A TIME BUFFERED ARC ROUTING PROBLEM

A Thesis presented to the Faculty of the Graduate School

University of Missouri-Columbia

In Partial Fulfillment

Of the Requirements for the Degree

Master of Science

By

KURT JAMES EHLERS

Dr. Cerry Klein, Thesis Supervisor

MAY 2015
The undersigned, appointed by the Dean of the Graduate Faculty, have examined the thesis entitled

**A TIME BUFFERED ARC ROUTING PROBLEM**

Presented by Kurt James Ehlers,

A candidate for the degree of Master of Science

And hereby certify that, in their opinion, it is worthy of acceptance.

____________________________________________________________

Dr. Cerry Klein, Thesis Advisor

____________________________________________________________

Dr. James S. Noble, Graduate Advisor

____________________________________________________________

Dr. Ronald McGarvey

____________________________________________________________

Dr. Antonie Stam
ACKNOWLEDGEMENTS

First, and foremost, I would like to most sincerely thank my thesis advisor, Dr. Cerry Klein. Without Dr. Klein’s direction, assistance, and support I never would have been able to complete this work. He never doubted my ability and always encouraged me, even after setbacks and failures. He is also the teacher that first introduced me to and sparked my interest in the field of arc routing and the daunting task of model formulation. He has a wealth of knowledge, is willing to read thesis drafts in the middle of the night, and is an invaluable mentor who I cannot thank enough.

Next, I would like to thank the rest of my committee members: Dr. Jim Noble, Dr. Ronald McGarvey, and Dr. Ton Stam. Dr. Noble has guided me through my entire career at Mizzou; he introduced me to research as an undergraduate, and offered his help and advice whenever I sought it. Without him I would not be where I am today. Dr. McGarvey also was pivotal in the completion of my thesis. His help and guidance through my model formulation and GAMS coding was essential to the completion of my work and he was always willing to help me address my errors, even when it was the seventh time I had come to his office that day. Dr. Stam has been extremely supportive of me, and sparked an idea for me. Going to Flatbranch and hearing his thoughts and eternally positive outlook after a long day’s work was always a high point in my week.

Finally, I would like to thank my friends and family. From my father who first introduced me to Industrial Engineering, to my mother who always pushed me to be better, to my sisters that were always willing to proofread my latest paper, to my roommate that helped me code my heuristic because I always forget that the logic ‘equals’ operator is two equals signs and not one. Thank you all for your support.
# TABLE OF CONTENTS

Acknowledgements ................................................................................................................ ii

List of Figures .......................................................................................................................... v

List of Tables .......................................................................................................................... vi

List of Abbreviations ................................................................................................................. vii

Abstract ................................................................................................................................... viii

Chapter

1. Introduction ................................................................................................................................. 1

2. Literature Review ......................................................................................................................... 6

   Chinese Postman Problem and Variants

   Capacitated Arc Routing Problems and Variants

   Profit Based Routing Problems

   Patrol Routing

   Mixed-Integer Nonlinear Programming

3. Problem Formulation and GAMS Solver ..................................................................................... 13

   Formulation

   Solving a Problem Instance with GAMS

   Results

4. Heuristic Approach ..................................................................................................................... 23

   Background

   Code Description

   Results

   Possible Improvements

5. Future Areas of Research and Conclusion .................................................................................. 35

   Potential Research Opportunities
Conclusion

APPENDIX

1. GAMS CODE ........................................................................................................39
2. GAMS OUTPUT FOR THREE TRAVELS GIVEN ........................................41
3. GAMS OUTPUT FOR FOUR TRAVELS GIVEN ........................................43
4. GAMS OUTPUT FOR FIVE TRAVELS GIVEN ........................................45
5. GAMS OUTPUT FOR EIGHT TRAVELS GIVEN .......................................47
6. GAMS OUTPUT FOR TEN TRAVELS GIVEN ........................................49
7. GAMS OUTPUT FOR TWELVE TRAVELS GIVEN ..................................51
8. CODE FOR MAX WEIGHT – SIX NODE PROBLEM ..................................53
9. CODE FOR MAX WEIGHT MINUS TIME – SIX NODE PROBLEM ..........60
10. CODE FOR MAX WEIGHT – 20 NODE PROBLEM .......................................67
11. OUTPUT FOR MAX WEIGHT – FOUR NODE PROBLEM ..........................74
12. OUTPUT FOR MAX WEIGHT MINUS TIME – FOUR NODE PROBLEM ......75
13. OUTPUT FOR MAX WEIGHT – SIX NODE PROBLEM ................................76
14. OUTPUT FOR MAX WEIGHT MINUS TIME – SIX NODE PROBLEM ........78
15. OUTPUT FOR MAX WEIGHT – 20 NODE PROBLEM .....................................80
16. OUTPUT FOR MAX WEIGHT – 21 NODE PROBLEM .....................................80
17. OUTPUT FOR MAX WEIGHT – 22 NODE PROBLEM .....................................80
18. OUTPUT FOR MAX WEIGHT – 23 NODE PROBLEM .....................................80
REFERENCES ........................................................................................................81
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Figure 1</td>
<td>3</td>
</tr>
<tr>
<td>2. Figure 2</td>
<td>3</td>
</tr>
<tr>
<td>3. Figure 3</td>
<td>19</td>
</tr>
</tbody>
</table>
LIST OF TABLES

Table                                                                 Page
1. Table 1 Title ........................................................................................................22
2. Table 2 Title ........................................................................................................31
LIST OF ABBREVIATIONS

Abbreviation

1. CPP ................................................................. Chinese Postman Problem
2. RPP ................................................................. Rural Postman Problem
3. k-CPP ............................................................... k-Chinese Postman Problem
4. CARP ............................................................... Capacitated Arc Routing Problem
5. VRP ................................................................. Vehicle Routing Problem
6. TSP ................................................................. Traveling Salesman Problem
A TIME BUFFERED ARC ROUTING PROBLEM

Kurt Ehlers

Dr. Cerry Klein, Thesis Advisor

ABSTRACT

Arc routing problems are prevalent in our world today. They are important to society and businesses on a daily basis. The purpose of this work is to consider a problem that has not been addressed in the literature. That problem is how to find the maximal weighted tour of an undirected set of arcs that visits all of the edges within some time constraint, and no edge twice within some time buffer. This problem, similar to most arc and vehicle routing problems, is known to be NP-hard, and thusly can only be solved optimally for the most simplistic problems. Furthermore, being a mixed-integer nonlinear program, most solvers may not be able to guarantee a solution is optimal for even small problem instances. To be able to address realistic problem sets with a large number of vertices, a comparative, route-building heuristic was developed to find a near optimal tour. This heuristic, the mixed-integer nonlinear program solver, and a hand solution were all compared for a relatively simple four node network problem. The program for the mixed-integer nonlinear solver could not solve the problem optimally. A hand solution found for the network was used to show the formulation’s accuracy through the solver program, and was checked for general optimality by the solver. A heuristic was built that was able to find this optimal solution. It is assumed that the heuristic is scalable and hence will also provide a near optimal solution for higher complexity problem instances with a larger number of nodes and arcs.
Chapter 1 – Introduction

The global transportation industry is one of the largest business sectors with total revenues in the trillions of dollars. Arc routing problems are representative of transportation and logistics in the real world and because of this direct relationship optimally solving arc routing problems is potentially a highly lucrative opportunity. Transportation companies that have implemented arc routing optimization tools reduce costs between 5% and 30% on average (Hasle, 2007). For a multi-trillion dollar industry, even the low end of that spectrum represents a substantial amount of dollars saved clearing the 100 billion dollar mark. This is why research in the area of arc routing is highly popular and the pool of problems relating to the field is pretty saturated.

Arc routing and vehicle routing problems are among the most thoroughly researched combinatorial optimization problems in the field of operations research. These research areas are primarily concerned with logistics and route optimization, and are applicable to most industries in one way or another. The primary goal of almost all arc routing problems is to minimize the cost associated with the overall path that meets the requirements set by the problem instance. Each variant of requirements related to a different type of problem, and each type of problem relates to at least one real world application. There are instances like the traveling salesman problem, the basis for all arc routing problems, which seeks to visit every node once with the minimal travel path, and also vehicle routing problems that aim to visit every arc, or a required subset of arcs, in the fastest way possible. These are just two of the many examples of arc routing problems, but just these two have numerous applications in the real world. The traveling
salesman problem can be used to model a number of things like bus routing, the flow of current on a circuit board, or even job shop scheduling. The vehicle routing problem can be utilized to generate routes for delivery of goods by companies like UPS, pick-up of items like waste collection, or even methods for coding in some languages.

The problem presented and discussed in this paper is based on the idea of a smaller city parking attendant that wants to visit the more regularly parked-on streets more often than the smaller, less parked-on streets. The goal is to find the overall path that generates the highest amount of meters checked and/or violations given. This requires that each arc be given both a weight corresponding to the potential value of traversing that arc and a time value corresponding to how long it takes to traverse and service that arc. This setup is most relatable to routing problems with profits that are most commonly seen in conjunction with the traveling salesman problem. The figures on the next page represent an example of how the real world city structure would translate to a nodal network. Figure 1 is a sample section of the downtown of a small city, and Figure 2 is the nodal version of that section.
For this parking attendant methodology, there is no constraint on how often an arc can be traversed within the maximal weighted tour. Without further constraints on the problem the maximal tour would be one traveling back-and-forth on a single high-weighted arc with a lower travel time. In a realistic environment, this would not be optimal because the likelihood of finding violators so quickly after checking a street is small. In addition to this, it is desired to service every arc in the edge set at least once. This prevents the attendant from ignoring any streets and avoids guaranteeing zero violators for that street both for that day’s tour and all future days’ tours. There are many ways to approach this issue, one being to introduce a time buffer into the problem to prevent an arc being traveled again for a set time frame after the last time it was traveled in the tour set. This is the method used in this paper, while other, more stochastic possibilities are presented in the “Potential Further Research” chapter of this paper.

This paper focuses on a time-capacitated, single vehicle instance similar to a traveling salesman problem. Multi-vehicle instances are not considered since this is the first attempt at a time buffered problem. While the idea behind this problem is based on parking attendant routing, the application will be more valuable long-term in other areas like maintenance checks on manufacturing lines, making rounds in a hospital wing, or other various inspection related activities. Furthermore, given a successful approach to the problem with a single vehicle approach, this research can be expounded upon further to develop a multi-vehicle approach, and can be tested with actual data.

