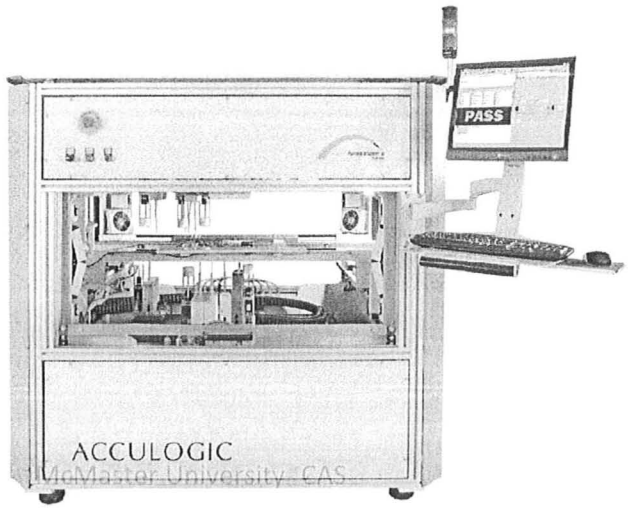


Figure 2 -1: Acculogic's Flying Scorpion, a flying probe tester [2]

Testing resistors involves checking its resistance values (through current and voltage), and capacitors requires checking its capacitance value as through voltage and discharge time. For testing each component, a minimum of one probe is used for providing a voltage/current source to the positive terminal of the component, and one more probes to read the corresponding output value at the other terminal. For example, by having a probe supplying a current to a resistor, we are able to measure the voltage output across the resistor with the other probe. Calculating its resistance value can be done through: Resistance= Voltage/Current. We would then check whether or not that value matches the one intended by the designer (data stored in the software).

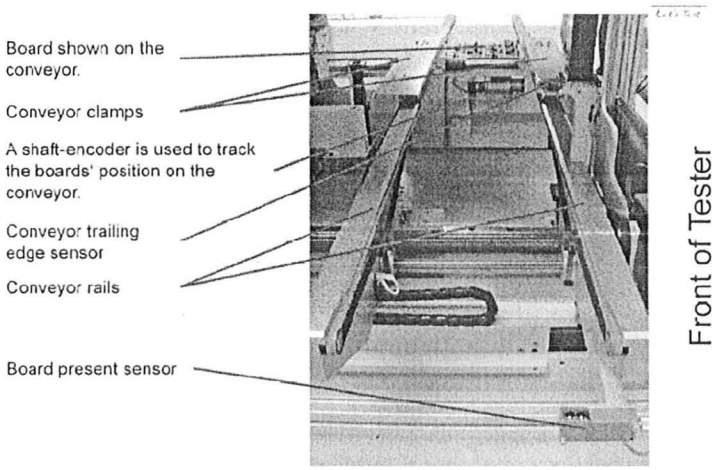


Figure 2 -2 Conveyer Belt of the Flying Scorpion [2]

The purpose of the flying probe test or any circuit board testing for that matter is to make sure the components are assembled properly, correct parts were used, as well as ensuring the functionality of the board is being realized. Manual testing all the components of a circuit board could be a long and tedious task, and would not be feasible in an industrial manufacturing environment. These automated testers are necessary to ensure the quality of the boards as well as maintaining a certain productivity level of the manufacturing company.

As for the software side of the system, how it works in general is that the circuit board's CAD file and BOM (bills of material) file are imported to the tester's software. Each board is represented as a project file. The board's CAD file includes the schematics of the circuit board while the BOM file includes all the components on the circuit board. Together, they provide enough information to the software so that it is able to translate/virtualize the physical geography of the board and translate that to the coordinate system of the tester system itself. In the software's database, it usually contains detailed information about commonly used components, such as their dimensions, characteristics, etc. The components read in from the BOM file should automatically associate with the components already stored in the database and appear as

individual entries within the project file. The user can then select which components to test by enabling/disabling these entries. Thus, when a desired test point needs to be tested, the system knows the exact x and y coordinate to place the test probes, as well as the method of testing required for that specific component.

Sometimes, the physical profile of a component prevents the probe making contact with a test point in a certain manner. The tester will need to acknowledge this and would attempt to approach that test point in a feasible way. Probes also need to avoid hitting components that have a great depth (tall), such as going around the component. The tester's software must take these physical restrictions into account and sometimes, it may require human monitoring. For the most part, the software is programmed to route feasible movement only, avoiding the component in question.

A circuit like the one in figure 2-3 below represents a parallel circuit. The four resistors in the figure are said to be "parallel" to each other, meaning that their positive terminal belong to the same node/junction. The terminals on the same junction have the same electric potential. Even though the electrical components may physically be placed apart on the circuit board, they are still connected, not visible to us. Assuming that for all four resistors, the top terminal represents the positive terminal and the bottom represents the negative. Let the left most resistor be labeled R_1 and the rightmost R_4. Each resistor has 2 terminals, and has the coordinates of:

R_i with coordinates (x_{i+}, y_{i+}) & (x_{i-}, y_{i-}); i = 1, 2, …, n (number of resistors)

Traditionally, to test R_1, we need to place the probes at the two terminals (x_{1+}, y_{1+}) and (x_{1-}, y_{1-}). However, since R_1 and R_2 are in parallel and have the same positive terminal, we can use R_2's

positive terminal (or R_3 and R_4 for that matter). So we can also use (x_{2+}, y_{2+}) along with (x_{1-}, y_{1-}) to test R_1 if (x_{1+}, y_{1+}) is inaccessible.

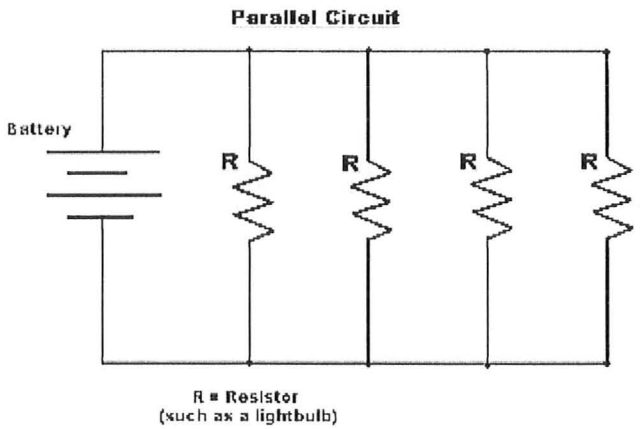


Figure 2 -3: Parallel Circuit [2]

The real benefit of this for parallel circuits is that when testing, we can place one probe on (x_{1+}, y_{1+}), the common positive terminal for all 4 resistors, and only move the probes for the negative terminals. This allows the first probe to remain stationary for 3 test sequences. The ability to switch a test point for another one of the same electrical potential is called point exchange and this is a feature in the test system's software. This feature when enabled may reduce the overall distance travelled by the probes.

When all the components are loaded in the project files, they are organized by categories. For example, all the resistors (R_1 to R_n) are followed by all the capacitors, followed by all the inductors. The system will use this chronological sequence of components as the testing sequence. The problem with this approach is that components that are next to each other in naming (ie. R_1 and R_2) are not necessary close to each other physically on the board. Having this sequential order will result in a test path that jumps all over the place. The technician is able to

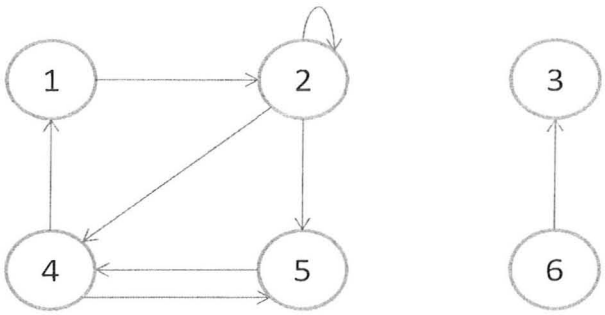
rearrange the sequence manually, but it is the goal of this thesis to produce an algorithm that finds a path for solving that will minimize the distance travelled by the probes.

Chapter 3: Graph Theory

3.1 Fundamentals:

This section introduces several elementary notions of graph theory that is used in the model we apply. Test points on the flying probe tester can be viewed as nodes in a graph and is modeled as such in this thesis. We can thus take advantage of the well known and developed work of graph theory to serve as the foundation of our model. Before proceeding, the following convention needs to be mentioned. A capital letter will refer to a set of values, while small caps will refer to individual elements.

A graph G consists of a set of vertices V, and a set of edges E: G = (V, E). The finite set of vertices, V, consists of all individual vertex, v_0 to v_{n-1}, in the graph, where n is the number of vertices in the graph [4]. The finite set of edges, E, consists of the edges, e_0 to e_{k-1}, that form the graph, where k is the number of edges in the graph. An edge is a relationship between two vertices, indicating they are connected in some manner (described below).



3 - 1: Directed Graph

3.2 Directed Graphs:

In a directed graph (figure 3-1), an edge is marked by arrows and it limits travel between nodes in the direction of the arrow, much like an one-way-street. An edge can be written as, e = (u, v), and is equivalent to saying the edge is leaves from u and the edge is incident from u, incident to v [4]. The case u = v is permitted in directed graphs, as illustrated by node 2, indicates a self-loop. Note that e = (u,v) ≠ e = (v,u), due to the direction restriction. Thus, for e = (u,v), we say u is adjacent to v, but v is not adjacent to u. illustrating the non-symmetric property of edges in directed graphs.

(u,v)	1	2	3	4	5	6
1	0	1	0	0	0	0
2	0	1	0	1	1	0
3	0	0	0	0	0	0
4	1	0	0	0	1	0
5	0	0	0	1	0	0
6	0	0	1	0	0	0

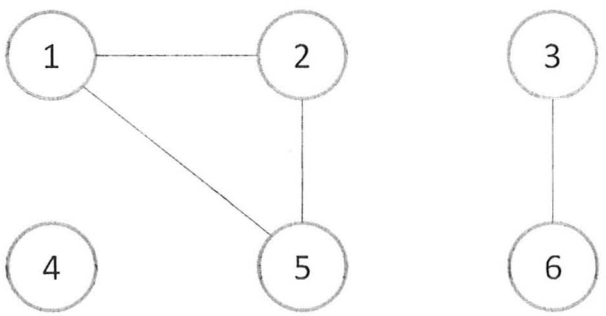
Table 3 – 1: Adjacency matrix of the directed graph of figure 3-1

Another way to represent the directed graph is with its adjacency matrix (table 3- 1). Each row represents u in E = (u, v), and each column represents v. When an edge exists, the

corresponding matrix element stores a 1, as illustrated by matrix element (1, 2). Adjacency matrix is needed for graphs to be stored in the computer and requires Θ ($|V|^2$) [6] memory for storage, V being the set of vertices.

3.3 Undirected Graphs:

The other type of graph is the undirected graph. Its major difference from directed graphs is that the edge has no directions, allowing passage in both directions between the two connected vertices. Figure 3- 2 shows an undirected graph, and the edge connecting (1, 2) allows 1 to go to 2 and vice versa. For a more formal definition, an undirected graph G = (V, E) has unordered pairs of vertices, and the edge set E consists of {u, v}, where u, v $\in$ V & u $\neq$ v [4]. The last restriction restricts any self loop in undirected graphs, unlike the directed version, requiring two distinct vertices in any edge. Also, (u, v) = (v, u) for undirected graphs, unlike the directed version as well. Table 3- 2 below represents the adjacency matrix of the undirected graph in figure 3- 3. Notice the symmetry across the drawn line between (u, v) & (v, u).



3 - 2: Undirected Graph

(u,v)	1	2	3	4	5	6
1	0	1	0	0	1	0
2	1	0	0	0	1	0
3	0	0	0	0	0	1
4	0	0	0	0	0	0
5	1	1	0	0	0	0
6	0	0	1	0	0	0

Table 3 – 2: Adjacency matrix of the undirected graph of figure 3-2.

3.4 Graph Properties:

Having established the two graphs, we now discuss some of their properties. In an undirected graph, the degree of a vertex is defined as the number of edges connected to it. Vertex 2 in figure 3- 3 has a degree of 2, while vertex 4 has a degree of 0, because it is not connected to any vertex, thus isolated. As for a directed graph, since the edges have direction, we can talk about the in-degree and out-degree. The overall degree is the sum of the two, which is essentially counting all the edges that the vertex is connected to, much like the undirected graph. In figure 3- 1, both of vertex 4's in-degree and out degree are 2, making its overall degree 4.