All of the different variants for the multitude of arc routing problems have significance to the real world, and ideas for new problem instances are not common. This is a driving factor behind the relevance of this work, and increases the confidence
that there are many potential applications, even beyond the ones discussed, for a solution to this problem as it is researched further. Even if the unorthodox method of maximizing the objective function does not serve many applications, the time buffer constraints found and proposed could have a multitude of uses in other problem formulations. Almost any time a new idea is proposed in a heavily researched field is a success and the additional potential applications for subparts of this problem, like the buffer constraint, only make this work more useful.
Chapter 2 – Literature Review

There are numerous problems and problem variants in the routing field of operations research. Because of this fact, in order to confirm the novelty of this problem a thorough investigation into this area was conducted. The different problems found in this area will be briefly presented in this section and while there may be some commonalities with many of the problems and ideas, none of the problem variants that were found and studied fully encompassed the thought behind the problem presented in this paper. For an overview of the general problems in this area, the book “Network Routing, volume 8 of Handbooks in Operations Research and Management Science” by Ball et al. is recommended.

Chinese Postman Problem and Variants

The first problem researched was the Chinese postman problem (CPP). This deals with attempting to find a minimal length tour that travels each edge in some graph G(V, E) at least once, and was initially proposed by Kwan Mei-Ko, a Chinese mathematician, who got the idea from his previous work as a postman during the Chinese Cultural Revolution (Eiselt “Part I”, 1995). This problem aims to minimize a tour based on the total travel time to span every edge in the network, and the optimal solution would be to cover each edge once and only once. This in itself is inherently different from the maximal weighted tour methodology in the new problem being presented. The CPP has many different variants as well.

The rural postman problem (RPP) is the most common derivation of the CPP, and only requires the spanning tour to traverse a subset R ⊆ A of arcs, and considers the
others edges within the graph \( G(V, E) \) to be unnecessary (Eiselt “Part 2”, 1995). This model is much more practical because, in general, most real life scenarios do not require servicing all of the arcs in a network. It also provides a basis for many more sub-problems as well. The windy rural postman problem is an important problem in arc routing, and occurs when for the graph \( G(V, E) \), the edges of the undirected graph have two different non-negative weights associated with them, \( c_{ij} \) and \( c_{ji} \). These two variables are costs for traveling down each side of the arc \( e(i, j) \) in opposite directions, and the graph then becomes a ‘windy’ graph still under the same constructs as the rural postman problem (Benavent, 2009). The street sweeper problem aims to complete a RPP, but is subject to constraints on when the streets can be serviced due to restrictions like parking regulations (Bodin 1978). Another sub-problem is garbage collection routing. This deviation is focused on generating routes for all the days in a week corresponding to each day’s pick-up region, often through utilizing time windows like was done by Kim, Kim, and Sahoo in 2006. There are several other sub-problems similar to these two like mail delivery routing, bus routing, utility meter reading, and the stacker crane problem, but none of them differ too greatly from one another (Eiselt “Part II”, 1995).

The CPP also has variants where there is more than one postman; these are called k-Chinese postman problems (k-CPP). For many of the previously discussed variants, there are also k-CPPs to match. Osterhues and Mariak discuss many of these alternatives in their 2005 paper. Some of the ones they detailed are: the minimum-maximum k-CPP where the goal is to minimize the cost of the postman with the maximum tour cost, the minimum absolute deviance k-CPP which attempts to minimize the sum of all deviations from a set time, the minimum square deviation k-CPP that instead minimizes the sum of
the deviations squared, and the minimum overtime k-CPP that minimizes the overtime for all postmen. Furthermore, in a 2007 paper, Tomaž Kramberger and Janez Žerovnik step through 17 variants of the CPP, many of which have already been discussed, before narrowing in on their proposed variant of a priority constrained CPP that seeks to visit all the edges in a set, and certain “priority” nodes at the earliest possible time with priority hierarchical from priority node $v_1$ to $v_2$ to $v_k$.

**Capacitated Arc Routing Problems and Variants**

A large part of real-world problems is the idea of capacity. This limiting factor is tackled in the heavily researched area of capacitated arc routing problems (CARP). The basic foundation of the CARP is for the graph $G(V, E)$, each edge in $E$ has associated with it some cost $c_{ij}$ and some demand $d_{ij}$ that must be satisfied through servicing performed by a vehicle that has a capacity $T$. The academic CARP is solely concerned with minimizing the total duration of the tour while satisfying all of the demand (Lacomme 2003). There are many derivations off of this general problem like the periodic CARP, the stochastic CARP, the bi-objective or multi-objective CARP, time-window CARP, and others. The periodic CARP varies from the traditional CARP by extending the single working period, or a day, to a “planning period” consisting of several days of work. In this problem, each task, $t$, has to be serviced some $f_t$ times over the planning period, but can only be serviced once a day by a single vehicle (Kansou, 2009). The stochastic CARP is simply the traditional, deterministic CARP with the change of the demand variables $d_{ij}$ being random variables (Fleury, 2002). The bi-objective and multi-objective CARPs both seek to optimize the problem for multiple
values. The bi-objective CARP most often aims to minimize the total duration of all the trips, and the length of the longest trip. This is because of applications like waste collection, similar to the minimum-maximum k-CPP, that want all of their vehicles to finish by a certain time and at least relatively close to the same time (Lacomme 2003). Other papers strive to optimize for more than two values, like Grandinetti et al. in their 2003 paper that looks to minimize the total route cost, the length of the longest route (makespan), and the number of vehicles utilized. The time-window CARP constrains the problem by allowing certain nodes to only be available during set time windows. These problems typically break down the nodes into sets of stages like depots, facilities, and customers to better assimilate to actual instances (Çetinkaya, 2013). All of these different variations on that original problem are an indicator of the usefulness of the CARP and how, like the similar CPP, it can have numerous real-world applications in many different areas.

**Profit Based Routing Problems**

There exists a category of problems that involve routing while considering profits. In this category are vehicle routing problems/arc routing problems and traveling salesman problems. Vehicle routing problems (VRP) are a component of most CARP instances and along with the CARP have many sub-problems and variants and are often capacitated, as well (Federgruen, 1995). The primary idea behind the vehicle routing problem with profits is that given a constrained problem where not all nodes can be visited, what nodes should be visited in what order. The solution is to assign the nodes “profit” values dependent on the importance of those nodes (Archetti, 2014). From this problem, a heuristic approach is necessary because mixed-integer nonlinear programs
have been proven to be NP-hard (Bellare, 1992; Kannan, 1978; Murty, 1987). Some of heuristics used to attack this problem are the branch and price algorithm and the local neighborhood search heuristic (Archetti, 2010). The same principal of nodes with “profits” assigned to them applies throughout all of the profit based routing problems. The traveling salesman problem (TSP) is a routing problem concerned with a single entity, or salesman, attempting to visit every node in a graph once by traveling the shortest path possible (Powell, 1995). This problem is the foundation for all arc routing problems and is therefore heavily researched. The applications for it are numerous which directly relates to the vast amount of variants on the problem. It often serves as the problem used to introduce arc routing in education due to its importance and relatively simple setup and methodology for the most basic version. “The traveling salesman problem: a computational study” by Applegate et al. (2011) is an excellent reference to illustrate the importance, vastness, and numerous applications of this problem, as well. This difference between this problem and the CPP is that the TSP only needs to visit every node, while the CPP requires the entity travel every arc or edge. Feillet, Dejax, and Gendreau (2005) illustrate the similarities between the vehicle routing problem/arc routing problem with profits and the traveling salesman problem with profits. This methodology is similar to the concept behind the problem presented in this paper, and is an important model.

Patrol Routing

The last routing instance examined is policeman patrol routing. There are two main types of policeman patrols, ones that are concerned with crime prevention and response and ones concerned with preventing and responding to traffic accidents.
The key characteristics of policeman patrol routing are randomness, hitting “hot spots”, irregularity of patrol rates, and maximal covering (Rosenshine, 1970; Curtin, 2010). These features are necessary in order to reduce criminals’ ability to predict policemen’s locations, monitor areas that have been deemed more likely to be home to criminal activity or reckless driving, reduce criminals’ ability to predict when policemen will be on patrol, and have patrolmen spread out strategically so that the response time to any one place within the region is minimized.

After thoroughly examining all of these routing topic areas, the problem proposed in this paper appears to contain novel concepts that have not yet been fully researched. The idea of maximizing a weighted tour is partially covered in previously researched routing problem with profits, but that is primarily for choosing which nodes or arcs to visit over others when capacity is not enough. This problem aims to visit all nodes and all arcs in a graph, and uses the weights to decide which arcs may be more profitable to continue to visit. Unlike capacitated problems, no arc possesses a capacity for being traveled, and the only constraining factor is an overall travel time constraint. The most unique factor of this problem is the concept of a buffer between repeat travels on an arc. No research was found with any sort of buffer in place in a routing problem of any type, and this is pivotal in the approach to addressing the proposed problem.

**Mixed-Integer Nonlinear Programming**

Mixed-integer nonlinear programs are “optimization problems where some of the variables are constrained to take integer values and the objective function and feasible region of the problem are described by nonlinear functions (Bonami, 2012).” This field
is considered to be relatively new, and the majority of problems and methods are significantly less understood and stable than linear mixed-integer programs (Hemmecke, 2009). The complexity of these problems generates difficulty in optimally solving them using current computing capabilities. Computers can only store numbers of finite accuracy, and unlike linear programs, the number of digits needed for the length of nonlinear optimal solutions cannot be known beforehand (Hochbaum, 2007). This means that heuristic solutions are important approaches to these problem instances. Many approaches have been researched in recent years including: diving heuristics, feasibility pump, RINS, local search, branch-and-cut, outer-approximation, and many other algorithms (Bonami, 2008; Bonami, 2010). Many of these algorithms are based in route-building and bounding to find feasible solutions, which is a similar method to the heuristic approach in this work. Mixed-integer nonlinear programs are extremely complex problems with most existing solver software being unable to find optimal solutions for these problems (Ambrosio, 2010).
Chapter 3 – Problem Formulation and GAMS Solver

Problem Description

The arc routing problem considered is quite different from the typical problems in the literature. The objective is almost exclusively and minimization function for the arc routing field, but this problem uses a maximization function to find the tour with the highest weighted value. Continuing the differences from the typical arc routing problem, the arcs in this case are free to be traveled any number of times by the optimal tour and are assumed to have no capacity restrictions, but are desired to not be traveled multiple times with some time period of the arc’s previous traversal. This requires the use of a time buffer constraint to keep the entity from traveling these arcs for some set time. This is highly unusual for an arc routing problem because the typical goal is simply to visit each node or arc once or as few times as possible to satisfy its demand. The general differences from the previous problems in the literature, along with the additional constraint, made this problem much more complex because of the lack of examples relating well and completely to the problem furthering the relevance of this work.