Say in a graph G = (V, E), we wish to go from a vertex, v_1, to another vertex v_k. The path of the journey requires taking vertices, $\{v_1, v_2, ..., v_k\}$, such that edge $(v_i, v_{i+1}) \in E$ for i = 1, …, k. This means that we are traveling through the intermediate vertices and edges, $E(v_1, v_2)$, $E(v_2, v_3)$, …, $E(v_{k-1}, v_k)$ until we reach the destination v_k. The path length refers to the number of edges involved in the path. In figure 3- 1, the path length from vertex 1 to 4 is 2 (due to direction of the edges, we must go to vertex 2 first). In figure 3- 2, the path length from vertex 1 to 4 is 0, because there is no way of getting there. The path is said to be simple, if all vertices in the path

are distinct, never visited twice. A sub-path refers to a portion of the original path $\{v_1, v_2, ..., v_k\}$, as long as the initial vertex is v_i, $i >= 1$, the final vertex is v_j, $j <= k$ and $i < j$. So a sub path of the original could be $\{v_2, v_3, ..., v_{k-3}\}$. The distance of the path traveled however, refers to the summation of all the edge weights in the path. Though the two graphs above are not assigned edge weights, it will be used later in the modeling section of this thesis.

In a directed graph, a path that is able to return to the initial vertex, $v_1 = v_k$, is called a cycle. In figure 3- 1, if vertex 1 is the starting vertex, there is a path that cycles back to it, specifically $v_1 \rightarrow v_2 \rightarrow v_5 \rightarrow v_4 \rightarrow v_1$ (ignoring the possibility of looping at v_2). On the contrary, a graph with no cycles is called acyclic. Self loops is also considered a cycle, with length 1. If in a cycle, all the vertices are unique (first last vertex still have to be the same to qualify as a cycle) and visited exactly once, the cycle is said to be simple. In figure 3- 5 below, vertex 1, 2 & 3 forms a cycle, but not a simple cycle, because in order to get back to vertex 1, vertex 2 is traversed again, thus all the vertices in the path are not unique. Vertecies 4, 5 & 6 however is part of a simple cycle, with the cycle being $v_4 \rightarrow v_5 \rightarrow v_6 \rightarrow v_4$. Furthermore a simple cycle consisting of all v's in V is called a Hamiltonian cycle.

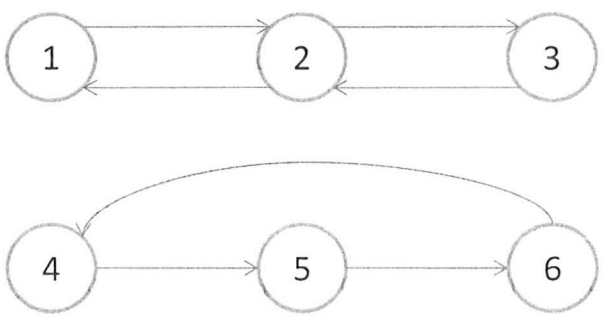


Figure 3 – 3: Vertex 1, 2 & 3 is not part of a simple cycle, but vertex 4, 5 & 6 is part of a simple cycle

Chapter 4: Linear Optimization

4.1 Operations Research:

Operations Research (OR), is defined as "representation of real-world systems by mathematical modeling together with the use of quantitative methods (algorithms) for solving such models". It is the branch of mathematics commonly used to find optimal solutions to real-life problems. The term optimal refers to optimizing the objective function of the problem, such as maximizing profits and minimizing costs. To do so, OR uses a wide range of mathematical techniques including graph theory from the previous chapter, statistical analysis, mathematical modeling to name a few.

When translating a real word problem to its math counterpart (to be analyzed and solved), there are 3 components that need to be defined. The first component is variables and it is all the different factors that are decision based. Their values are determined by the user and directly change the value of the solution. The second component is constraints. They deal with things such as limited resources that impose certain restrictions. An example of this could be land

governing how much crops a farmer can grow. The last component is an objective function that needs to be optimized (maximize or minimize). The objective function essentially represents what the problem is trying to achieve. The solution to the overall problem is a specific set of values (for variables in the objective function) that optimizes the objective function, ultimately, indicating how to operate most effectively.

When the OR problem that we are trying to optimize have variables that are all continuous, contains a single objective function that is linear, all constraints are linear, we are able to use the method of linear optimization. It is used extensively to solve problems in the business and economics world, as well as some fields of engineering.

4.2 Linear Optimization:

When the objective function is a linear function along with linear and non-negative constraints, linear optimization can be used. Linear optimization (LO) has the general canonical form of:

$$\text{Maximize: } c^T x$$

$$\text{Subject to : } Ax \leq b$$

The problem consists of variables for which we're trying to determine the values,

$$x_1, x_2, \ldots, x_n$$

which will maximize/minimize some objective function,

$$z = c_1x_1 + c_2x_2 + \ldots + c_nx_n$$

subject to certain constraints.

$$a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \leq b_1$$

$$a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n \leq b_2$$

$$\vdots \qquad \vdots \qquad\qquad \vdots$$

$$a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n \leq b_n$$

With these conditions the constraints form a convex polytope, illustrating a feasible region. Any point in the feasible region has a set of specific values for $(x_1, x_2,\ldots,x_n)$ which satisfies all the constraints and thus giving some value when substituted into the objective function, z. Finding the specific point in the feasible region that gives the optimal (max/min depending on the goal) object function value is the overall goal of linear optimization. The optimal point may not be unique.

If the solution needs to take the form of integers, then the problem is called Integer Linear Optimization, though its constraints can still be fractional. The integrality constraints make Integer Linear Optimization usually much harder to solve. A common approach to tackle the computational challenge is to use linear optimization to get fractional values, and then apply rounding techniques that will give rise to integer values [16].

Example [17]:

Let x and y denote the amount of tomatoes and squash to be grown respectively in square meters. A farmer who wishes to maximize his revenue (the objective function), selling a square meter of tomatoes for \$3 and \$5 for squash. The first constraint deals with fertilizer, as a square meter of tomatoes need 2 units of fertilizer and squash needs 3 units. The farmer has a total of 10 fertilizers so the total used has to be at most that. The second constraint deals with

insecticides, as a square meter of tomatoes and squash both need 1 unit. The total of units of insecticides the farmer has is 4. The third and forth constraint is trivial and states that the farmer cannot grow negative crops. The fifth constraint deals with the amount of tomato seeds available, and the famer only has enough for 4 square meters. The last constraint is similar to the previous, with the farmer only have enough seeds for 3 square meters of squash.

Objective Function (to be maximized): $z = 3x + 5y$

Constraints: $2x + 3y \leq 10;\ x + y \leq 4;\ x \geq 0;\ y \geq 0;\ x \leq 4;\ y \leq 3$

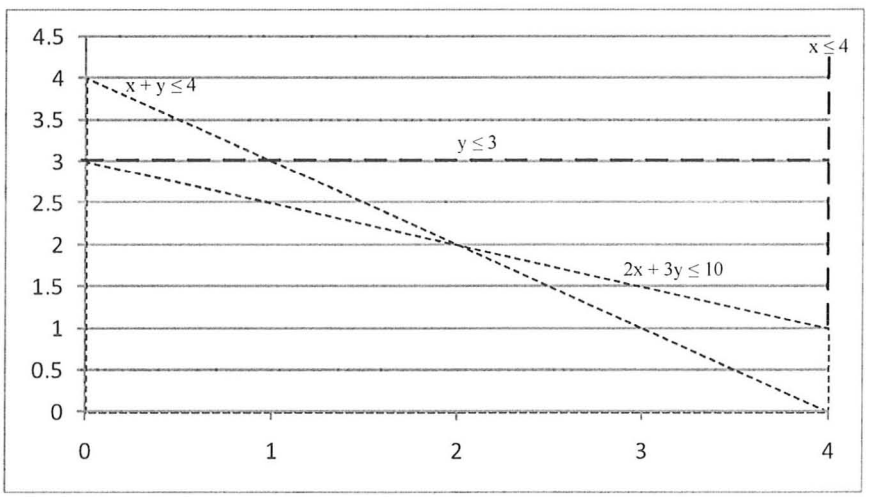


Figure 4 - 1: Feasible region of the linear optimization proble m

Figure 4- 1 above shows the feasible region in yellow. This is obtained by plotting all the constraints and observing their intersections. Any combination of x and y in the feasible region will satisfy all constraints, but there usually is a specific point where it provides the optimal solution. In figure 4- 2 below, the optimal solution is obtained when x = 2 and y = 2. By growing 2 square meters of both tomatoes and squash, the farmer will produce the optimal revenue of \$3*2 + \$5*2 = \$16.

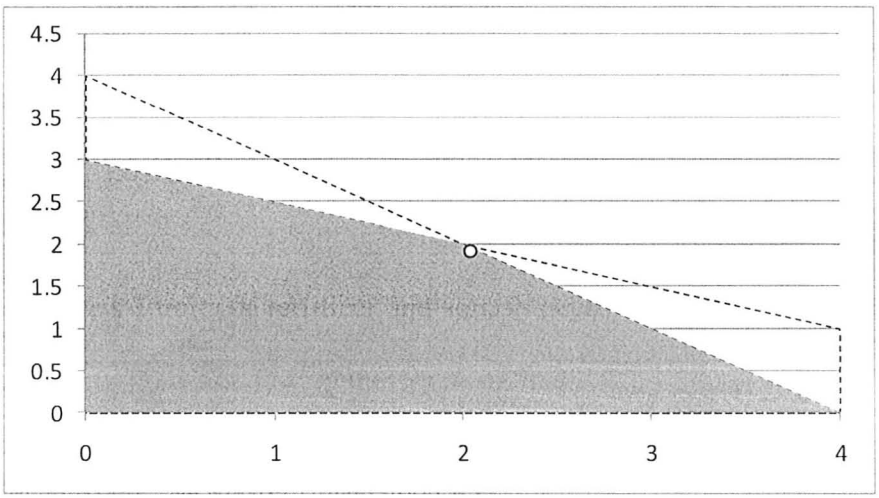


Figure 3 - 2: The optimal point in the feasible region

4.3 Simplex and Interior Point Method:

The above example is 2-dimensional (dealing with only 2 variables x and y) and by plotting the constraints, the feasible region forms a polygon and can be visualize easily. Anything beyond 3-dimensions cannot be visualized geometrically and must be solved mathematically. The most commonly used methods for finding the optimal point(s) in the feasible region are the simplex and the interior-point-methos, and each one has many variants. The first is the simplex method created by George Dantzig in 1947 [3]. The idea behind it is shown in figure 4-2. By traversing adjacent edges of the polytope, the optimal point can be found as it always lie on an edge. While the simplex method is not proven to be polynomial, it is efficient in practice and widely used.

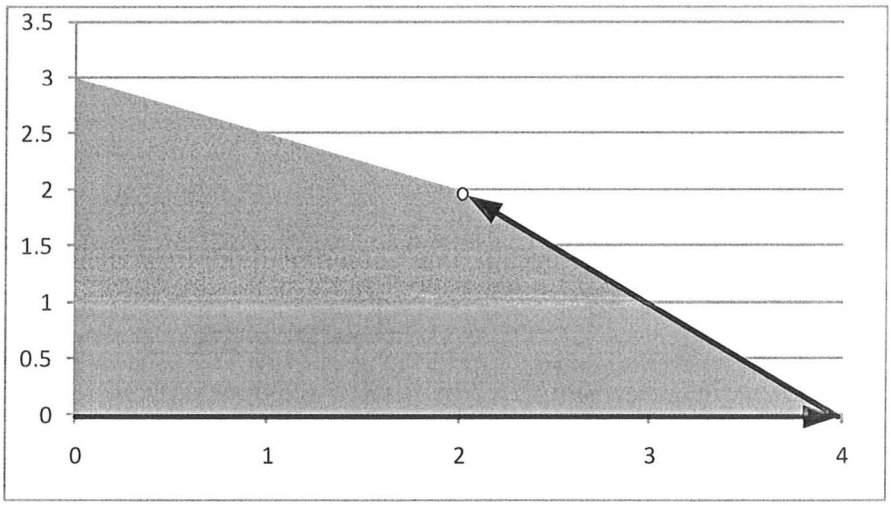


Figure 4 - 3: The approach of the simplex method

The other algorithm is the interior point method, illustrated in figure 4- 3. The idea behind it is to start examining feasible points from within the feasible region of the polytope (not on any edges), and converge outwards (graphically) until the optimal solution is reached [16]. The interior point method is both polynomial time for both the worst case and the average case, as proven by Karmarkar in 1984 [22], and very efficient in practice.

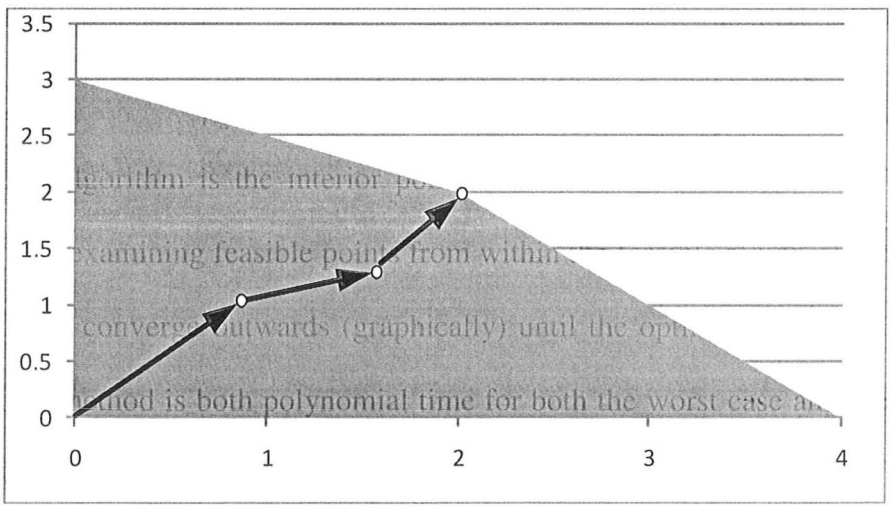


Figure 4 - 4: The approach of the interior point method

4.4 Commercial Solvers:

There are numerous commercial software that are able to solve linear optimization problems. Two commonly used ones are CPLEX and SeDuMi. CPLEX is a software package that solves optimization problems. The name is a combination of the simplex method and interfacing with the C programming language. Since its existence, it has expanded to include interior point method and interfacing with C++, C# and Java. SeDuMi on the other hand is under GPL (general public license) as long as the user has a license with Matlab, as it is a Matlab library. It uses interior point method which has polynomial complexity [15].

Chapter 5: Modeling of the Flying Probe Test System

The objective of the thesis is to model the flying probe tester such that we can use some algorithm to find an efficient testing path. To do so, some features of the flying probe tester will be relaxed in order to reduce complexity of the problem. This will allow the model to relate to some of the existing work done in the Traveling Salesman Problem (TSP). TSP is the mathematical problem of minimizing the travel cost through a finite number of cities (looping back to the beginning), given the cost between each possible pair of cities. What we wish to accomplish through the model is to obtain an optimal testing sequence. The distance travelled of this optimal testing order needs to be an improvement over the default sequential order (set by the tester).

5.1 Notations and Objective Function:

Consider a circuit board that is ready for testing. We define a single test-step as the individual testing done to a single component. For example, testing an arbitrary and single resistor R_1 counts as one test-step and testing an arbitrary capacitor C_1 counts as another independent test-step. Testing of the whole board includes executing the test-steps specified by the technician (not all components need to be tested). The testing sequence of these test-steps is the main focus of this research. Stored within each individual test-step, are its associated test points. For example, if our current test-step involves testing an arbitrary resistor R_1, the associated test-points that will be stored are the x-y coordinate system of the resistor's positive and negative terminals respectively. The computer is able to place the probes to those exact positions during testing.

Let n = the total number of the test-steps that needs to be executed. For each test-step T_i ($i \leq n$), the coordinates of all its associated-testing-points are recorded as $P_1(x,y)$, $P_2(x,y)$, etc. The naming convention used for each test-step's associated-test-points always start with P_1, then P_2. To differentiate the P1's of different test steps, the dot notation will be used. For example, T_1 has its associated-test-points of $P_1(0,0)$ & $P_2(10,10)$, while T_2 has its associated-test-points of $P_1(20,20)$ & $P_2(30,30)$. They can be referred to uniquely as T1.P1, T1.P2, T2.P1 and T2.P2 respectively. On absolute terms, all these test points are unique, and the integrator software will name them uniquely.

In order to reduce the complexity of the model, here are some of the assumptions made about the flying probe tester:

- For each test-step, only one set of associated-test-points exists, meaning the point-exchange feature is disabled.
- We ignore the height of all components so that the probes don't have to go around any objects. Also, we ignore the the z-axis movement of probes, restricting all movement to 2D.
- The flying probe travels in a straight line manner from point to point. This assumes that the shortest and direct path is taken for all movements.
- The speed at which the probes move is constant.
- In TSP, we tend to traverse through cities. However, each test-step has 2 or more associated-test-points and thus requires two probes to move simultaneously. Since all the associated-test-points are relatively close to each other, each test-step's $P_1(x,y)$ will be used as the landmark that represents the entire test-step. That is, all test-steps are represented as a city, with the location of its $P_1(x,y)$. A complete tour will cycle through all these single points that represent different test-steps.

Here, we define some test-steps and their associated-test-points with arbitrary values assigned:

$$T_1.P_1(2,3)\ T_1.P_2(4,5) \quad T_2.P_1(4,3)\ \&\ T_2.P_2(5,2) \quad \ldots \quad T_n.P_1(3,4);\ T_n.P_2(12,30)$$

The goal now is to build a model that tries to find the shortest distance of travel when cycling (testing) through all test-steps. This problem takes the form of the classical TSP as mentioned, and it is a NP-hard problem.

We now define the functional variable x(i,j) and it takes two values, 0 & 1. The variables i and j represent test steps T_i and T_j,. For simplicity, only the subscript of of T_i and T_j will be

used when describing x(i, j). When x(i,j) = 1, it means that the straight and direct path from test-step i to j is included as a part of our optimal tour. Similarly, x(i,j) = 0, means that the straight path from test-step i to j is not part of the optimal tour, meaning test-step i doesn't go directly to j, but rather i goes to some other test-step. Our overall goal is to find all the paths that are a part of the optimal tour, such that all test-steps are visited exactly once, while minimizing the total distance travelled in the trip. In other words, find all x(i,j)'s where i = 1.

The test-steps on the circuit board can be represented as an undirected graph, as any point is able to reach any other point at a distance, having the exact same distance backwards. Thus, the i-to-j path and the j-to-i path are referring to the same edge and we only need to specify this value once. For example, x(1, 2) = x(2, 1), so storing the latter is redundant. For simplicity, in x(i, j), variable i will always refer to the test-step of the lower naming value than j, i < j. Thus, T_1 is lower than T_2, is lower than T_{10}. Figure 5-1 below an example of a undirected graph which represents how circuit boards will be modeled. Table 5- 1 shows the adjacency matrix of the graph in figure 5- 1.

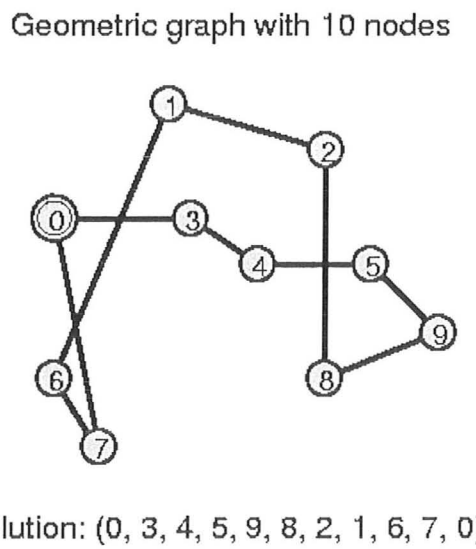


Figure 5 – 1: A circuit board and its test-steps are able to be represented as a Hamiltonian cycle and an undirected graph [19]

Based on the modeling work, a flying probe TSP with n number of test-steps would have n(n-1)/2 number of x(i,j)'s to be determined. In figure 5- 1, there are 10 test-steps and the edges represent the path to be taken during testing. We would need 10(10-1)/2 = 45 x(i,j)'s as illustrated in table 5- 1. Having all the data tabulated in a matrix, it becomes easier to work with as will be seen later on.

x(i,j)	j = 0	1	2	3	4	5	6	7	8	9
i = 0		0	0	1	0	0	0	1	0	0
1			1	0	0	0	1	0	0	0
2				0	0	0	0	0	1	0
3					1	0	0	0	0	0
4						1	0	0	0	0
5							0	0	0	1
6								1	0	0
7									0	0
8										1
9										

Table 5 - 1: x(i,j) matrix of TSP in Figure 1. There are 45 (i,j)'s

Since the objective is to minimize distance traveled, we need variables that store the physical distance between edges. Let w(i, j) be the weight variable that stores the distance between test-step T_i & T_j. Once again, only the subscript will be used. Using the following equation, it is straight forward to calculate w(i, j):

$$w(i, j) = dist|Ti.P1-Tj.P2| \qquad (5.1)$$

If T_1 is represented by its $P_1(2,3)$ and T_2 is represented by its $P_1(4,3)$, $w(1,2) = dist|T_1(P_1)-T_2(P_1)| = dist|(2,3)-(4,3)| = 2$

Combining x(i, j) and w(i, j), the total distance traveled by the probes during a tour of testing can be computed by:

$$\sum w(i,j) * x(i,j) \qquad (5.2)$$

This is the objective function that will be used in linear optimization for minimizing the probe distance. Since x(i, j) only take the values 0 or 1, this type of problem is classified as the 0-1 LO (linear optimization, also known as linear programming) problem.

5.2 Fractional 2-Factor LO:

In a Hamiltonian cycle which represents the tour that the probes undertake, there is a property that is common to all test-steps. For any test-step, T_i, there exist exactly 2 edges, one arriving at T_i from another test-step and the other leaving T_i to another test-step. The test-steps are represented as an undirected graph because the tour can go in either direction, but there is still an incoming and outgoing edge when testing starts. With this property, for some test-step k:

$$\sum x(i, j) = 2; \quad i \text{ or } j = k \qquad (5.3)$$

Proof:

If an arbitrary test-step T_3 is connected to T_2 and T_9, x(2, 3) = 1, x(3, 9) = 1, all other x(i, j) that have either i = 3 or j = 3, would = 0. This will give us the overall sum of x(i, j) with either i = 3 or j = 3, equal to 2.

This property allows us to introduce constraints to our linear optimization model, to one known as the fractional 2-factor LO (2-factor being that x(i, j) takes two values):

$$\text{Minimize} \quad \sum w(i,j) * x(i,j) \qquad (5.4)$$

Subject to:

$$\sum_{i=1}^{k-1} x(i,k) + \sum_{j=k+1}^{n} x(k,j) = 2 \qquad (5.5)\ [3]$$

$$0 \leq x(i, j) \leq 1$$
$$1 \leq i < j \leq n$$
$$k = 1, 2, 3 \ldots n$$

There are commercial and open-source solvers that will be able to solve LO problems as described in section 4.4. By giving the solver the appropriate inputs, it will output the x(i, j)'s, which corresponds to the paths of the overall tour. However, since TSP is a NP-hard problem [3], it is difficult computationally and mathematically for the solver to find integer values (0 and 1) for the x(i, j)'s directly. Fortunately, the fractional LO techniques discussed in section 4.2 can be used to obtain solutions in fractions, ie. x(i, j) = 0.98. After applying rounding to those numbers, it can give us integer solutions that are desired.

Ultimately, the correct solution will output a tour that is in fact a Hamiltonian cycle that includes all the test-steps. However the solutions we would get from LO solvers may not represent the complete tour that is desired, even though the undesirable solution does satisfy all the mathematical constraint above.

Proof: In figure 5-2, the left graph represents the Hamiltonian Cycle that includes all the test-steps and is the optimal solution. The right graph contains 2 separate sub-tours. Though it is not the solution to our problem, it does mathematically satisfy the constraints of the fractional 2-factor LO. Each test-step has exactly 2 edges though there are no edges connecting the two sub-tours.