Formulation

The problem of finding the maximal weighted tour that visits every edge within the network is relatively straightforward, but the addition of a buffer constraint complicates the formulation significantly. The method of implementing this constraint can be seen in the formulation below and is explained later in this section.

For an undirected graph G (V, E):
Max $\sum_k \sum_v \sum_{(i,j) \in E} w_{ij} * X_{ijkv}$

s.t.

(2) $\sum_k \sum_v \sum_{(i,j) \in E} t_{ij} * X_{ijkv} \leq T$

(3) $\sum_k \sum_i X_{ijkv} - \sum_k \sum_i X_{ijkv} = 0 \quad \forall j, \forall v$

(4) $\sum_j X_{ijkv} = 1 \quad i = 0, k = 1, v = 1$

(5) $\sum_v \sum_i X_{ijkv} = 1 \quad j = 0, k = 1$

(6) $\sum_j X_{ijkv} = 0 \quad i = 0, k > 1, v > 1$

(7) $\sum_v \sum_i X_{ijkv} = 1 \quad j = 0, k > 1$

(8) $X_{ijkv} (U_{i(v-1)} + t_{ij} - R_{ijkv}) = 0 \quad \forall (i,j) \in E, \forall (j,l) \in E, \forall k, v > 1$

(9) $X_{ijkv} (R_{ijkv} - U_{jv}) = 0 \quad \forall (i,j) \in E, \forall (j,l) \in E, \forall k, \forall v$

(10) $R_{ijkv} - t_{ij} - R_{ij(k-1)e} - P \geq 0 \quad \forall i \in V, \forall j \in V, k > 1, \forall v, e < v$

(11) $R_{ijkv} - t_{ij} - R_{jilq} - P \geq 0 \quad \forall i \in V, \forall j \in V, \forall k, \forall q, v > 2, e < v$

(12) $\sum_k \sum_v X_{ijkv} + X_{jikv} \geq 1 \quad i > 0, j > 0$

(13) $X_{ijkv} \leq \sum_{v-1}^v X_{ij(k-1)v} \quad \forall (i,j) \in E, k \geq 2, \forall v$

(14) $\sum_v X_{ijkv} \leq 1 \quad \forall (i,j) \in E, \forall k$

(15) $\sum_k \sum_{(i,j) \in E} X_{ijkv} = 1 \quad \forall v$

(16) $X_{ijkv}$ is binary \quad $\forall (i,j) \in E, \forall k, \forall v$

(17) $R_{ijkv}$ is integer \quad $\forall i \in V, \forall j \in V, \forall k, \forall v$

(18) $U_{iv}$ is integer \quad $\forall i \in V, \forall v$

Where:

$X_{ijkv}$ is the decision variable for traversing edge $ij$ for the $k^{th}$ time for the $v^{th}$ overall pass
$X_{ijke}$ is the decision variable for traversing edge $ij$ for the $k^{th}$ time for the $e^{th}$ overall pass, where $e < v$

$w_{ij}$ is the weight of edge $ij$

$R_{ijke}$ is the arrival time to node $j$ for travel from node $i$ to node $j$ the $k^{th}$ time

$R_{ijkv}$ is the arrival time to node $j$ for travel from node $i$ to node $j$ the $q^{th}$ time

$U_{iv}$ is the arrival time to node $i$ for the $v^{th}$ time

$U_{jv}$ is the arrival time to node $j$ for the $v^{th}$ time

$t_{ij}$ is the travel time for edge $ij$

$T$ is the time constraint for total allotted time

$P$ is the time buffer between transversals

Some node 0 acts as a dummy node that travels to the starting point and is traveled to from the ending point

The first equation is the objective function that aims to maximize the total weight of the tour. The reason for this is because the weight represents some form of importance or profit associated with the arcs and the server’s goal is to maximize this value. The first constraint, equation 2, is the total time constraint placed on the problem that keeps the sum of the time values for each arc multiplied by the decision variables under the user defined time limit. This sum is done over the range of all of the decision variables so that every travel is accounted for since the decision variable $X_{ijke}$ is set as one whenever arc $(i, j)$ is traveled for the $k^{th}$ time and $v^{th}$ overall pass. The second constraint is managing the flow into and out of nodes. The first summation is adding all of the decision variable values for travels into some node $j$, and then the second summation is subtracted from this value for all travels out of that node $j$. The difference between these two summations
is set as zero to force every subsequent travel to come out of the ending node of the previous travel, and is done for every value of \( j \).

The third through sixth constraints are all concerned with managing the flow into and out of the dummy node. The dummy node is necessary in the formulation because the flow constraint previously discussed is done for every node, and since the vehicle in this problem is not depot based, the starting and ending nodes are different. This requires that a dummy node be in the simulation to act as a theoretical depot in order to satisfy the flow constraint. All of the arcs into and out of the node are given a weight and travel time of zero so as not to affect the rest of the formulation. The third through sixth constraints aim to restrict travels into and out of the dummy node to solely the first travel to the starting node and the final travel from the ending node. The third and fifth constraints are concerned with the travels out of the dummy node, while the fourth and sixth deal with the travels into the node. Constraints three and four force there to be one traversal into and out of the dummy node, and constraints five and six do not allow any other travels to or from the node.

The seventh, eighth, ninth, and tenth constraints are all concerned with buffering the travels along an arc between \( k-1 \) and \( k \) traversals. The first two of this constraint subset are relating the variables \( R_{ijkv} \) and \( U_{iw} \) to the objective function through the decision variable \( X_{ijkv} \). The last two constraints are placing the actual buffer \( P \) between the two travels of the arc. The idea behind these constraints was based on part of the formulations presented by Lystlund and Wøhlk (2012). Constraint seven, equation eight, sets the arrival time into node \( j \) from node \( i \) equal to the arrival time into node \( i \) for the
previous traversals $v$ value. The next equation is simply setting the time that the entity leaves node $j$ for the $v$th travel equal to the time that the entity entered node $j$ from node $i$. Both of these equations are multiplied by the decision variable $X_{ijkv}$ to ensure that the only time these variables are affected is when the arc $ij$ is being traveled for the $k$th time and the $v$th overall traversal in the tour. The next equation, constraint nine, provides the time buffer by stating that the time of arrival into $j$ for arc $ij$ being traveled for the $k$th time and the $v$th overall traversal in the tour minus the time it took, $t_{ij}$, to travel the arc minus the time it was traveled the previous time, $k-1$ and $e$ is less than $v$, minus the user defined buffer value $P$ must be greater than or equal to zero. The $t_{ij}$ is in the equation because the time starting the travel needs to be $P$ amount of time later then the end of the arc’s previous travel time. So if an arc was traveled at time 40, this constraint will not allow it to be traveled again until time $40 + P$. This is done only when $k$ is greater than one because there is no value for $R$ when $k$ is zero. This problem is undirected though, meaning that the arc needs to be considered as one arc for both the $ij$ and $ji$ directions of travels. Since constraint nine only sets a buffer for the forward flowing travels, an additional constraint, equation eleven, was needed to prevent travels for that same $P$ time period in the $ji$ direction. The difference in this constraint is that the previous travel being checked is in the $ji$ direction some unknown $q$th time where $e$ is less than $v$. These constraints are also the sole reason that a $v$ value was needed in the decision variable so that travels before and after whatever travel $i$ to $j$ is being considered can be identified.

Constraint eleven assures that every arc is traveled at least once in the overall tour, and can be traversed in either direction. The undirected part of the graph is accounted for by allowing either $X_{ijk}$ or $X_{jik}$ be one and satisfy the constraint. Next, the
twelfth constraint makes it impossible to travel an arc for the $k^{th}$ if it was not traveled the $k-1^{th}$ time. The thirteenth and fourteenth constraints ensure that a singular value is given for each $k$ for an (i, j) combination and each v. This ensures that if an arc is traveled twice that the $k$ value is not one both times, and that for each $v$ value there must be a travel somewhere. If the $v$ values in the solver are more than needed, the dummy constraints can be relaxed to allow for the tour to travel through the dummy to satisfy all of the $v$ values. The final three constraints define the types of variables $X_{ijk}$, $T_{ijk}$, and $T_{jq}$ as binary, integer, and integer, respectively.

Solving a Problem Instance with GAMS

The formulation of this problem shows that it is a mixed-integer nonlinear program that is NP-hard and can possibly have instances that are not convex. These two factors illustrate that it is a highly complicated problem and the use of a solver would be necessary in order to find an optimal or near optimal solution, but can only be done for the simplest of problems. The CPLEX solver was used through GAMSIDE programming software. A simple four node problem was solved using this software, and even for an instance as simple as a four node, twelve arc network the solver could not find a feasible solution on its own. The GAMS coded program was run through successive trials that provided some of the optimal tour’s steps to the program as givens. The GAMS code can be found on the first two pages of the Appendix, the model statistics can be found in the Appendix on pages 39 through 50, and the four node network solved for is illustrated in Figure 3 below. The four node network was also
solved by hand for a total weight of 67 and a total time of 60, this solution is what was given to the GAMS program in pieces.

Figure 3

Stepping through the GAMS code, the goal of this program is to set up the problem directly like the formulation. The first step done was to define and set the variables. This was a three part process. First, i, k, and v were defined and given values. Second, the alias function was used to make j, q, and e equivalent to i, k, and v, respectively. Finally, the binary decision variable $X_{ij kv}$ was defined. Next in the process, the times and weights associated with each arc were input in two tables. The first row and column for the two tables are both all zeros because these are the arcs into and out of the dummy node, except for the travel from zero to one (the first travel in the tour) which is set equal to the buffer to satisfy constraint ten. Then, the constant values for the total time allotment and buffer size are defined as scalars. The time allotment is set at 75 instead of 60, because the tenth constraint forces the $R_{ij kv}$ to be equal to the buffer making the tour start at time fifteen instead of zero. The free variable seen next is objective function variable that changes as the program is run. The following lines set the travels to be made to find the optimal path because the solver could not handle the
number of equations and variables resulting from the constraints. Following this, the positive variables $R_{ijkv}$ and $U_{iv}$ are defined. After this the most important part of the code is written, defining and writing the equations. The first half is naming the equations and providing a brief description of them. The variables seen in the parentheses behind each equation name are over which variables the equation acts. This is equivalent to the parameters set on the right side of the formulation previously presented. The second half is where the actual equations are written out in the GAMS code language. The equations seen are ordered the same as the formulation, starting with the objective function. The $\$ symbol seen represents “such that” in the code, and the “ord” operators seen are finding the values in the set seen in the parentheses following the operator. For example, the equation being called as “flow(j,v)$\$(ord(v)<12)\$” translates to “for equation ‘flow’ for all values of j and v, such that the values of v are the set’s values before the twelfth value in the set.” In this case, the code is being told to stop at the twelfth value because that is the last value in the set and the equation contains a value $v+1$ that could cause infeasibility. Beyond that, the code should be a direct translation of the formulation into the GAMS language. The last four lines seen are an option and the procedural calls to run the software. The option “LIMROW” is setting how many lines of output in the constraints section to show, and this is done to allow for further investigation into the results to check that the code is being run correctly. The “Model” function names the code, and the “Solve” function solves the model using whatever solver is chosen. The solver used on this model is the mixed-integer nonlinear program solver maximizing $z$, with CPLEX being the backbone of the solver. Finally, the “Display” command initiates the results of the program to be shown.
Results

The formulation solved through the GAMS program was too complex, and GAMS could not solve it with all of the constraints in the problem because the mixed-integer nonlinear solvers have significant limitations. After about an hour and a half the program quit and returned that no feasible solution could be found. Another approach was necessary, so the model was run six times, with the first run providing the entire path to the program by fixing all twelve of the x values in the tour found by the hand solution. The GAMS solver returned that this was a feasible solution given the formulation provided. Next, the first ten travels were provided to the model and it was able to find the last two. Then, the first 8 traverses were set. This run provided the correct optimal solution and a feasible path with the same arcs traveled, just in a slightly different order. A similar result was returned when the given path was stopped at the fifth v value, same optimal solution with a different arc ordering. When the provided values were reduced to only the first three x values, the program stopped operating correctly, and returned a solution that was infeasible and significantly less than the optimal solution. The reason it was significantly less than the optimal was because it reached the limit of travels restricted by the available number values in the “v” set due to making unallowable travels to the dummy node and from a node into itself. This implies that the model’s limit is either reached when only the first three travels or first four travels are provided. To confirm this, the first four travels were given to the model and it successfully returned the optimal solution similar to the previous runs. The results of these runs can be seen in Table 1 on the next page. This indicates that while the GAMS program cannot run on its own, the formulation can be assumed to be correct.
<table>
<thead>
<tr>
<th>GAMS Model Run</th>
<th>Number of Arcs Set in Model</th>
<th>Path</th>
<th>Total Weight</th>
<th>Total Time (for GAMS 60 = 75)</th>
<th>Program Run Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hand Solution</td>
<td>N/A</td>
<td>0-1-4-2-3-4-1-2-3-1-4-3-0</td>
<td>67</td>
<td>60</td>
<td>About 600</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>0-1-4-2-3-3-4-1-0-2-2-1-3-0</td>
<td>46</td>
<td>58</td>
<td>1271.460</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>0-1-4-2-3-4-1-3-4-1-2-3-0</td>
<td>67</td>
<td>75</td>
<td>386.508</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>0-1-4-2-3-4-1-3-2-1-4-3-0</td>
<td>67</td>
<td>75</td>
<td>417.407</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>0-1-4-2-3-4-1-2-3-4-1-3-0</td>
<td>67</td>
<td>75</td>
<td>495.055</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>0-1-4-2-3-4-1-2-3-1-4-3-0</td>
<td>67</td>
<td>75</td>
<td>518.463</td>
</tr>
<tr>
<td>6</td>
<td>12</td>
<td>0-1-4-2-3-4-1-2-3-1-4-3-0</td>
<td>67</td>
<td>75</td>
<td>7.166</td>
</tr>
</tbody>
</table>

Table 1
Chapter 4 – Heuristic Approach

Background

Because this problem is NP-hard, a heuristic approach is necessary. The heuristic built is, according to Fisher (1995), a relatively simplistic route building, improvement based heuristic. The heuristic was coded in the language C utilizing the babbage server, Filezilla software, and the Putty application. The heuristic was used to solve a four node problem, a six node problem, and a 20 node problem. The four node problem solutions were compared between the CPLEX solver and the heuristic, and the heuristic was shown to also provide the optimal solution.

The heuristic built is route building in nature and looks to find the most advantageous single step forward from the path’s current location. Many different types of heuristics were considered to approach this problem, like simulated annealing and genetic algorithms, but these were deemed overly complex for the initial problem. The heuristic chosen provides a good method for finding near optimal paths, and is both scalable and easily manipulated. To determine the best possible single move, all of the possible short-term outcomes for making a travel to that node were compared to the possible short-term outcomes for making a travel to each of the other nodes. This means that if the best possible, or highest weighted, short-term outcome is predicated on first traveling to node \( j \), then that is the next travel to be made in the tour. The short-term outcome set is then cleared and the process is repeated from this new current node \( j \). The short-term outcomes chosen for the version of the heuristic presented are the sums of arcs’ weights for every possible five travel combination from the current node. Two different methods of summing were tested as well, summing the difference of the arcs’
weight minus their time and a flat summation of solely the arcs’ weight. The idea behind using these two different approaches is that if the goal is to have the highest weighted tour and the stopping criteria is once you reach a set time limit the entity can no longer travel, then a tour would be deemed better than another tour if its total weight is better than the other tour’s or, if the time constraints are held constant, the tour’s weight minus the time used is better than the other tour’s. This also make logical sense when considering being in the middle of an overall tour. If one tour is at total weight \( W \) and total time \( T \), and the other tour is also at total weight \( W \) but at some total time \( E < T \), this second tour must be in a better state as it has more time left than the first and may be able to include more total travels than the first.

These two techniques, summing the weight and summing the weight minus the time, were compared for both a four node network and a six node network to see if one approach was better than the other. After confirming which of the two types of heuristic approaches was better, a 20 node problem was tested. The code for the weight sum maximizing heuristic and weight minus time maximizing heuristic for the six node problem and the 20 node problem heuristic can all be found in the Appendix. All of the code for these problems is set up and written about the same. The “Maximizing Weight for a Six Node Problem” will serve as the example for the general heuristic’s code setup and process, and can be found on pages 51 through 57 in the Appendix.

The heuristic is still constrained according to the problem formulation; with the only major change being the decision variable \( X_{ij\kappa} \) no longer being binary because the \( k \) value is held in the value of the \( X_{ij} \) variable, and it no longer needs a \( \nu \) value because the
heuristic is a user created program and code runs by making sequential steps and also provides a great deal of flexibility in defining variables that are able to be greatly manipulated. The buffer constraint also becomes a variable that tracks the time the arc was traveled and updates after each traversal. These variables are updated permanently when an official step is made, and temporarily when finding and comparing all the feasible short-term paths from the current node. The general algorithm for the heuristic is relatively simple.

1. Define parameters
2. Set current node as initial node
3. Run loop
   a. From the current node find an initial five step path
   b. Set this initial path equal to the best temporary path
      I. Find next possible five step path and compare to best temporary path
      II. If the path is better than the best temporary path, reset to this path
      III. Repeat for all possible paths
   c. Set temporary path to be the current path
   d. Make travel from the first node in the current path to the second node
   e. Increase values of decision variable and time traveled for this arc
   f. Increase weight by the arc’s weight value for overall path
   g. Increase time by the arc’s time value for overall path
   h. Set new current node
   i. Add current node to the overall path
j. If time plus the minimum feasible arc time out of the current node is greater than the time constraint go to step 4

I. Else go to step a

4. Display overall path, total weight, and total time

**Code Description**

The first thing done in the code is declaring variables and functions. The two functions declared can be seen at the end of the code and are responsible taking the temporary step forward, and the “findLastNode” function additionally sets the temporary path as the current path if its sum is higher. The constant variables are defined next. These consist of the “NODE_SIZE” value that is the total number of nodes in the problem, the “TRAVEL_DIS” value that is how many travels forward the heuristic is going to go, the “BUFFER_SIZE” is how long the buffer time between allowable travels of the same arc is, the “TIME_CONST” value is the maximum amount of time the tour is allowed to be, and the “W_INC” value is the amount the theoretical weight of an arc is increased temporarily if it is still untraveled after some amount of time. Lastly, the global variables are defined. The node set is defined for six nodes from zero to five, not one to six, for simplicity because in coding the index of a set starts at zero not one. Next, the X, Xtemp, and Z sets are defined as empty square sets of dimension equal to the NODE_SIZE. The purpose of these three sets are to hold the values of the number of passes along an arc for the overall tour, the temporary number of passes within the nested for loops, and the time of the last travel for each arc, respectively. Then the weight and time value sets are defined for all the arcs. The values were chosen randomly for every problem set, with the 20 node set be generated by the Excel random number generator.
with values ranging from four to twelve. Finally, the variables responsible for all the tracking within the function are defined. The “currentNode” is the node that the overall tour is at, the “totalWeight” is the overall weight of the tour, the “totalTime” is the overall time of the tour, the “tempNode” set is the set of nodes that are constantly being manipulated in the code to find the feasible temporary paths, the “loopnum” variable simply tracks which for loop the code is in, the “currentPathSum” is the value of the highest weighted temporary tour found at any time, the “tempPathSum” is the weight of the temporary path that will be compared to the currentPathSum once it has reached its furthest point, the “tempTotalTime” tracks the time that the tour would be at while on the temporary tour for checking the feasibly of arc traversals, the “tempPathNodes” is the set of nodes comprising the temporary path being considered, and finally, the “currentPathNodes” is the set of nodes in the best path considered at any point in time.