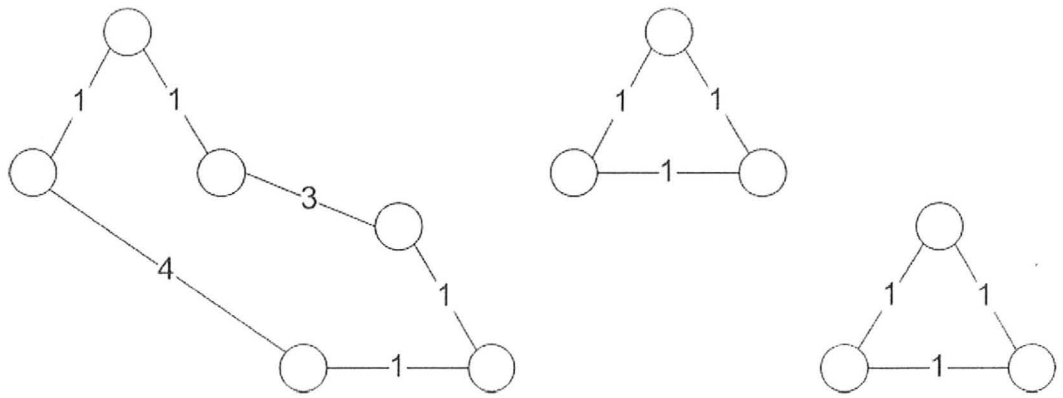


Figure 5- 2:- The figure on the left represents the desired solution, a Hamiltonian Cycle that traverses through all the testp-steps. The figure on the right represents two sub-tours. This illustrates the solution we may get undesirable solutions from the fractional 2-facor LO, even though it works out mathematically.

By chance, the optimal solution from the solver does in fact form a complete Hamiltonian cycle. However, it is common that the LO solver gives a solution that does not correspond to the desired complete tour. Those values can be used as a lower bound and constraints used for further computation (through iteration). This will guide us towards the optimal solution that will eventually produce a complete tour. The rest of the thesis deals with finding algorithms and introducing constraints that will forbid sub-tours from forming in the solution.

5.3 Solving the Testing Path Using TSP Techniques:

Before proceeding to sub-tour elimination, we will apply the models developed so far, to solve a hypothetical example. Say we have a circuit board that requires testing, and it has a total of 6 test.-steps. The first step is to translate the layout of the circuit boards and the location of the test steps, into to a graph, as illustrated in figure 5- 3. For simplicity, the routable edges are given in the figure (though in reality, every vertex is able to form an edge with every other vertex), as well as their corresponding edge weight.

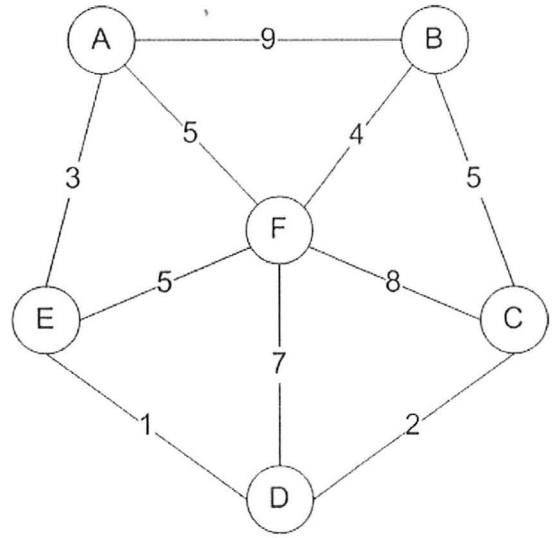


Figure 5 - 3: Layout of test-steps

Since the size of the problem is small, the optimal solution may be easily visualized. However, a small sample size is chosen to demonstrate the modeling work of this thesis in a feasible manner. The optimal solution is shown in figure 5- 4 below.

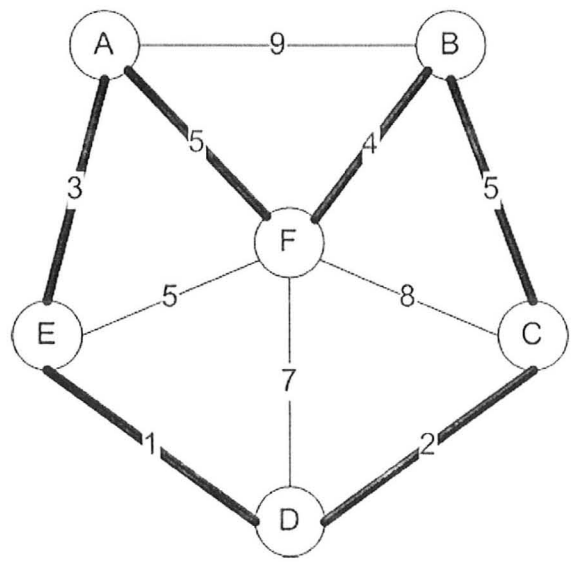


Figure 5 - 4: Optimal Tour of figure 5- 3

Keep in mind that vertex A is a test-step and it represents the first test point of the multiple test points associated with test step A. The same goes for the rest. Table 5- 2 below illustrates the naming of the edges in our graph.

AB → x_1	CD → x_6
AE → x_2	CF → x_7
AF → x_3	DE → x_8
BC → x_4	DF → x_9
BF → x_5	EF → x_{10}

Table 5 - 2: x(i,j) matrix of TSP in Figure 1. There are 45 (i,j)'s

Having all the necessary information, we can proceed and treat this as a fractional 2-factor LO. Using (5.4), (5.5) and the edge weights, the problem is modeled as the following:

Minimize:

$$9x_1 + 3x_2 + 5x_3 + 5x_4 + 4x_5 + 2x_6 + 8x_7 + x_8 + 7x_9 + 5x_{10} \qquad (5.6)$$

Subject to:

$$\begin{aligned}
x_1 + x_2 + x_3 &= 2 \\
x_1 + x_4 + x_5 &= 2 \\
x_4 + x_6 + x_7 &= 2 \\
x_6 + x_8 + x_9 &= 2 \\
x_2 + x_8 + x_{10} &= 2 \\
x_3 + x_5 + x_7 + x_9 + x_{10} &= 2
\end{aligned} \qquad (5.7)$$

$$0 \le x_i \le 1 \qquad i = 1, 2, \ldots, 10$$

The constraints all have the R.H.S. equal to 2, corresponding to the fact that the vertices represent a Hamiltonian Cycle. As the last vertex loops back to the first, each vertex will have an edge going in, as well as going out. The last constraint illustrates the permitted values of x_i's. Equaling to zero means that edge is not used and vice versa. Keep in mind that this is a fractional 2-factor LO problem, so the values of x's are permitted to be non-integers.

The solver that was used was SeDuMi, which is designed to run in the Matlab environment. Since the LO solves for min. $c^T x$ subject to $Ax \leq b$, we need to translate the constraints into their corresponding matrix form. Table 5- 3 contains 3 matrices that correspond to the constrains of the linear optimization problem. For matrix A, the first row with the variable names is not part of the input, but is just there for reference.

A =

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}
1	1	1	0	0	0	0	0	0	0
1	0	0	1	1	0	0	0	0	0
0	0	0	1	0	1	1	0	0	0
0	0	0	0	0	1	0	1	1	0
0	1	0	0	0	0	0	1	0	1
0	0	1	0	1	0	1	0	1	1

b =

2	2	2	2	2	2

c =

9	3	5	5	4	2	8	1	7	5

Table 5 - 3: Constraints in matrix form

The matrices in table 5- 3 cannot be used with SeDuMi as is because SeDuMi solves the problem in the format of:

$$\text{min. } cTx$$

$$\text{subject to } Ax = b \qquad (5.8)$$

$$x_i \geq 0$$

In order to impose $x_i \leq 1$, the 3 matrices in table 5- 3 must be modified. In order to add the $x_i \leq 1$ constraint to SeDuMi, it can be rewritten as:

$$x_i \leq 1 \rightarrow x_i + s_i = 1 \qquad (5.9)$$

$$\text{with } s_i \geq 0$$

Thus, enforcing $x_i \leq 1$ requires 10 additional variables, also known as slacks. Modifying the matrices in table 5- 3, the matrices become the following and is represented in table 5- 4:

$$\min [C^T, 0^T] * \begin{bmatrix} x \\ s \end{bmatrix}$$

$$\text{such that: } A_2 * \begin{bmatrix} x \\ s \end{bmatrix} \leq \begin{bmatrix} b_2 \\ I \end{bmatrix} \qquad x \geq 0, s \geq 0$$

$$\text{where } A_2 = \begin{bmatrix} A & 0 \\ I & I \end{bmatrix}$$

$$b_2 = \begin{bmatrix} B \\ I \end{bmatrix} \qquad c_2 = [C^T, 0^T]$$

Having set up the matrices, a straight forward function call to SeDuMi will yield the optimal solution, if such solution exists. Once again, SeDuMi uses the interior point method, which traverses the geometric feasible region of the solution from within, working outwards an edge, as all optimal solution lie on some edge or hyperplane (higher dimensions).

A2 =

(x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10})
1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0
0	0	1	0	1	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1

b2 =

2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1

c2 =

9	3	5	5	4	2	8	1	7	5	0	0	0	0	0	0	0	0	0	0

Table 5 - 4: Constraints in matrix form conforming to SeDuMi input

Input:

```
sedumi(A2, b2, c2)
```

Output:

```
SeDuMi 1.21 by AdvOL, 2005-2008 and Jos F. Sturm, 1998-2003.
Alg = 2: xz-corrector, Adaptive Step-Differentiation, theta = 0.250, beta = 0.500
eqs m = 16, order n = 21, dim = 21, blocks = 1
nnz(A) = 40 + 0, nnz(ADA) = 76, nnz(L) = 48
it :     b*y       gap    delta  rate   t/tP*  t/tD*  feas cg cg  prec
 0 :            5.59E+001 0.000
 1 :  2.25E+001 2.06E+001 0.000 0.3692 0.9000 0.9000   3.76  1  1  1.6E+000
 2 :  2.09E+001 4.52E+000 0.000 0.2191 0.9000 0.9000   1.38  1  1  3.1E-001
 3 :  2.00E+001 1.26E-002 0.000 0.0028 0.9990 0.9990   1.04  1  1
```

```
iter seconds digits       c*x              b*y
 3     0.7   Inf  2.0000000000e+001  2.0000000000e+001
|Ax-b| =  4.5e-016, [Ay-c]_+ =  5.4E-016, |x|= 3.2e+000, |y|= 9.3e+000

Detailed timing (sec)
   Pre          IPM          Post
5.460E-001   7.488E-001   2.496E-001
Max-norms: ||b||=2, ||c|| = 9,
Cholesky |add|=0, |skip| = 0, ||L.L|| = 1.

ans =

   (2,1)       1.0000
   (3,1)       1.0000
   (4,1)       1.0000
   (5,1)       1.0000
   (6,1)       1.0000
   (8,1)       1.0000
  (11,1)       1.0000
  (17,1)       1.0000
  (19,1)       1.0000
  (20,1)       1.0000
```

From the answer variable named "ans" above, the solver outputs the variables and their values, while variables that are not displayed have the value of 0. More specifically, (2,1) correspond to x_2, so the full solution is:

$$x_2 = x_3 = x_4 = x_5 = x_6 = x_8 = s_1 = s_7 = s_9 = s_{10} = 1$$

Referring back to figure 5- 4, the solution (x_i's) computed by SeDuMi does indeed match the edges of the known optimal tour. The x_i's that are not part of the optimal tour have their respective s_i's = 1, which conforms to equation (5.9).

The total distance traveled in the tour that is computed by SeDuMi is 20 units, as illustrated in figure 5- 4. The worst case distance is 33 units, by taking the path shown in figure 5- 5 below. If the average edge weights were taken for all 10 edges, the result is 4.9 units per

edge. Since in a complete tour, 6 edges are traveled, the distance traveled based on the average is 4.9 * 6 = 29.4. This example verifies how the test path generated and solved by the model is much more efficient than that of the flying probe tester's sequential paths. If the sequential order produces a sparse graph, the test path generated by the model will be significantly more efficient. If the sequential order produces a dense graph, the test path generated by the model will not be as efficient, but still more nonetheless. The results are summarized in table 5- 5.

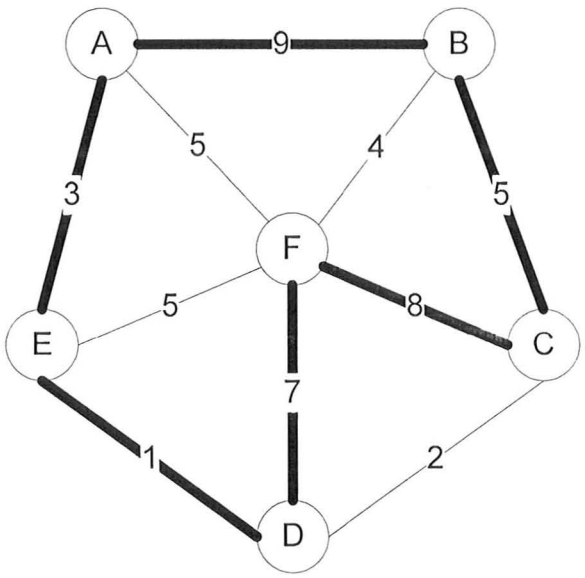


Figure 5 - 5: Worst case travel distance

	Tour Distance (units)
Solution from Model	20
Average Case	29.4
Worst Case	33

Table 5 - 5: Total distance from model's solution, average case and worst case

5.4 Subtour Relaxation of the Fractional 2-Factor:

Now back to the topic of sub-tour. In figure 5.6, a tour is mapped out. The red line splits all the cities into two groups, the ones above and below. Since the tour is indeed a complete tour, there has to be at least 2 paths that cross the red line, joining two groups together. This property holds regardless of any cut (orient the red line) made to the group , since the tour is a complete one, touring each city exactly once. In the right figure of figure 5- 2, if a cut was made between the two triangles, the number of paths crossing each clusters is 0, because sub-tours exist and does not contain all the test-steps.

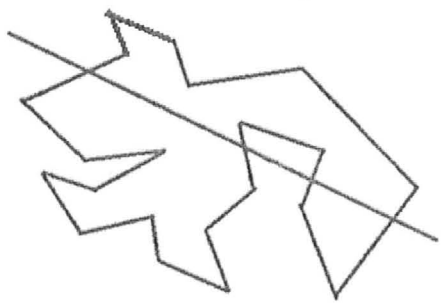


Figure 5 – 6: The points above and below the red line represents two different and unique clusters

Let S be any collection of test points having at least 3 and at most k-1 members (S cannot contain the entire set of test points on the board), and let Q denote the remaining testing that is not part of S. To forbid the sub-tours from forming, a condition needs to be added stating: the sum of the variable corresponding to the path from S to Q must be at least 2. A path from S to Q

involves a point that is part of S, going to another point that is part of Q, crossing the border (cut) that separates the two sets.

$$\sum x(i,j) \geq 2 \quad with\ 1\ of\ i\ and\ j\ is\ in\ S, the\ other\ in\ Q \quad (5.10)$$

The solution quality of this lower bound (obtained from solving the sub-tour relaxation) is much better than the one obtained from the fractional 2-factor LO. However, this greatly increases the computational difficulty of the problem and we can no longer feed the equations to the LO solver to solve directly, as its complexity is $O(2^n)$ [3].

5.5 Graph Representation:

If we record the problem as a graph, G=(V,E), where V is the set of all the points representing each test-steps and E stores all the edge weights w(i,j).

We have following definition:

Let S be a subset of V. An cut, G, that separates S from the rest of V should come intersect edges that have one of its point in S, the other not in S. Let $\delta(v)$ represent these edges. As stated in the previous section, when there exist two sets/clusters of vertices, there has to be at least 2 edges whose starting and ending points from different sets. This property must hold in order to have a complete tour.

We could re-write the fractional 2-factor LO as the following:

Minimize w'x (5.11)

Subject to:

$$x(\delta(v)) = 2 \; for \; all \; the \; vertices \; v \qquad (5.12)$$

$0 \leq x_e \leq 1$, for all edges e (fractional) (5.13)

Equation (5.11) is essentially the same objective function as (5.4), but in a different form. Equation (5.12) is equivalent to (5.5), but modified to represent cuts, where all vertices/points must have an incoming and outgoing edge. Given a non-empty proper subset S of V, the sub-tour inequality for S requires that the variables corresponding to edges joining vertices in S to vertices in V-S sum to at least 2, giving rise to (5.14).

Thus the sub-tour relaxation of the fractional 2-factor LO becomes:

Minimize w'x

Subject to:

$$x(\delta(v)) = 2 \; for \; all \; the \; vertices \; v$$

$$x(\delta(S)) \geq 2 \;\; or \; x(S, V-S) \geq 2 \;\; \text{m} for \; all \; S \subset V, S \neq V, |S| \geq 3 \qquad (5.14)$$

$0 \leq x_e \leq 1$, for all edges e

5.6 Cutting-Plane Method:

Consider the equation (5.13) above, the size of the problem is extremely large. A fundamental idea could be apply in order to reduce the size of the problem. An LO relaxation can be improved during a solution procedure by adding constraints in the form of linear inequalities that are satisfied by all points. This means we will add some constraints during

solving process instead for add all the constraints at the beginning. The flowchart below illustrates the idea behind the algorithm:

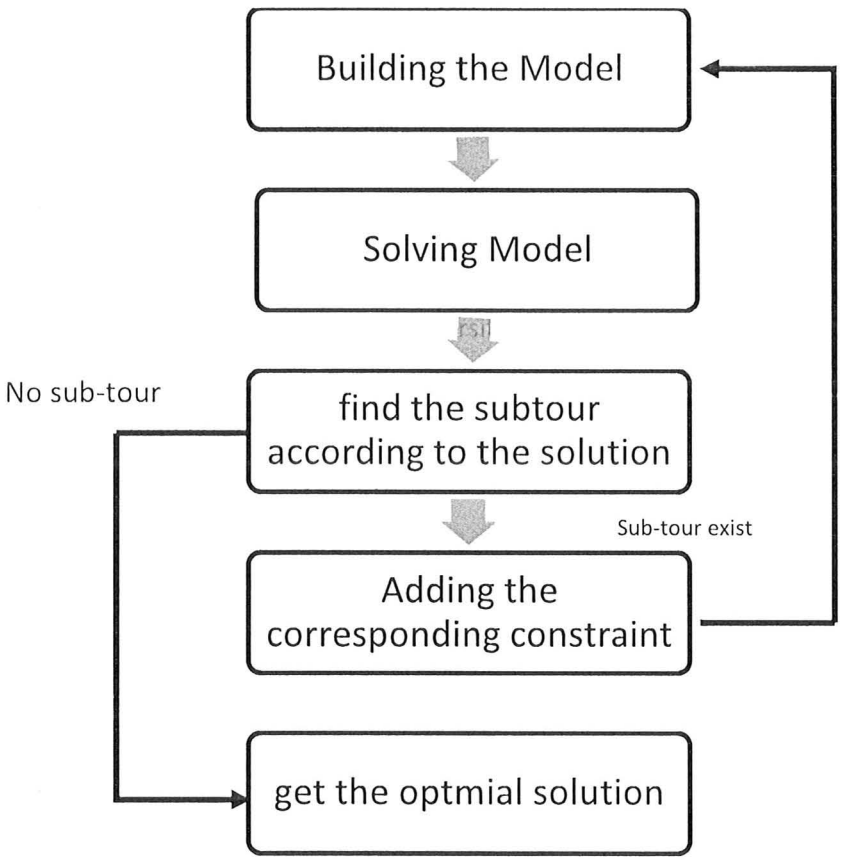


Figure 5 – 7: Idea behind sub-tour elimination algorithm

Algorithm:

1. Build the initial linear system: $x(\delta(v)) = 2\ for\ all\ the\ vertices\ v$
2. Solve the system, returning the solution x*
3. **While** (Subtour exist in x*)
4. FindCut(x*), return a nonempty set
5. **do** rebuild the system, adding constraint based on the suboutrs found
6. Solve system again
7. **end**
8. Return the solution

The algorithm will successfully eliminate all sub-tours, however, it is computationally heavy. Along with section 5.3, the models introduced in this thesis are able to successfully find a tour path that decreases the distance travelled by the probe. Under a mass production setting where many of the same boards are tested, the cost of computing the optimal tour is divided into many circuit boards in the sense that it is computed once but executed multiple times.

Chapter 6: Testing

In this section, the tour distance of Acculogic's sequential path is compared with that of the model developed in the previous chapter. Using CPLEX as the linear optimization solver, 3 additional programs were written in Java. The first program handles inputs (coordinates of test points) and sets up the problem, while the other 2 are used to eliminate any sub-tours that may arise from CPLEX's output. In section 6.1, these 3 programs will be explained in more details, including their purpose, input and output files, as well as their general algorithm.

6.1 Java Programs:

Program: fileData.java

Input File: input.txt (contains coordinate of all points)

Output File: output.lp

Description:

The purpose of this program is to read in all the test points that we wish to find the shortest path of. Points are in the x, y coordinate format and is read by the program and stored as objects. From these objects, the decisional variables (distance from every city to all others) are computed. This program then prints the LO problem to the output file, conforming to the format required by the CPLEX solver.

Algorithm:

1. Read all test points from input.txt
2. From test points, determine the n(n-1)/2 decisional variables (name + edge weights)
3. Print objective function to output file
4. Print constraints to output file
5. Print bounds to output file

Program: Cplex.java

Input file: cplex.txt

cplex2.txt

output.lp

Output file: output.lp

Description:

To relax the model, the bound of each decisional variable allows it to take non-integer values between 0 and 1. Though integer solutions desired, there are cases where the optimal solution would contain non-integer values. If the value is < 0.5, it can be rounded down to 0. If the value is > 0.5, it can be rounded to 1. However, when the value is equal to 0.5, it would be inaccurate to round all of it up, because we would end up having more edges in the tour than it is allowed in a Hamiltonian cycle. This program is to be executed proceeding the use of the

CPLEX solver, after copying its solution from the command prompt to either cplext.txt or cplext2.txt. For the very first use of CPLEX, the solution is placed in the former, while all succeeding solutions are placed in the latter. This program processes the CPLEX solution by checking whether or not any decisional variables in cplex2.txt with the value of 0.5, are also present in cplext.txt (the previous solution). If those new decisional variable with the value of 0.5 do exists in the previous solution, then nothing is to be done. For those decisional variables that do not exist in the previous solution, we must change its corresponding bound in output.txt to 0. For example, if in cplex2.txt, x2_5 = 0.5, x2_5 does not exist in cplex.txt, then in output.lp, we must change the bound of x2_5 to 0 <= x2_5 <= 0.

The idea behind this is that for a decisional variable with a value of 0.5, if that edge is also present in cplex.txt, it is a part of some sub-tour in the previous solution, and sub-tours form because they are the local minimum of that part of the graph. Though joining two separate clusters together do requiring the breaking of edges amongst each cluster, keeping these local minimum (edges existing in the previous solution) is essential to prevent the creation of sub-tours within our sub-tours. It is expected that at least two edges will be broken, an edge from the first cluster that is closest to the second cluster, as well as an edge from the second cluster that is closest to the first. Each of the two loose points of each cluster will form an edge with a separate loose point of the other cluster, thus creating two cluster joining edges. Whenever CPLEX outputs decisional variables with the value of 0.5, they always come in pairs, because together they have the sum of 1, which represent a whole edge. Thus the idea is to keep the edge that is also present in the previous solution (making the edge value = 1) while changing the bounds of the other one that is not part of the previous tour, to 0 (making the edge value = 0). Doing so will prevent the formation sub-tour creating edges within cluster (because bounds are changed to

0, that edge can't exist) as well as ensuring only the edges that are closest to the other cluster are broken.

Algorithm:

1. Read in cplex2.txt
2. if decisional variables in cplex2.txt contain 0.5
3. if decisional variables contains 0.5
4. read cplex.txt, check if cplex.txt also contains it
5. if cplex.txt does not contain it
6. change its bound in output.txt to 0, ie. 0 <= x2_4 <= 0
7. else
8. overwrite cplex.txt with contents of cplex2.txt (newest solution)

Program: subTour.java

Input Files: input.txt

cplex.txt

Output: output.lp

Description:

This program is responsible for analyzing whether the solution solved and output by CPLEX contains any sub-tours. This is to be executed after running Cplex.java if and only if no decisional variables have the value 0.5, which the Cplex.java will confirm. Along with the CPLEX solution, this program also must read the input file that contains all the coordinates. Starting from the first decision variable in the solution (first row of cplex.txt), the program traverses to the next point that it is connected, then onto that's next, and so on. For example, if

the first edge in cplex.txt is x1_15, the program find the next edge that contains test point 15, say x8_15, and finds the next edge that contains test point 8 and so on. If the path that leads back to the first point and contains all the points in the system, then the program outputs the total distance traveled, as a complete tour is established. If any sub-tour exists, then the program will generate a new constraint based on the points in the current sub-tour to all external points. This constraint is updated (written) to the appropriate place in output.lp, to be solved by CPLEX.

Algorithm:

1. Read all test points from input.txt
2. Read CPLEX solution from cplex.txt
3. Using the first decisional variable as starting point, traverse through the connected edges
4. if no sub-tour exist
5. Output to screen the distance traveled
6. else if sub-tour exists
7. Generate new constraint: sum of all edges in the sub-tour to external edges = 2
8. Update constraint to output file

6.2 Steps of Testing:

With a better understanding of what each program does, the following sequence explains the overall testing process:

1. Enter test point coordinates into input.txt
2. Run fileData.java to obtain LO problem, written to output.lp
3. Read output.lp in CPLEX, optimize it and copy solution to cplex.txt
4. Run subTour.java, will update output.lp

If no sub-tour exist, output total distance traveled, go to step 7.