Next, the main function is begun. First, variables used randomly throughout the code, for keeping counts within loops and other sundry tasks, are defined. In the weight minus time heuristic there are additional variables defined for the purpose of making sure the tempPathSum is reset after every feasible path beyond any one node is checked. The “printf” actions that follow are to display the parameters set for this problem in the output. The "while loop that constrains the problem to stay within the time allotment then begins. The count variable then seen to be increased in simply for the purpose of tracking the total number of travels in case the user wanted to know, in which case this number could be called to be displayed at the end of the code. Then, the temporary path is set to start at the current node, and the current path’s sum is reset to zero for the next round of comparisons.
The core part of the heuristic then begins as the first of the five for loops is called. Being the first of the loops, the temporary total time and first temporary node are set to the overall tour’s current values. The loopnum value is set to zero since it is the first loop, and the temporary path’s sum is reset to zero. The three if statements after this are for determining feasibility. The first if statement restricts any node from traveling to itself, even though this should happen anyways it is an effective backup measure, and could be substituted to make sure that only connected nodes are feasible routes to be taken if the network considered did not have all nodes within it connected (as would often be the case in real-world scenarios). The next if statement makes sure that the arc considered is either being traveled for the first time or that the previous time it was traveled was longer ago than the size of the buffer time. The last if statement confirms that the arc considered, if traveled, would not put the overall tour of the time limit. If any of these if statements are not met (are true), the “continue” function jumps the loop to the next node to be considered. If all of the if statements are met (are false) then the code continues and the temporary X node value is set equal to the overall X node value for consistency purposes. Then the extra variables “l” and “p” are used to hold the values of the temporary node and its “Z” value for later reference. The next if statement provides added weight, by W_INC, after half of the allotted time is up in order to make untraveled arcs more appealing so that all arcs are traveled at least once. This is also repeated in the second for loop to make sure all possible untraveled arcs are considered after the halfway mark has occurred. The time this is done can also be reduced depending on by what time the user wants to have all arcs traveled. Next, the “findNextTempNode” function is called. This function adds the arc’s weight to the temporary sum, the arc’s time to the
temporary total time, increases the arc’s X value by one, for travels in both directions since they are undirected arcs, sets the arc’s Z value equal to the temporary total time, also for travels in both directions, and sets the next loops starting node, or the value of the next temporary node, to the considered arc’s ending node. This is where the weight minus time approach differs by summing that difference instead of just the weight of the arc.

The next three for loops follow the same general procedure as the first, and the fifth, and final, loop is the same except it calls the function “findLastNode” instead. This function has the added responsibility of comparing the temporary path to the current path, and setting it as the new current path if its weighted sum is better. After this function is performed, the temporary path sum is reduced by the weight that was added onto it so that when it comes back through the loop again for the next node it is reset to its original value. After exiting this loop and entering back into the fourth loop, a similar assessment procedure to the “findLastNode” is done for possibly setting the current path to the temporary path. This is only done when the tour is nearing the overall time constraint and a fifth travel cannot be performed. After this, the temporary path sum, temporary total time, temporary X, and Z values are all reduced back to their previous settings, just as was done in the fifth loop. This is the same for the ends of the remaining loops, with the ends of the first and second loops also potentially decreasing the temporary path sum by the W_INC if necessary. For the weight minus time approach, the same weight minus time value is subtracted instead of just the weight of the arc.

After the code has gone through all the for loops, the current path is set to display in the results, along with the current total weight and the current total time, and the X and
Z values of the arc that was chosen to be traveled. The overall values for all of these items displayed are also set. Then, given the new current node, the minimum travel distance of all possible arcs to each other node is found. Finally, this minimum value is added to the total time, and if this combined value is greater than the total time allowed then the code breaks from the while loop. Whenever this does eventually happen, the end of the function displays the final total weight and time of the optimal, or near optimal, tour and also displays if any arcs were not traveled. If arcs were not traveled it is almost certainly a matter of better setting the parameters or reducing the time before the program adds weight to the yet untraveled arcs. The 20 node network program’s display is different from the others because its optimal tour consisted of 657 travels. So, instead of displaying every X value and step values, only the overall path, total weight, and total time were displayed.

**Results**

The results outputs of all the problems run can be found on pages 72 through 78 of the Appendix, and a summary of all the approaches used can be seen in Table 2 on the next page.
<table>
<thead>
<tr>
<th>Heuristic Number</th>
<th>Maximizing Sum of</th>
<th>Number of Nodes</th>
<th>Total Weight</th>
<th>Total Time</th>
<th>Program Run Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hand Solution</td>
<td>Weight</td>
<td>4</td>
<td>67</td>
<td>60</td>
<td>About 600</td>
</tr>
<tr>
<td>1</td>
<td>Weight</td>
<td>4</td>
<td>67</td>
<td>60</td>
<td>.000181</td>
</tr>
<tr>
<td>2</td>
<td>Weight – Time</td>
<td>4</td>
<td>60</td>
<td>58</td>
<td>.000271</td>
</tr>
<tr>
<td>3</td>
<td>Weight</td>
<td>6</td>
<td>168</td>
<td>158</td>
<td>.001334</td>
</tr>
<tr>
<td>4</td>
<td>Weight – Time</td>
<td>6</td>
<td>165</td>
<td>157</td>
<td>.001468</td>
</tr>
<tr>
<td>5</td>
<td>Weight</td>
<td>20</td>
<td>5545</td>
<td>5000</td>
<td>4.887014</td>
</tr>
<tr>
<td>6</td>
<td>Weight</td>
<td>21</td>
<td>9048</td>
<td>7998</td>
<td>12.942527</td>
</tr>
<tr>
<td>7</td>
<td>Weight</td>
<td>22</td>
<td>9165</td>
<td>8000</td>
<td>18.958577</td>
</tr>
<tr>
<td>8</td>
<td>Weight</td>
<td>23</td>
<td>9543</td>
<td>8000</td>
<td>29.323686</td>
</tr>
</tbody>
</table>

Table 2

The four node outputs were the first completed, and show several things. They demonstrate that the code works according to its intent and design, and is displaying the items and values requested. The maximizing weight approach also gave what was found to be the optimal solution to the problem, as well. The results also gave the first inclination that the maximum weighted tour design is superior to the weight minus time design as it outperformed the other a weight of seven and was able to use the maximum amount of allotted time. It also took less time to run the program, but the difference, like the total time, is just fractions of milliseconds. The problem was then increased in complexity to a six node problem, and again the two techniques were compared. The maximal weighted summation technique still out performed the weight minus time technique, but only by a weight of three this time and only took advantage of one additional time unit, and the difference in the time to run the program was about negligible, again. However, this time difference could become more significant as the problem complexity increases. Due to the performance in the two smaller problems, the
maximum weighted tour method was chosen as the optimal heuristic, and was performed on the 20 node problem. The heuristic solved the problem in less than five seconds, and returned what appears to be a desirable result, and did not miss any node traversals. The program was then performed on 21, 22, and 23 node problems to get an idea of how much the time to solve the program increases as complexity increases. The times seen show that increasing complexity increases the run time, but not exponentially which is the goal of any heuristic. This complexity increasing was also to see if the program had limits. The limit was found to be 24 nodes, but because of memory and not timing out.

The constraints for this heuristic are restrictive and prohibit one or more nodes from being considered at each for loop step. How many nodes that are being eliminated from consideration is impossible to know for each loop iteration. Because of this, for determining the complexity of the heuristic in relation to scaling, the worst case scenario is what should be considered to find the bound for the heuristic complexity. For this methodology the worst case scenario considers all possible travels whether they are feasible or not. If every possible travel is considered then the number of steps to find each individual travel is \( N^6 \), where \( N \) is the number of nodes in the problem. This is the polynomial function because each for loop will be considering potential travel to every other node connected to it in the network, and in this problem instance each node is connected to every other node. Because the polynomial function is only for finding the next travel to make, the total number of steps will be the polynomial \( N^6 \) multiplied by the number of travels in the optimal tour. The worst case scenario for the number of travels in the optimal tour, or the largest the number could be, is the time constraint divided by the minimal arc travel time. This means that if the number of nodes is \( N \), the
length of the checked path (or number of for loops) is \( L \), the time constraint for the problem is \( T \), and the minimal arc travel time is \( t'_{ij} \), then the maximum number of steps possible is:

\[
(N^L) * \left( \frac{T}{t'_{ij}} \right)
\]

As the number of nodes increases this essentially becomes a heuristic bounded by the polynomial \( N^L \).

**Possible Improvements**

Many potential enhancements could be made to this code. The first change that could be made would be to make the code more recursive. If done well, this would allow for the capability to simply change the values of global variables at the top of the code that would affect change throughout the program. For example, how far the temporary path travels before comparing against the other possible paths could be changed by increasing the number of nested for loops used, and if coded properly this could be as simple as increasing a global variable that sets the number of for loops to be run. Another improvement would be to make the if statements that break the code out of the loops to also account for networks where not every node is connected to each other. The simplest way to change this, as previously mentioned, would be by changing the if statement that currently checks if node \( i \) is equal to node \( j \) to instead eliminate any travel from \( i \) to \( j \) with a where the arc’s weight is zero. Not having the ability to travel to any node from the current position could also increase the likelihood of getting stuck in subtours, but this problem could also be handled relatively easily. By adding the weight
increase value to untraveled arcs in all of the for loops, eventually the heuristic will reach the untraveled arc and insert it into the current path. If this does not work then the user would just need to increase the number of loops and the travel distance of the temporary path. The code could also be used with realistic data and compared with the current real world results. This testing could provide an idea of how much the heuristic improves the current method, whatever it may be. Finally, higher level coding methods could be used to make the heuristic more efficient. This would give the heuristic the capacity to handle much bigger problems. Currently, the code reaches memory limitations at a 24 node problem with a total time constraint of 8000, even though it reaches this point in around 30 seconds. Since the networks considered have each node connected to every other node in the network, a 24 node problem corresponds to 506 arcs in the network, a significantly sized problem that would require significant memory allocation.
Chapter 5 – Future Areas of Research and Conclusion

Potential Research Opportunities

There are many avenues to further the ideas presented in this paper. A highly relevant one would be to add a stochastic element to the problem. There are many ways to do this, but the most useful would probably be the transformation of arcs weights into stochastic variables that adjust based on data gathered from the traversal. The real world equivalent using the parking attendant idea would be: given the parking attendant traveled down a street checking its meters, and knows how much time was left on each meter; what should the new demand for that street be and how long should the attendant wait to travel it again. Or after each traversal, have the demand go to zero, and increase to the maximum at a rate set by the information gathered from the travel along that arc. Other parts of the problem could be made stochastic as well, like the times associated with each arc, but these have been researched previously (Kenyon, 2003).

A relatively simple extension would be to consider this problem for a directed graph. This could be more realistic depending on the application this methodology is being applied to. For a parking attendant this is likely more applicable, but for other applications like hospital rounds and quality assessment purposed the undirected graph is probably more accurate. Applying previously researched topics like the Windy postman problem could be very interesting extensions and would provide many research opportunities.