If sub-tour exists, will update output.lp

5. Feed output.lp to CPLEX solver, optimize it and copy solution cplex2.txt
6. Run cplex.java

   If decisional variable = 0.5 exist, go back to step 5

   If decisional variable = 0.5 don't exist, go back to step 4

7. End

6.3 Testing and Results:

The coordinates of Acculogic's demo board (which is used for the testing of the flying scorpion) is entered into input.txt. In this test instance, there are a total of 20 test points. In table 6-1 below the (x, y) coordinates of each point (in cm), the distance from the current point to the next, as well as the sum of these distances (the 20th goes back to the first point, completing the tour). In the input.txt file, the coordinate of each point lies on a separate line, with the x coordinate first, followed by a comma, then the y coordinate.

x	Y		Distance		Sum
4.2799	7.66064				
4.5466	7.6454		0.267135		64.34205
5.0038	7.6454		0.4572		
6.6548	7.6454		1.651		
3.81	6.5786		3.038248		
4.4196	3.5306		3.108362		
4.3942	3.52044		0.027357		
1.76784	0.6604		3.882988		
6.3246	0.6604		4.55676		
0.79756	3.3782		6.159108		
4.6228	0.6604		4.69243		
2.7686	5.2324		4.933684		
8.6106	6.3246		5.94322		

8.9662	5.461		0.933947		
9.271	3.1242		2.356594		
5.207	3.5179		4.083025		
3.937	3.0734		1.345541		
6.6548	1.7653		3.016217		
3.4417	5.04444		4.590944		
3.94716	1.7272		3.355528		
4.2799	7.66064		5.942763		

Table 6 – 1: Test Points Coordinate from Acculogic's demo board (values obtained from Integrator software).

Testing includes running any of the 3 programs and CPLEX at different stages of testing as described in section 6.2. Some programs will be required to execute multiple times, depending on the number of iterations needed to eliminate all sub-tours. This is directly linked to the sample size. For the specifics of testing this instance, please refer to Appendix C, which will illustrate which bounds are changed, constraints are added, as well as the CPLEX solution after each iteration. The final output produced by CPLEX is of the following form:

x1_2	1.000000
x1_5	1.000000
x2_3	1.000000
x3_4	1.000000
x4_13	1.000000
x5_19	1.000000
x6_7	1.000000
x6_16	1.000000
x7_17	1.000000
x8_10	1.000000
x8_17	1.000000
x9_11	1.000000
x9_18	1.000000
x10_12	1.000000
x11_20	1.000000
x12_19	1.000000
x13_14	1.000000
x14_15	1.000000
x15_18	1.000000
x16_20	1.000000

On the left column are the decisional variable names. Their value on right side confirms that they are indeed on the path (1 being part of the tour, 0 being not part of the tour). The solution can also be confirmed by tracing the tour path: 1 → 2 → 3 → 4 → 13 →14 → 15 →18 →9 →11 →20 →16 →6 →7 →17 →8 →10 →12 →19 →5 →1. A Hamiltonian cycle did indeed form as the tour looped back to test point number 1. The number of edges in the tour is 20, which is equal to the number of test points in the test instance. From the subTour.java program, when there are no sub-tours present, it computes the distance of the tour, which is 31.04 cm, a 51.76% reduction of the sequential distance of 64.24 cm.

Chapter 7: Future work

Future work could include incorporating the point exchange feature in the modeling. It will increase computation time, but should reduce the tour distance further as it takes advantage of test points common to different test-steps. Other features of the flying probe tester that can be considered are test-point selection, pre-processing of test points, motion sort and model the vertical z-axis of components as well as probe movements. Also, work can be done in implementing the model on the actual Flying Scorpion and see how it performs.

Appendix A: CPLEX Output

The test program in section 5.3 was also done in CPLEX. Here is the output.

```
CPLEX> set logfile solutionPrimal.txt
Logfile 'cplex.log' closed.
Logfile 'solutionPrimal.txt' open.
CPLEX> opt
No problem exists.
CPLEX> read test.lp
Problem 'test.lp' read.
Read time =    0.01 sec.
CPLEX> opt
Tried aggregator 1 time.
No LP presolve or aggregator reductions.
Presolve time =    0.00 sec.

Iteration log . . .
Iteration:     1    Infeasibility =            3.000000
Switched to devex.
Iteration:     3    Objective     =           20.000000

Primal simplex - Optimal:  Objective =  2.0000000000e+01
Solution time =    0.00 sec.  Iterations = 3 (2)

CPLEX> display solution variables -
Variable Name           Solution Value
x2                            1.000000
x3                            1.000000
x4                            1.000000
x5                            1.000000
x6                            1.000000
x8                            1.000000
All other variables in the range 1-10 are 0.
CPLEX>


------------------------------------------------------------------------------
---
Dual Simplex

------------------------------------------------------------------------------
---
CPLEX> set lp
Present value for method for linear optimization: 1
 0 = automatic
 1 = primal simplex
 2 = dual simplex
 3 = network simplex
 4 = barrier
 5 = sifting
```

```
 6 = concurrent dual, barrier, and primal
New value for method for linear optimization: 2
New value for method for linear optimization: 2
CPLEX> read test.lp
Problem 'test.lp' read.
Read time =    0.01 sec.
CPLEX> opt
Tried aggregator 1 time.
No LP presolve or aggregator reductions.
Presolve time =    0.00 sec.

Iteration log . . .
Iteration:     1   Dual objective     =             8.000000

Dual simplex - Optimal:  Objective =  2.0000000000e+01
Solution time =    0.00 sec.  Iterations = 4 (0)

CPLEX> display solution variables =
Variable '=' does not exist.
Display values of which variable(s): -
Variable Name           Solution Value
x2                            1.000000
x3                            1.000000
x4                            1.000000
x5                            1.000000
x6                            1.000000
x8                            1.000000
All other variables in the range 1-10 are 0.

================================================================================
===
IPM
================================================================================

CPLEX> set lp 4
New value for method for linear optimization: 4
CPLEX> read test.lp
Problem 'test.lp' read.
Read time =    0.01 sec.
CPLEX> opt
Tried aggregator 1 time.
No LP presolve or aggregator reductions.
Presolve time =    0.00 sec.
Parallel mode: using up to 16 threads for barrier.
Number of nonzeros in lower triangle of A*A' = 10
Using Approximate Minimum Degree ordering
Total time for automatic ordering = 0.00 sec.
Summary statistics for Cholesky factor:
  Threads                     = 16
  Rows in Factor              = 6
  Integer space required      = 6
  Total non-zeros in factor   = 21
  Total FP ops to factor      = 91
 Itn      Primal Obj        Dual Obj  Prim Inf Upper Inf  Dual Inf
   0   4.9000000e+01  -4.9000000e+01  8.00e+00  1.00e+01  0.00e+00
   1   2.5927682e+01   1.1222573e+01  1.11e-15  0.00e+00  6.66e-15
   2   2.1408090e+01   1.9362105e+01  6.66e-16  1.39e-16  3.44e-15
```

```
   3   2.0040014e+01   1.9937715e+01  4.44e-16  1.60e-16  3.98e-15
   4   2.0002543e+01   1.9997428e+01  4.44e-16  2.63e-16  4.95e-15
   5   2.0000128e+01   1.9999872e+01  8.88e-16  2.75e-16  4.60e-15
   6   2.0000006e+01   1.9999994e+01  0.00e+00  2.93e-16  2.93e-15
   7   2.0000000e+01   2.0000000e+01  0.00e+00  2.87e-16  2.63e-15
   8   2.0000000e+01   2.0000000e+01  8.88e-16  2.80e-16  4.99e-15
Parallel barrier real time =    0.02 sec.

Primal crossover.
  Primal:  Fixed no variables.
  Dual:  Fixing 6 variables.
        5 DMoves:  Infeasibility  0.00000000e+00  Objective  2.00000000e+01
        0 DMoves:  Infeasibility  0.00000000e+00  Objective  2.00000000e+01
  Dual:  Pushed 5, exchanged 1.
Using devex.
Total crossover time =    0.00 sec.

Total real time on 16 threads =    0.02 sec.

Primal simplex - Optimal:  Objective =  2.0000000000e+01
Solution time =    0.02 sec.  Iterations = 0 (0)

CPLEX> display solution var
Display values of which variable(s): -
Variable Name           Solution Value
x2                            1.000000
x3                            1.000000
x4                            1.000000
x5                            1.000000
x6                            1.000000
x8                            1.000000
All other variables in the range 1-10 are 0.
```

Appendix B: Flying Scorpion's Testing Capabilities

These are the electrical components that the Flying Scorpion is capable of testing:

- Resistors
- Capacitors (need to discharge first)
- Inductors
- Shorts
- Switch: opened or closed circuits (test presence or absence of jumpers)
- Diode: forward and reverse/leakage tests
- Zener Diode
- BJT: NPN, PNP
- MOSFET
- Opto
- Thyristor
- Triac

Appendix C: Testing Procedure and Results

1. Enter test point coordinates into input.txt

Contents of input.txt

4.2799,7.66064
4.5466,7.6454
5.0038,7.6454
6.6548,7.6454
3.81,6.5786
4.4196,3.5306
4.3942,3.52044
1.76784,0.6604
6.3246,0.6604
0.79756,3.3782
4.6228,0.6604
2.7686,5.2324
8.6106,6.3246
8.9662,5.461
9.271,3.1242
5.207,3.5179
3.937,3.0734
6.6548,1.7653
3.4417,5.04444
3.94716,1.7272

2. Running fileData.java → Adds LO problem to output.lp

Contents of output.lp

minimize

0.27 x1_2 + 0.72 x1_3 + 2.37 x1_4 + 1.18 x1_5 + 4.13 x1_6 + 4.14 x1_7 + 7.44 x1_8 + 7.29 x1_9 + 5.52 x1_10 + 7.01 x1_11 + 2.86 x1_12 + 4.53 x1_13 + 5.18 x1_14 + 6.74 x1_15 + 4.25 x1_16 + 4.6 x1_17 + 6.36 x1_18 + 2.75 x1_19 + 5.94 x1_20 + 0.46 x2_3 + 2.11 x2_4 + 1.3 x2_5 + 4.12 x2_6 + 4.13 x2_7 + 7.52 x2_8 + 7.21 x2_9 + 5.68 x2_10 + 6.99 x2_11 + 3.0 x2_12 + 4.27 x2_13 + 4.93 x2_14 + 6.54 x2_15 + 4.18 x2_16 + 4.61 x2_17 + 6.25 x2_18 + 2.83 x2_19 +

5.95 x2_20 + 1.65 x3_4 + 1.6 x3_5 + 4.16 x3_6 + 4.17 x3_7 + 7.7 x3_8 + 7.11 x3_9 + 5.99 x3_10 + 7.0 x3_11 + 3.29 x3_12 + 3.84 x3_13 + 4.52 x3_14 + 6.22 x3_15 + 4.13 x3_16 + 4.69 x3_17 + 6.11 x3_18 + 3.03 x3_19 + 6.01 x3_20 + 3.04 x4_5 + 4.68 x4_6 + 4.7 x4_7 + 8.52 x4_8 + 6.99 x4_9 + 7.25 x4_10 + 7.27 x4_11 + 4.57 x4_12 + 2.36 x4_13 + 3.18 x4_14 + 5.22 x4_15 + 4.37 x4_16 + 5.32 x4_17 + 5.88 x4_18 + 4.13 x4_19 + 6.51 x4_20 + 3.11 x5_6 + 3.11 x5_7 + 6.26 x5_8 + 6.43 x5_9 + 4.4 x5_10 + 5.97 x5_11 + 1.7 x5_12 + 4.81 x5_13 + 5.28 x5_14 + 6.46 x5_15 + 3.36 x5_16 + 3.51 x5_17 + 5.59 x5_18 + 1.58 x5_19 + 4.85 x5_20 + 0.03 x6_7 + 3.91 x6_8 + 3.44 x6_9 + 3.63 x6_10 + 2.88 x6_11 + 2.37 x6_12 + 5.04 x6_13 + 4.94 x6_14 + 4.87 x6_15 + 0.79 x6_16 + 0.66 x6_17 + 2.85 x6_18 + 1.8 x6_19 + 1.86 x6_20 + 3.88 x7_8 + 3.45 x7_9 + 3.6 x7_10 + 2.87 x7_11 + 2.36 x7_12 + 5.06 x7_13 + 4.97 x7_14 + 4.89 x7_15 + 0.81 x7_16 + 0.64 x7_17 + 2.86 x7_18 + 1.8 x7_19 + 1.85 x7_20 + 4.56 x8_9 + 2.89 x8_10 + 2.85 x8_11 + 4.68 x8_12 + 8.88 x8_13 + 8.65 x8_14 + 7.9 x8_15 + 4.47 x8_16 + 3.24 x8_17 + 5.01 x8_18 + 4.69 x8_19 + 2.43 x8_20 + 6.16 x9_10 + 1.7 x9_11 + 5.79 x9_12 + 6.11 x9_13 + 5.48 x9_14 + 3.84 x9_15 + 3.07 x9_16 + 3.39 x9_17 + 1.15 x9_18 + 5.25 x9_19 + 2.61 x9_20 + 4.69 x10_11 + 2.71 x10_12 + 8.35 x10_13 + 8.43 x10_14 + 8.48 x10_15 + 4.41 x10_16 + 3.15 x10_17 + 6.08 x10_18 + 3.13 x10_19 + 3.56 x10_20 + 4.93 x11_12 + 6.93 x11_13 + 6.47 x11_14 + 5.26 x11_15 + 2.92 x11_16 + 2.51 x11_17 + 2.31 x11_18 + 4.54 x11_19 + 1.26 x11_20 + 5.94 x12_13 + 6.2 x12_14 + 6.84 x12_15 + 2.98 x12_16 + 2.45 x12_17 + 5.21 x12_18 + 0.7 x12_19 + 3.7 x12_20 + 0.93 x13_14 + 3.27 x13_15 + 4.41 x13_16 + 5.69 x13_17 + 4.96 x13_18 + 5.33 x13_19 + 6.55 x13_20 + 2.36 x14_15 + 4.23 x14_16 + 5.57 x14_17 + 4.36 x14_18 + 5.54 x14_19 + 6.26 x14_20 + 4.08 x15_16 + 5.33 x15_17 + 2.95 x15_18 + 6.14 x15_19 + 5.5 x15_20 + 1.35 x16_17 + 2.27 x16_18 + 2.33 x16_19 + 2.19 x16_20 + 3.02 x17_18 + 2.03 x17_19 + 1.35 x17_20 + 4.59 x18_19 + 2.71 x18_20 + 3.36 x19_20

subject to

x1_2 + x1_3 + x1_4 + x1_5 + x1_6 + x1_7 + x1_8 + x1_9 + x1_10 + x1_11 + x1_12 + x1_13 + x1_14 + x1_15 + x1_16 + x1_17 + x1_18 + x1_19 + x1_20 = 2

x1_2 + x2_3 + x2_4 + x2_5 + x2_6 + x2_7 + x2_8 + x2_9 + x2_10 + x2_11 + x2_12 + x2_13 + x2_14 + x2_15 + x2_16 + x2_17 + x2_18 + x2_19 + x2_20 = 2

x1_3 + x2_3 + x3_4 + x3_5 + x3_6 + x3_7 + x3_8 + x3_9 + x3_10 + x3_11 + x3_12 + x3_13 + x3_14 + x3_15 + x3_16 + x3_17 + x3_18 + x3_19 + x3_20 = 2

x1_4 + x2_4 + x3_4 + x4_5 + x4_6 + x4_7 + x4_8 + x4_9 + x4_10 + x4_11 + x4_12 + x4_13 + x4_14 + x4_15 + x4_16 + x4_17 + x4_18 + x4_19 + x4_20 = 2

x1_5 + x2_5 + x3_5 + x4_5 + x5_6 + x5_7 + x5_8 + x5_9 + x5_10 + x5_11 + x5_12 + x5_13 + x5_14 + x5_15 + x5_16 + x5_17 + x5_18 + x5_19 + x5_20 = 2

x1_6 + x2_6 + x3_6 + x4_6 + x5_6 + x6_7 + x6_8 + x6_9 + x6_10 + x6_11 + x6_12 + x6_13 + x6_14 + x6_15 + x6_16 + x6_17 + x6_18 + x6_19 + x6_20 = 2

x1_7 + x2_7 + x3_7 + x4_7 + x5_7 + x6_7 + x7_8 + x7_9 + x7_10 + x7_11 + x7_12 + x7_13 + x7_14 + x7_15 + x7_16 + x7_17 + x7_18 + x7_19 + x7_20 = 2

x1_8 + x2_8 + x3_8 + x4_8 + x5_8 + x6_8 + x7_8 + x8_9 + x8_10 + x8_11 + x8_12 + x8_13 + x8_14 + x8_15 + x8_16 + x8_17 + x8_18 + x8_19 + x8_20 = 2

x1_9 + x2_9 + x3_9 + x4_9 + x5_9 + x6_9 + x7_9 + x8_9 + x9_10 + x9_11 + x9_12 + x9_13 + x9_14 + x9_15 + x9_16 + x9_17 + x9_18 + x9_19 + x9_20 = 2

x1_10 + x2_10 + x3_10 + x4_10 + x5_10 + x6_10 + x7_10 + x8_10 + x9_10 + x10_11 + x10_12 + x10_13 + x10_14 + x10_15 + x10_16 + x10_17 + x10_18 + x10_19 + x10_20 = 2

x1_11 + x2_11 + x3_11 + x4_11 + x5_11 + x6_11 + x7_11 + x8_11 + x9_11 + x10_11 + x11_12 + x11_13 + x11_14 + x11_15 + x11_16 + x11_17 + x11_18 + x11_19 + x11_20 = 2

x1_12 + x2_12 + x3_12 + x4_12 + x5_12 + x6_12 + x7_12 + x8_12 + x9_12 + x10_12 + x11_12 + x12_13 + x12_14 + x12_15 + x12_16 + x12_17 + x12_18 + x12_19 + x12_20 = 2

x1_13 + x2_13 + x3_13 + x4_13 + x5_13 + x6_13 + x7_13 + x8_13 + x9_13 + x10_13 + x11_13 + x12_13 + x13_14 + x13_15 + x13_16 + x13_17 + x13_18 + x13_19 + x13_20 = 2

x1_14 + x2_14 + x3_14 + x4_14 + x5_14 + x6_14 + x7_14 + x8_14 + x9_14 + x10_14 + x11_14 + x12_14 + x13_14 + x14_15 + x14_16 + x14_17 + x14_18 + x14_19 + x14_20 = 2

x1_15 + x2_15 + x3_15 + x4_15 + x5_15 + x6_15 + x7_15 + x8_15 + x9_15 + x10_15 + x11_15 + x12_15 + x13_15 + x14_15 + x15_16 + x15_17 + x15_18 + x15_19 + x15_20 = 2

x1_16 + x2_16 + x3_16 + x4_16 + x5_16 + x6_16 + x7_16 + x8_16 + x9_16 + x10_16 + x11_16 + x12_16 + x13_16 + x14_16 + x15_16 + x16_17 + x16_18 + x16_19 + x16_20 = 2

x1_17 + x2_17 + x3_17 + x4_17 + x5_17 + x6_17 + x7_17 + x8_17 + x9_17 + x10_17 + x11_17 + x12_17 + x13_17 + x14_17 + x15_17 + x16_17 + x17_18 + x17_19 + x17_20 = 2

x1_18 + x2_18 + x3_18 + x4_18 + x5_18 + x6_18 + x7_18 + x8_18 + x9_18 + x10_18 + x11_18 + x12_18 + x13_18 + x14_18 + x15_18 + x16_18 + x17_18 + x18_19 + x18_20 = 2

x1_19 + x2_19 + x3_19 + x4_19 + x5_19 + x6_19 + x7_19 + x8_19 + x9_19 + x10_19 + x11_19 + x12_19 + x13_19 + x14_19 + x15_19 + x16_19 + x17_19 + x18_19 + x19_20 = 2

x1_20 + x2_20 + x3_20 + x4_20 + x5_20 + x6_20 + x7_20 + x8_20 + x9_20 + x10_20 + x11_20 + x12_20 + x13_20 + x14_20 + x15_20 + x16_20 + x17_20 + x18_20 + x19_20 = 2

bound
0 <= x1_2 <= 1
0 <= x1_3 <= 1
0 <= x1_4 <= 1

$$0 <= x1\_5 <= 1$$
$$0 <= x1\_6 <= 1$$
$$0 <= x1\_7 <= 1$$
$$0 <= x1\_8 <= 1$$
$$0 <= x1\_9 <= 1$$
$$0 <= x1\_10 <= 1$$
$$0 <= x1\_11 <= 1$$
$$0 <= x1\_12 <= 1$$
$$0 <= x1\_13 <= 1$$
$$0 <= x1\_14 <= 1$$
$$0 <= x1\_15 <= 1$$
$$0 <= x1\_16 <= 1$$
$$0 <= x1\_17 <= 1$$
$$0 <= x1\_18 <= 1$$
$$0 <= x1\_19 <= 1$$
$$0 <= x1\_20 <= 1$$
$$0 <= x2\_3 <= 1$$
$$0 <= x2\_4 <= 1$$
$$0 <= x2\_5 <= 1$$
$$0 <= x2\_6 <= 1$$
$$0 <= x2\_7 <= 1$$
$$0 <= x2\_8 <= 1$$
$$0 <= x2\_9 <= 1$$
$$0 <= x2\_10 <= 1$$
$$0 <= x2\_11 <= 1$$
$$0 <= x2\_12 <= 1$$
$$0 <= x2\_13 <= 1$$
$$0 <= x2\_14 <= 1$$
$$0 <= x2\_15 <= 1$$
$$0 <= x2\_16 <= 1$$
$$0 <= x2\_17 <= 1$$
$$0 <= x2\_18 <= 1$$
$$0 <= x2\_19 <= 1$$
$$0 <= x2\_20 <= 1$$
$$0 <= x3\_4 <= 1$$
$$0 <= x3\_5 <= 1$$
$$0 <= x3\_6 <= 1$$
$$0 <= x3\_7 <= 1$$
$$0 <= x3\_8 <= 1$$
$$0 <= x3\_9 <= 1$$
$$0 <= x3\_10 <= 1$$
$$0 <= x3\_11 <= 1$$
$$0 <= x3\_12 <= 1$$
$$0 <= x3\_13 <= 1$$
$$0 <= x3\_14 <= 1$$
$$0 <= x3\_15 <= 1$$

0 <= x3_16 <= 1
0 <= x3_17 <= 1
0 <= x3_18 <= 1
0 <= x3_19 <= 1
0 <= x3_20 <= 1
0 <= x4_5 <= 1
0 <= x4_6 <= 1
0 <= x4_7 <= 1
0 <= x4_8 <= 1
0 <= x4_9 <= 1
0 <= x4_10 <= 1
0 <= x4_11 <= 1
0 <= x4_12 <= 1
0 <= x4_13 <= 1
0 <= x4_14 <= 1
0 <= x4_15 <= 1
0 <= x4_16 <= 1
0 <= x4_17 <= 1
0 <= x4_18 <= 1
0 <= x4_19 <= 1
0 <= x4_20 <= 1
0 <= x5_6 <= 1
0 <= x5_7 <= 1
0 <= x5_8 <= 1
0 <= x5_9 <= 1
0 <= x5_10 <= 1
0 <= x5_11 <= 1
0 <= x5_12 <= 1
0 <= x5_13 <= 1
0 <= x5_14 <= 1
0 <= x5_15 <= 1
0 <= x5_16 <= 1
0 <= x5_17 <= 1
0 <= x5_18 <= 1
0 <= x5_19 <= 1
0 <= x5_20 <= 1
0 <= x6_7 <= 1
0 <= x6_8 <= 1
0 <= x6_9 <= 1
0 <= x6_10 <= 1
0 <= x6_11 <= 1
0 <= x6_12 <= 1
0 <= x6_13 <= 1
0 <= x6_14 <= 1
0 <= x6_15 <= 1
0 <= x6_16 <= 1