Another area for further research could be to add a randomization factor. This would likely be similar to the randomization techniques used in the policeman patrol
routing, and done for similar reasons. By adding a randomization factor into the methodology, the general randomness inherent to day to day changes is accounted for, and also, depending on the use of the approach, other parties are not privy to route chosen and traveled daily. A relatively simple way of approaching this would be to do something similar to the potential stochastic changes previously discussed, with the added complexity of something like the periodic capacitated arc routing problem presented by Kansou and Yassine in 2009. The demand for an arc could be treated as a random variable that is assigned at the beginning of each new period, almost assuring that a different optimal tour would be found.

The third potential research possibility considered in this paper is taking other heuristic approaches to solve the problem, and comparing them to the heuristic presented in this paper to see if they outperform it. Many general heuristics exist that could be altered to fit this kind of problem, and potentially could provide much better solutions. At the very least they would provide a measuring stick to assess the functionality of this route building heuristic approach. The most likely choice would probably be the use of a genetic algorithm to find an optimal path. This is a moderately simple heuristic that could be developed in a reasonable amount of time.

Lastly considered, using the heuristic for real world applications could be an interesting research topic. While parking attendant routing was the original idea behind this problem, it is not a field that is highly relevant now or in the future and other areas should be considered instead. Some potential uses could be in hospitals for doctors and nurses making their rounds, distributing repetitive medication, or addressing regularly occurring problems. Another application could be for quality assurance purposes. Given
a network consisting of different areas producing parts with certain tolerances or likelihoods of being in defect, which ones should you visit more often, how often should you visit them, and in what order should you check all of the areas? This problem could be addressed by the heuristic presented in this paper as well. Maintenance is a similar problem that can also be addressed by this paper’s solution. In a manufacturing facility, there are many lines and machines, and each one is likely to have a different probability of malfunction or breaking down than the others. The similarity in routing would be in which lines or machines should you visit more often than others, and what is the best route for maintenance checks to take in order to satisfy each arc’ requirements. This same problem is also present for supervisors wishing to oversee certain workers or areas more. One of the final areas that could apply this approach is delivery and pickup of product, whether it is in-house or outbound logistics. If there are multiple locations with more or less demand than it would be a likely application. Almost any network with variable demand and areas requiring more passes than others would be a worthwhile application of the methodology presented in this paper.

Conclusion

This paper presented a new problem instance in arc routing involving a time buffered maximal weighted tour. This problem could be applicable to many different real world scenarios, and has great potential for further research. Being an NP-hard problem, optimal solvers are not applicable beyond the simplest of problem instances, so a heuristic approach would be necessary. This paper also presented a feasible heuristic that produced good results, and could handle larger problems without much trouble. It performed optimally on a small four node problem, and can be assumed to perform at
least near optimality on larger problems. Given these results, this paper successfully presented a new idea in to the field of operations research and logistics, and provided a feasible solution to the problem, as well.
Appendix

GAMS code:

Sets
i nodes /0, 1, 2, 3, 4/  
k travels /1, 2/  
v passes /1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12/;

alias(i,j);  
alias(k,q);  
alias(v,e);  
binary variable x(i,j,k,v) equals 1 if arc is traveled for kth time;

Table w(i,j) time to go down arc
     0  1  2  3  4
0 0 0 0 0 0  
1 0 0 5 6 9  
2 0 5 0 6 5  
3 0 6 6 0 6  
4 0 9 5 6 0;

Table t(i,j) weight to go down arc
     0  1  2  3  4
0 0 15 0 0 0  
1 0 0 4 8 7  
2 0 4 0 5 7  
3 0 8 5 0 5  
4 0 7 7 5 0;

Scalar P time buffer on next visit to same arc /15/;  
Scalar Y Time constraint for model /75/;  
Free Variable z objective function value;
  x.fx('0','1','1','1') = 1;  
  x.fx('1','4','1','2') = 1;  
  x.fx('4','2','1','3') = 1;  
  x.fx('2','3','1','4') = 1;  
  x.fx('3','4','1','5') = 1;  
  x.fx('4','1','1','6') = 1;  
  x.fx('1','2','1','7') = 1;  
  x.fx('2','3','2','8') = 1;  
  x.fx('3','1','1','9') = 1;  
  x.fx('1','4','2','10') = 1;  
  x.fx('4','3','1','11') = 1;  
  x.fx('3','0','1','12') = 1;
Positive Variables
\( r(i,j,k,v) \)  
departure time from \( i \) to \( j \) \( k \)th time
\( u(i,v) \) 
time at which the \( v \)th overall pass arrives at node \( i \);

Equations
\( \text{obj} \) 
define objective function
\( \text{time} \) 
time traveled
\( \text{flow}(j,v) \) 
flow constraint
\( \text{dum1}(i,k,v) \) 
dummy equation 1
\( \text{dum2}(j,k) \) 
dummy equation 2
\( \text{dum3}(i,k,v) \) 
dummy equation 3
\( \text{dum4}(j,k) \) 
dummy equation 4
\( \text{buff1}(i,j,k,v) \) 
buffer equation 1
\( \text{buff2}(i,j,k,v) \) 
buffer equation 2
\( \text{buffer1}(i,j,k,v,e) \) 
buffering equation 1
\( \text{buffer2}(i,j,k,q,v,e) \) 
buffering equation 2
\( \text{visit}(i,j) \) 
make sure all arcs traveled at least once
\( \text{prev}(i,j,k,v) \) 
make sure arc traveled \( i \) before \( i+1 \)
\( \text{onek}(i,j,k) \) 
only one value for \( k \) for each arc
\( \text{onev}(v) \) 
only one value for all \( v \);

\[
\text{obj..} \quad z = \sum(k, \sum(v, \sum((i,j), w(i,j)*x(i,j,k,v)))),
\]
\[
\text{time..} \quad \sum(k, \sum(v, \sum((i,j), t(i,j)*x(i,j,k,v)))) = Y;
\]
\[
\text{flow}(j,v)$$(\text{ord}(v)<12)$. \quad \sum(k, \sum(i, \sum((i,j), x(i,j,k,v)))) - \sum(k, \sum(i, \sum((j,i), x(j,i,k,v)))) =e= 0;
\]
\[
\text{dum1}(i,k,v)$$(\text{ord}(i)=1 \text{} \text{and} \text{} \text{ord}(k)=1 \text{} \text{and} \text{} \text{ord}(v)=1)$. \quad \sum(j, \sum(x(i,j,k,v))) =e= 1;
\]
\[
\text{dum2}(j,k)$$(\text{ord}(j)=1 \text{} \text{and} \text{} \text{ord}(k)=1)$$. \quad \sum(v, \sum(i, \sum(x(i,j,k,v)))) =e= 1;
\]
\[
\text{dum3}(i,k,v)$$(\text{ord}(i)=1 \text{} \text{and} \text{} \text{ord}(k)>1 \text{} \text{and} \text{} \text{ord}(v)>1)$$. \quad \sum(j, \sum(x(i,j,k,v))) =e= 0;
\]
\[
\text{dum4}(j,k)$$(\text{ord}(j)=1 \text{} \text{and} \text{} \text{ord}(k)>1)$$. \quad \sum(v, \sum(i, \sum(x(i,j,k,v)))) =e= 0;
\]
\[
\text{buff1}(i,j,k,v)$$(\text{ord}(v)>1)$. \quad x(i,j,k,v) * (u(i,v-1) + t(i,j) - r(i,j,k,v)) =e= 0;
\]
\[
\text{buff2}(i,j,k,v)$. \quad x(i,j,k,v) * (r(i,j,k,v) - u(j,v)) =e= 0;
\]
\[
\text{buffer1}(i,j,k,v,e)$$(\text{ord}(e)<\text{ord}(v) \text{} \text{and} \text{} \text{ord}(k)>1)$. \quad x(i,j,k,v) * (r(i,j,k,v) - t(i,j) - r(i,j,k-1,e) - P) =e= 0;
\]
\[
\text{buffer2}(i,j,k,q,v,e)$$(\text{ord}(e)<\text{ord}(v) \text{} \text{and} \text{} \text{ord}(v)>2)$. \quad x(i,j,k,v) * (r(i,j,k,v) - t(i,j) - r(j,i,q,e) - P) =e= 0;
\]
\[
\text{visit}(i,j)$$(\text{ord}(i)<>\text{ord}(j) \text{} \text{and} \text{} \text{ord}(i)>1 \text{} \text{and} \text{} \text{ord}(j)>1)$$. \quad \sum(k, \sum(v, \sum((i,j), x(i,j,k,v) + x(j,i,k,v)))) =e= 1;
\]
\[
\text{prev}(i,j,k,v)$$(\text{ord}(k)>1)$. \quad x(i,j,k,v) =l= \sum(e$(\text{ord}(e)<\text{ord}(v)), x(i,j,k-1,e));
\]
\[
\text{onek}(i,j,k)$. \quad \sum(v, \sum(x(i,j,k,v))) =l= 1;
\]
\[
\text{onev}(v)$. \quad \sum(k, \sum((i,j), x(i,j,k,v))) =e= 1;
\]

Option LIMROW = 30;
option reslim=4000;
Model Thesismodel /all/;
Solve Thesismodel using minlp maximizing z;
Display x.l;
GAMS Output for Three Travels Given:
MODEL STATISTICS
 BLOCKS OF EQUATIONS 15  SINGLE EQUATIONS 9,745  
 BLOCKS OF VARIABLES 4  SINGLE VARIABLES 1,261  
 NON ZERO ELEMENTS 33,507  NON LINEAR N-Z 27,900  
 DERIVATIVE POOL 10  CONSTANT POOL 20  
 CODE LENGTH 64,236  DISCRETE VARIABLES 597  
 GENERATION TIME = 0.094 SECONDS 8 MB 24.3.3 r48116 WEX-WEI  
 EXECUTION TIME = 0.094 SECONDS 8 MB 24.3.3 r48116 WEX-WEI  

GAMS 24.3.3 r48116 Released Sep 19, 2014 WEX-WEI x86 64bit/MS Windows  
04/22/15 10:04:17 Page 5  
General Algebraic Modeling System  
Solution Report  SOLVE Thesismodel Using MINLP From line 78  
SOLVE SUMMARY  
MODEL Thesismodel  OBJECTIVE z  
TYPE MINLP  DIRECTION MAXIMIZE  
SOLVER ANTIGONE  FROM LINE 78  
**** SOLVER STATUS 1 Normal Completion  
**** MODEL STATUS 2 Locally Optimal  
**** OBJECTIVE VALUE 46.0000  
RESOURCE USAGE, LIMIT 1271.460 40000.000  
ITERATION COUNT, LIMIT 4 2000000000  
EVALUATION ERRORS 0 0  
ANTIGONE 24.3.3 r48116 Released Sep 19, 2014 WEX-WEI x86 64bit/MS Windows  
---------------------------------------------------------------------  
ANTIGONE: Algorithms for coNTinuous/Integer Global Optimization; Version 1.1  
Ruth Misener and Christodoulos A. Floudas  
Computer-Aided Systems Laboratory (CASL)  
Department of Chemical & Biological Engineering; Princeton University  
---------------------------------------------------------------------  
Termination Status : Found feasible point; guessed variable bounds  
Best Feasible Point: +4.600000e+001  
Best Possible Point: +6.700000e+001  
Relative Gap: +4.565217e-001  
CONOPT 3 24.3.3 r48116 Released Sep 19, 2014 WEX x86 64bit/MS Windows  
CONOPT 3 version 3.16C  
Copyright (C) ARKI Consulting and Development A/S  
Bagsvaerdvej 246 A  
DK-2880 Bagsvaerd, Denmark  
** Warning **  The number of nonlinear derivatives equal to zero  
in the initial point is large (= 55 percent).  
A better initial point will probably help the  
optimization.  
** Warning **  The variance of the derivatives in the initial
point is large (= 4.0 ). A better initial point, a better scaling, or better bounds on the variables will probably help the optimization.

** Optimal solution. Reduced gradient less than tolerance.**

** CONOPT time Total 0.071 seconds**

of which: Function evaluations 0.015 = 21.1%

1st Derivative evaluations 0.016 = 22.5%

** LOWER LEVEL UPPER MARGINAL**

---- EQU obj . . . 1.000

---- EQU time -INF 58.000 75.000 .

---- VAR x equals 1 if arc is traveled for kth time

** LOWER LEVEL UPPER MARGINAL**

0.1.1.1 1.000 1.000 1.000 EPS
0.2.1.9 . 1.000 1.000 EPS
1.0.1.8 . 1.000 1.000 EPS
1.3.1.12 . 1.000 1.000 6.000
1.4.1.2 1.000 1.000 1.000 9.000
2.1.1.11 . 1.000 1.000 5.000
2.2.1.10 . 1.000 1.000 EPS
2.3.1.4 . 1.000 1.000 6.000
3.3.1.5 . 1.000 1.000 EPS
3.4.1.6 . 1.000 1.000 6.000
4.1.1.7 . 1.000 1.000 9.000
4.2.1.3 1.000 1.000 1.000 5.000

** LOWER LEVEL UPPER MARGINAL**

---- VAR z -INF 46.000 +INF

z objective function value
GAMS Output for Four Travels Given:

MODEL STATISTICS
BLOCKS OF EQUATIONS 15 SINGLE EQUATIONS 9,745
BLOCKS OF VARIABLES 4 SINGLE VARIABLES 1,261
NON ZERO ELEMENTS 33,507 NON LINEAR N-Z 27,900
DERIVATIVE POOL 10 CONSTANT POOL 20
CODE LENGTH 64,236 DISCRETE VARIABLES 596
GENERATION TIME = 0.093 SECONDS 8 MB 24.3.3 r48116 WEX-WEI
EXECUTION TIME = 0.093 SECONDS 8 MB 24.3.3 r48116 WEX-WEI

GAMS 24.3.3 r48116 Released Sep 19, 2014 WEX-WEI x86 64bit/MS Windows
04/22/15 10:27:37 Page 5

General Algebraic Modeling System
Solution Report SOLVE Thesismodel Using MINLP From line 79

SOLVE SUMMARY
MODEL Thesismodel OBJECTIVE z
TYPE MINLP DIRECTION MAXIMIZE
SOLVER ANTIGONE FROM LINE 79

**** SOLVER STATUS 1 Normal Completion
**** MODEL STATUS 8 Integer Solution

** OBJECTIVE VALUE 67.0000
RESOURCE USAGE, LIMIT 386.508 40000.000
ITERATION COUNT, LIMIT 4 2000000000
EVALUATION ERRORS 0 0

ANTIGONE 24.3.3 r48116 Released Sep 19, 2014 WEI x86 64bit/MS Windows

-------------------------------------------------------------------
ANTIGONE: Algorithms for coNTinuous/Integer Global Optimization; Version 1.1
Ruth Misener and Christodoulos A. Floudas
Computer-Aided Systems Laboratory (CASL)
Department of Chemical & Biological Engineering; Princeton University

-------------------------------------------------------------------
Termination Status : Global maximum
Best Feasible Point: +6.700000e+001
Best Possible Point: +7.370000e+001
Relative Gap: +1.000000e-001

CONOPT 3 24.3.3 r48116 Released Sep 19, 2014 WEI x86 64bit/MS Windows

Copyright (C) ARKI Consulting and Development A/S
Bag gsvaerd vej 246 A
DK-2880 Bagsvaerd, Denmark

** Warning ** The number of nonlinear derivatives equal to zero
in the initial point is large (= 55 percent).
A better initial point will probably help the
optimization.

** Warning ** The variance of the derivatives in the initial
point is large (= 4.5 ). A better initial
a better scaling, or better bounds on the variables will probably help the optimization.

** Optimal solution. Reduced gradient less than tolerance.**

<table>
<thead>
<tr>
<th>CONOPT time Total</th>
<th>0.055 seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>of which:</td>
<td></td>
</tr>
<tr>
<td>1st Derivative evaluations</td>
<td>0.012 = 21.8%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LOWER</th>
<th>LEVEL</th>
<th>UPPER</th>
<th>MARGINAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQU obj</td>
<td>.</td>
<td>.</td>
<td>1.000</td>
</tr>
<tr>
<td>EQU time</td>
<td>-INF</td>
<td>75.000</td>
<td>75.000</td>
</tr>
</tbody>
</table>

--- VAR x equals 1 if arc is traveled for kth time

<table>
<thead>
<tr>
<th>LOWER</th>
<th>LEVEL</th>
<th>UPPER</th>
<th>MARGINAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1.1.1</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>1.2.1.10</td>
<td>.</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>1.3.1.7</td>
<td>.</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>1.4.1.2</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>2.3.1.4</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>2.3.2.11</td>
<td>.</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>3.0.1.12</td>
<td>.</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>3.4.1.5</td>
<td>.</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>3.4.2.8</td>
<td>.</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>4.1.1.6</td>
<td>.</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>4.1.2.9</td>
<td>.</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>4.2.1.3</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LOWER</th>
<th>LEVEL</th>
<th>UPPER</th>
<th>MARGINAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.4.1.5</td>
<td>.</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>4.1.1.6</td>
<td>.</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>4.1.2.9</td>
<td>.</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>4.2.1.3</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
</tbody>
</table>

--- VAR z equals 1 if arc is traveled for kth time

<table>
<thead>
<tr>
<th>LOWER</th>
<th>LEVEL</th>
<th>UPPER</th>
<th>MARGINAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>-INF</td>
<td>67.000</td>
<td>+INF</td>
<td>.</td>
</tr>
</tbody>
</table>

z objective function value
GAMS Output for Five Travels Given:
MODEL STATISTICS
BLOCKS OF EQUATIONS 15 SINGLE EQUATIONS 9,745
BLOCKS OF VARIABLES 4 SINGLE VARIABLES 1,261
NON ZERO ELEMENTS 33,507 NON LINEAR N-Z 27,900
DERIVATIVE POOL 10 CONSTANT POOL 20
CODE LENGTH 64,236 DISCRETE VARIABLES 595
GENERATION TIME = 0.109 SECONDS 8 MB 24.3.3 r48116 WEX-WEI
EXECUTION TIME = 0.109 SECONDS 8 MB 24.3.3 r48116 WEX-WEI

GAMS 24.3.3 r48116 Released Sep 19, 2014 WEX-WEI x86 64bit/MS Windows
04/22/15 09:49:59 Page 5
General Algebraic Modeling System
Solution Report SOLVE Thesismodel Using MINLP From line 80
SOLVE SUMMARY
MODEL Thesismodel OBJECTIVE z
TYPE MINLP DIRECTION MAXIMIZE
SOLVER ANTIGONE FROM LINE 80
**** SOLVER STATUS 1 Normal Completion
**** MODEL STATUS 8 Integer Solution
**** OBJECTIVE VALUE 67.0000
RESOURCE USAGE, LIMIT 417.407 40000.000
ITERATION COUNT, LIMIT 4 2000000000
EVALUATION ERRORS 0 0
ANTIGONE 24.3.3 r48116 Released Sep 19, 2014 WEX x86 64bit/MS Windows
-----------------------------------------------
ANTIGONE: Algorithms for coNTinuous/Integer Global Optimization; Version 1.1
Ruth Misener and Christodoulos A. Floudas
Computer-Aided Systems Laboratory (CASL)
Department of Chemical & Biological Engineering; Princeton University
-----------------------------------------------
Termination Status: Global maximum
Best Feasible Point: +6.700000e+001
Best Possible Point: +7.370000e+001
Relative Gap: +1.000000e-001
CONOPT 3 24.3.3 r48116 Released Sep 19, 2014 WEI x86 64bit/MS Windows
CONOPT 3 version 3.16C
Copyright (C) ARKI Consulting and Development A/S
Bagsvaerdvej 246 A
DK-2880 Bagsvaerd, Denmark
** Warning ** The number of nonlinear derivatives equal to zero
in the initial point is large (= 55 percent).
A better initial point will probably help the optimization.
** Warning ** The variance of the derivatives in the initial
point is large (= 4.8 ). A better initial
point, a better scaling, or better bounds on the
variables will probably help the optimization.