$$0 <= x6\_17 <= 1$$
$$0 <= x6\_18 <= 1$$
$$0 <= x6\_19 <= 1$$
$$0 <= x6\_20 <= 1$$
$$0 <= x7\_8 <= 1$$
$$0 <= x7\_9 <= 1$$
$$0 <= x7\_10 <= 1$$
$$0 <= x7\_11 <= 1$$
$$0 <= x7\_12 <= 1$$
$$0 <= x7\_13 <= 1$$
$$0 <= x7\_14 <= 1$$
$$0 <= x7\_15 <= 1$$
$$0 <= x7\_16 <= 1$$
$$0 <= x7\_17 <= 1$$
$$0 <= x7\_18 <= 1$$
$$0 <= x7\_19 <= 1$$
$$0 <= x7\_20 <= 1$$
$$0 <= x8\_9 <= 1$$
$$0 <= x8\_10 <= 1$$
$$0 <= x8\_11 <= 1$$
$$0 <= x8\_12 <= 1$$
$$0 <= x8\_13 <= 1$$
$$0 <= x8\_14 <= 1$$
$$0 <= x8\_15 <= 1$$
$$0 <= x8\_16 <= 1$$
$$0 <= x8\_17 <= 1$$
$$0 <= x8\_18 <= 1$$
$$0 <= x8\_19 <= 1$$
$$0 <= x8\_20 <= 1$$
$$0 <= x9\_10 <= 1$$
$$0 <= x9\_11 <= 1$$
$$0 <= x9\_12 <= 1$$
$$0 <= x9\_13 <= 1$$
$$0 <= x9\_14 <= 1$$
$$0 <= x9\_15 <= 1$$
$$0 <= x9\_16 <= 1$$
$$0 <= x9\_17 <= 1$$
$$0 <= x9\_18 <= 1$$
$$0 <= x9\_19 <= 1$$
$$0 <= x9\_20 <= 1$$
$$0 <= x10\_11 <= 1$$
$$0 <= x10\_12 <= 1$$
$$0 <= x10\_13 <= 1$$
$$0 <= x10\_14 <= 1$$
$$0 <= x10\_15 <= 1$$
$$0 <= x10\_16 <= 1$$

$$0 <= x10\_17 <= 1$$
$$0 <= x10\_18 <= 1$$
$$0 <= x10\_19 <= 1$$
$$0 <= x10\_20 <= 1$$
$$0 <= x11\_12 <= 1$$
$$0 <= x11\_13 <= 1$$
$$0 <= x11\_14 <= 1$$
$$0 <= x11\_15 <= 1$$
$$0 <= x11\_16 <= 1$$
$$0 <= x11\_17 <= 1$$
$$0 <= x11\_18 <= 1$$
$$0 <= x11\_19 <= 1$$
$$0 <= x11\_20 <= 1$$
$$0 <= x12\_13 <= 1$$
$$0 <= x12\_14 <= 1$$
$$0 <= x12\_15 <= 1$$
$$0 <= x12\_16 <= 1$$
$$0 <= x12\_17 <= 1$$
$$0 <= x12\_18 <= 1$$
$$0 <= x12\_19 <= 1$$
$$0 <= x12\_20 <= 1$$
$$0 <= x13\_14 <= 1$$
$$0 <= x13\_15 <= 1$$
$$0 <= x13\_16 <= 1$$
$$0 <= x13\_17 <= 1$$
$$0 <= x13\_18 <= 1$$
$$0 <= x13\_19 <= 1$$
$$0 <= x13\_20 <= 1$$
$$0 <= x14\_15 <= 1$$
$$0 <= x14\_16 <= 1$$
$$0 <= x14\_17 <= 1$$
$$0 <= x14\_18 <= 1$$
$$0 <= x14\_19 <= 1$$
$$0 <= x14\_20 <= 1$$
$$0 <= x15\_16 <= 1$$
$$0 <= x15\_17 <= 1$$
$$0 <= x15\_18 <= 1$$
$$0 <= x15\_19 <= 1$$
$$0 <= x15\_20 <= 1$$
$$0 <= x16\_17 <= 1$$
$$0 <= x16\_18 <= 1$$
$$0 <= x16\_19 <= 1$$
$$0 <= x16\_20 <= 1$$
$$0 <= x17\_18 <= 1$$
$$0 <= x17\_19 <= 1$$
$$0 <= x17\_20 <= 1$$

0 <= x18_19 <= 1
0 <= x18_20 <= 1
0 <= x19_20 <= 1
end

3. Send output.lp to CPLEX → copy solution to cplex.txt

Contents of cplex.txt

x1_2	1.000000
x1_5	1.000000
x2_3	1.000000
x3_4	1.000000
x4_13	1.000000
x5_19	1.000000
x6_7	1.000000
x6_16	1.000000
x7_17	1.000000
x8_10	1.000000
x8_20	1.000000
x9_11	1.000000
x9_18	1.000000
x10_12	1.000000
x11_20	1.000000
x12_19	1.000000
x13_14	1.000000
x14_15	1.000000
x15_18	1.000000
x16_17	1.000000

4. Execute subTour.java → new constraint is generated and added to output.lp

The newly generated constraint that is added to output.lp

x1_6 + x1_7 + x1_16 + x1_17 + x2_6 + x2_7 + x2_16 + x2_17 + x3_6 + x3_7 + x3_16 + x3_17 + x4_6 + x4_7 + x4_16 + x4_17 + x5_6 + x5_7 + x5_16 + x5_17 + x6_8 + x7_8 + x8_16 + x8_17 + x6_9 + x7_9 + x9_16 + x9_17 + x6_10 + x7_10 + x10_16 + x10_17 + x6_11 + x7_11 + x11_16 + x11_17 + x6_12 + x7_12 + x12_16 + x12_17 + x6_13 + x7_13 + x13_16 + x13_17 + x6_14 + x7_14 + x14_16 + x14_17 + x6_15 + x7_15 + x15_16 + x15_17 + x6_18 + x7_18 + x16_18 + x17_18 + x6_19 + x7_19 + x16_19 + x17_19 + x6_20 + x7_20 + x16_20 + x17_20 = 2

5. Send output.lp to CPLEX → copy solution to cplex2.txt

Contents of cplex2.txt

x1_2	1.000000
x1_5	1.000000
x2_3	1.000000
x3_4	1.000000
x4_13	1.000000
x5_19	1.000000
x6_7	1.000000
x6_16	1.000000
x7_16	1.000000
x8_10	1.000000
x8_17	1.000000
x9_11	1.000000
x9_18	1.000000
x10_12	1.000000
x11_20	1.000000
x12_19	1.000000
x13_14	1.000000
x14_15	1.000000
x15_18	1.000000
x17_20	1.000000

6. Execute Cplex.java →no decisional variables with the value of 0.5 is detected. Thus, cplex.txt is overwritten with contents of cplex2.txt → execute subTour.java → new constraint is generated and added to output.lp

The newly generated constraint that is added to the contents of output.lp

x1_6 + x1_7 + x1_16 + x2_6 + x2_7 + x2_16 + x3_6 + x3_7 + x3_16 + x4_6 + x4_7 + x4_16 + x5_6 + x5_7 + x5_16 + x6_8 + x7_8 + x8_16 + x6_9 + x7_9 + x9_16 + x6_10 + x7_10 + x10_16 + x6_11 + x7_11 + x11_16 + x6_12 + x7_12 + x12_16 + x6_13 + x7_13 + x13_16 + x6_14 + x7_14 + x14_16 + x6_15 + x7_15 + x15_16 + x6_17 + x7_17 + x16_17 + x6_18 + x7_18 + x16_18 + x6_19 + x7_19 + x16_19 + x6_20 + x7_20 + x16_20 = 2

7. Send output.lp to CPLEX → copy solution to cplex2.txt

Contents of cplex2.txt

x1_2	1.000000
x1_5	1.000000
x2_3	1.000000
x3_4	1.000000
x4_13	1.000000
x5_19	1.000000

x6_7	1.000000
x6_16	1.000000
x7_17	1.000000
x8_10	1.000000
x8_11	0.500000
x8_20	0.500000
x9_11	1.000000
x9_18	1.000000
x10_12	1.000000
x11_20	0.500000
x12_19	1.000000
x13_14	1.000000
x14_15	1.000000
x15_16	0.500000
x15_18	0.500000
x16_18	0.500000
x17_20	1.000000

8. Execute Cplex.java → decisional variables with value of 0.5 is detected → change the bounds of those decisional variables that don't also exist in cplex.txt → update bounds of output.lp

Bounds that are changed in output.lp are

0 <= x8_11 <= 1 to 0 <= x8_11 <= 0
0 <= x8_20 <= 1 to 0 <= x8_20 <= 0
0 <= x15_16 <= 1 to 0 <= x15_16 <= 0
0 <= x16_18 <= 1 to 0 <= x16_18 <= 0

9. Send output.lp to CPLEX → copy solution to cplex2.txt

Contents of cplex2.txt

x1_2	1.000000
x1_5	1.000000
x2_3	1.000000
x3_4	1.000000
x4_13	1.000000
x5_19	1.000000
x6_7	1.000000
x6_16	1.000000
x7_17	1.000000
x8_10	1.000000
x8_17	1.000000
x9_11	1.000000

x9_18	1.000000
x10_12	1.000000
x11_20	1.000000
x12_19	1.000000
x13_14	1.000000
x14_15	1.000000
x15_18	1.000000
x16_20	1.000000

10. Execute Cplex.java →no decisional variables with the value of 0.5 is detected. Thus, cplex.txt is overwritten with contents of cplex2.txt → execute subTour.java → no sub-tours exist, the program outputs the distance of the tour

No Sub-Tours Exist
The total cost of the trip is 31.04

Thus, the total tour distance is 31.04 cm, which is much better than the sequential way of testing of 64.34 cm.
This results to a 51.76% reduction in this instance.

Bibliography

[1] Acculogic Inc. (2007) "Integrator Training on the FLS"

[2] Acculogic Inc. (2005) "Training Manual for the Flying Scorpion Test System"

[3] Applegate, D. L., Bixby, R. E., Chvatal, V., Cook, W. J. (2006) "The Traveling Salesman Problem: A Computational Study", Princeton University Press.

[4] Bondy, J. A., Murty, U. S. R. (1982) "Graph Theory with Applications", Elsevier Science Publishing Co., Inc.

[5]Cook, D. The Robot Room. http://www.robotroom.com/SandwichPCB/TestPointLoop.png (figure)

[6] Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C. (2001) "Introduction to Algorithms" (2nd Ed.), McGraw-Hill.

[7] Dantzig, G.B (1963) "Linear Programming and Extensions", Princeton University Press

[8]Dietzel, B. (2009) "Software Documentation: Acculogic Integrator Software Optimizer", Acculogic Inc.

[9] Gutin, G., Punnen, A. (2002) "The Traveling Salesman Problem and its Variations", Kluwer Academic Publishers.

[10] A. J. Hoffman and P. Wolfe (1985), "History" in *The Traveling Salesman Problem*, Lawler, Lenstra, Rinooy Kan and Shmoys, eds., Wiley, 1-16.

[11] Hoffman, K., Padberg M. (1994) "Traveling Salesman Problem", George Mason University, http://iris.gmu.edu/~khoffman/papers/trav_salesman.html

[12] Johnson, D. (2004) "Finding Thévenin Equivalent Circuits", http://cnx.org/content/m0021/latest/

[13]Kern Computers, http://www.kern-computers.com/wp-content/uploads/2009/06/Computer-Parts.jpg (figure)

[14] Kolman, B., Beck, R. (1995) "Elementary Linear Programming with Application" (2nd Ed.), Academic Press

[15] Lu, W. S. (2009) "Use SeDuMi to solve LP, SDP, and SCOP Problems: Remarks and Examples", Department of Electrical Engineering, Univeristy of Victoria

[16] Murty, K. G. (1995) "Operations Research: Deterministic Optimization Models", Prentice-Hall Inc.

[17] Orlin, J. (2007) "Optimization Methods in Management Science", MIT, Sloan School of Mangement, Lecture Series. http://ocw.mit.edu/courses/sloan-school-of-management/15-053-optimization-methods-in-management-science-spring-2007/lecture-notes/ (figures)

[18] Shepertycky G, "Physics Tutorial: Parallel Circuits and Hazard Lights", Physics 24/7, http://www.physics247.com/physics-tutorial/parallel-circuits.shtml (figure)

[19] Sibeyn J. F. (2005) "Algorithms and Datastructures II", Halle (Gemany)University, Institute of Computer Science, Lecture Series. http://users.informatik.uni-halle.de/~jopsi/dinf503/notes_full.shtml

[20] Terlaky, T. (1996) "Interior Point Methods of Mathematical Programming", Dordrecht, Netherlands; Boston : Kluwer

[21] Wright, S.J. (1997) "Primal-Dual Interior-Point Methods," SIAM Publications, Shepertycky

[22] Y. Ye. (1997) "Interior-Point Algorithms: Theory and Analysis," Wiley-Interscience Series in Discrete Mathematics and Optimization.