** Optimal solution. Reduced gradient less than tolerance.

** CONOPT time Total 0.058 seconds
of which: Function evaluations 0.010 = 17.2%
1st Derivative evaluations 0.013 = 22.4%

<table>
<thead>
<tr>
<th>LOWER</th>
<th>LEVEL</th>
<th>UPPER</th>
<th>MARGINAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQU</td>
<td>obj</td>
<td>.</td>
<td>1.000</td>
</tr>
<tr>
<td>EQU</td>
<td>time</td>
<td>-INF</td>
<td>75.000</td>
</tr>
</tbody>
</table>

--- VAR x equals 1 if arc is traveled for kth time

<table>
<thead>
<tr>
<th>LOWER</th>
<th>LEVEL</th>
<th>UPPER</th>
<th>MARGINAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1.1.1</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>0.3.1.7</td>
<td></td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>1.4.1.2</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>1.4.2.10</td>
<td></td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>2.1.1.9</td>
<td></td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>2.3.1.4</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>3.0.1.12</td>
<td></td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>3.2.1.8</td>
<td></td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>3.4.1.5</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>4.1.1.6</td>
<td></td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>4.2.1.3</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>4.3.1.11</td>
<td></td>
<td>1.000</td>
<td>1.000</td>
</tr>
</tbody>
</table>

--- VAR z -INF 67.000 +INF |

z objective function value
GAMS Output for Eight Travels Given:
MODEL STATISTICS
BLOCKS OF EQUATIONS          15     SINGLE EQUATIONS        9,745
BLOCKS OF VARIABLES           4     SINGLE VARIABLES        1,261
NON ZERO ELEMENTS            33,507     NON LINEAR N-Z        27,900
DERIVATIVE POOL             10     CONSTANT POOL          20
CODE LENGTH                64,236     DISCRETE VARIABLES      592
GENERATION TIME            =        0.078 SECONDS      8 MB 24.3.3 r48116 WEX-WEI
EXECUTION TIME             =        0.078 SECONDS      8 MB 24.3.3 r48116 WEX-WEI

GAMS 24.3.3  r48116 Released Sep 19, 2014 WEX-WEI x86 64bit/MS Windows
04/22/15 10:40:03 Page 5
General Algebraic Modeling System
Solution Report  SOLVE Thesismodel Using MINLP From line 83

SOLVE SUMMARY
MODEL   Thesismodel         OBJECTIVE  z
TYPE    MINLP               DIRECTION MAXIMIZE
SOLVER  ANTIGONE            FROM LINE  83
**** SOLVER STATUS     1 Normal Completion
**** MODEL STATUS      8 Integer Solution
**** OBJECTIVE VALUE               67.0000
RESOURCE USAGE, LIMIT        495.055     40000.000
ITERATION COUNT, LIMIT         4    2000000000
EVALUATION ERRORS              0             0
ANTIGONE         24.3.3 r48116 Released Sep 19, 2014 WEI x86 64bit/MS Windows
-------------------------------------------------------------------
ANTIGONE: Algorithms for coNTinuous/Integer Global Optimization; Version 1.1
Ruth Misener and Christodoulos A. Floudas
Computer-Aided Systems Laboratory (CASL)
Department of Chemical & Biological Engineering; Princeton University
-------------------------------------------------------------------
Termination Status : Global maximum
Best Feasible Point: +6.700000e+001
Best Possible Point: +7.370000e+001
Relative Gap: +1.000000e-001
ANTIGONE 24.3.3 r48116 Released Sep 19, 2014 WEI x86 64bit/MS Windows
-------------------------------------------------------------------
CONOPT 3   24.3.3 r48116 Released Sep 19, 2014 WEI x86 64bit/MS Windows
Copyright (C) ARKI Consulting and Development A/S
   Bagsvaerdvej 246 A
   DK-2880 Bagsvaerd, Denmark
** Warning ** The number of nonlinear derivatives equal to zero
   in the initial point is large (= 55 percent).
   A better initial point will probably help the optimization.
** Warning ** The variance of the derivatives in the initial
   point is large (= 5.4 ). A better initial
point, a better scaling, or better bounds on the variables will probably help the optimization.

** Optimal solution. Reduced gradient less than tolerance.**

CONOPT time Total \(0.063\) seconds

of which: Function evaluations \(0.014 = 22.2\%\)

1st Derivative evaluations \(0.015 = 23.8\%\)

<table>
<thead>
<tr>
<th>LOWER</th>
<th>LEVEL</th>
<th>UPPER</th>
<th>MARGINAL</th>
</tr>
</thead>
</table>

--- EQU obj . . . 1.000

--- EQU time -INF 75.000 75.000 .

--- VAR x equals 1 if arc is traveled for kth time

<table>
<thead>
<tr>
<th>LOWER</th>
<th>LEVEL</th>
<th>UPPER</th>
<th>MARGINAL</th>
</tr>
</thead>
</table>

0.1.1.1 1.000 1.000 1.000 EPS

1.2.1.7 1.000 1.000 1.000 5.000

1.3.1.11 . 1.000 1.000 6.000

1.4.1.2 1.000 1.000 1.000 9.000

2.3.1.4 1.000 1.000 1.000 6.000

2.3.2.8 1.000 1.000 1.000 6.000

3.0.1.12 . 1.000 1.000 EPS

3.4.1.5 1.000 1.000 1.000 6.000

3.4.2.9 . 1.000 1.000 6.000

4.1.1.6 1.000 1.000 1.000 9.000

4.1.2.10 . 1.000 1.000 9.000

4.2.1.3 1.000 1.000 1.000 5.000

<table>
<thead>
<tr>
<th>LOWER</th>
<th>LEVEL</th>
<th>UPPER</th>
<th>MARGINAL</th>
</tr>
</thead>
</table>

--- VAR z -INF 67.000 +INF .

z objective function value
GAMS Output for Ten Travels Given:
MODEL STATISTICS
BLOCKS OF EQUATIONS  15  SINGLE EQUATIONS  9,745
BLOCKS OF VARIABLES  4  SINGLE VARIABLES  1,261
NON ZERO ELEMENTS  33,507  NON LINEAR N-Z  27,900
DERIVATIVE POOL  10  CONSTANT POOL  20
CODE LENGTH  64,236  DISCRETE VARIABLES  590
GENERATION TIME  =  0.093 SECONDS  8 MB  24.3.3 r48116 WEX-WEI
EXECUTION TIME  =  0.093 SECONDS  8 MB  24.3.3 r48116 WEX-WEI

GAMS 24.3.3  r48116 Released Sep 19, 2014 WEX-WEI x86 64bit/MS Windows
04/22/15 10:49:36 Page 5
General Algebraic Modeling System
Solution Report  SOLVE Thesismodel Using MINLP From line 85

SOLVE SUMMARY
MODEL Thesismodel  OBJECTIVE  z
TYPE MINLP  DIRECTION MAXIMIZE
SOLVER ANTIGONE  FROM LINE 85

**** SOLVER STATUS  1 Normal Completion
**** MODEL STATUS  8 Integer Solution
**** OBJECTIVE VALUE  67.0000
RESOURCE USAGE, LIMIT  518.463  40000.000
ITERATION COUNT, LIMIT  4  200000000
EVALUATION ERRORS  0  0

ANTIGONE  24.3.3 r48116 Released Sep 19, 2014 WEI x86 64bit/MS Windows

---------------------------------------------------------------------
ANTIGONE: Algorithms for coNTinuous/Integer Global Optimization; Version 1.1
Ruth Misener and Christodoulos A. Floudas
Computer-Aided Systems Laboratory (CASL)
Department of Chemical & Biological Engineering; Princeton University
---------------------------------------------------------------------
Termination Status : Global maximum
Best Feasible Point: +6.700000e+001
Best Possible Point: +7.370000e+001
Relative Gap: +1.000000e-001

CONOPT 3  24.3.3 r48116 Released Sep 19, 2014 WEI x86 64bit/MS Windows

CONOPT 3 version 3.16C
Copyright (C) ARKI Consulting and Development A/S
Bagsvaerdvej 246 A
DK-2880 Bagsvaerd, Denmark
** Warning  ** The number of nonlinear derivatives equal to zero
in the initial point is large (= 56 percent).
A better initial point will probably help the
optimization.
** Optimal solution. Reduced gradient less than tolerance.
CONOPT time Total  0.070 seconds
of which: Function evaluations       0.017 = 24.3%
  1st Derivative evaluations       0.018 = 25.7%

<table>
<thead>
<tr>
<th>LOWER</th>
<th>LEVEL</th>
<th>UPPER</th>
<th>MARGINAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQU obj</td>
<td>.</td>
<td>.</td>
<td>1.000</td>
</tr>
<tr>
<td>EQU time</td>
<td>-INF</td>
<td>75.000</td>
<td>75.000</td>
</tr>
</tbody>
</table>

--- VAR x equals 1 if arc is traveled for kth time

<table>
<thead>
<tr>
<th>LOWER</th>
<th>LEVEL</th>
<th>UPPER</th>
<th>MARGINAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1.1.1</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>1.2.1.7</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>1.4.1.2</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>1.4.2.10</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>2.3.1.4</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>2.3.2.8</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>3.0.1.12</td>
<td>.</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>3.1.1.9</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>3.4.1.5</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>4.1.1.6</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>4.2.1.3</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>4.3.1.11</td>
<td>.</td>
<td>1.000</td>
<td>1.000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LOWER</th>
<th>LEVEL</th>
<th>UPPER</th>
<th>MARGINAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAR z</td>
<td>-INF</td>
<td>67.000</td>
<td>+INF</td>
</tr>
</tbody>
</table>

z objective function value
GAMS Output for Twelve Travels Given:

MODEL STATISTICS
BLOCKS OF EQUATIONS 15  SINGLE EQUATIONS 9,745
BLOCKS OF VARIABLES 4  SINGLE VARIABLES 1,261
NON ZERO ELEMENTS 33,507  NON LINEAR N-Z 27,900
DERIVATIVE POOL 10  CONSTANT POOL 20
CODE LENGTH 64,236  DISCRETE VARIABLES 588
GENERATION TIME = 0.156 SECONDS  8 MB  24.3.3 r48116 WEX-WEI
EXECUTION TIME = 0.156 SECONDS  8 MB  24.3.3 r48116 WEX-WEI

GAMS 24.3.3 r48116 Released Sep 19, 2014 WEX-WEI x86 64bit/MS Windows
General Algebraic Modeling System
Solution Report  SOLVE Thesismodel Using MINLP From line 87

SOLVE SUMMARY
MODEL Thesismodel OBJECTIVE z
TYPE MINLP DIRECTION MAXIMIZE
SOLVER ANTIGONE FROM LINE 87
**** SOLVER STATUS 1 Normal Completion
**** MODEL STATUS 8 Integer Solution
**** OBJECTIVE VALUE 67.0000
RESOURCE USAGE, LIMIT 7.166  4000.000
ITERATION COUNT, LIMIT 4 2000000000
EVALUATION ERRORS 0 0

ANTIGONE: Algorithms for coNTinuous/Integer Global Optimization; Version 1.1
Ruth Misener and Christodoulos A. Floudas
Computer-Aided Systems Laboratory (CASL)
Department of Chemical & Biological Engineering; Princeton University

Termination Status : Global maximum
Best Feasible Point: +6.700000e+001
Best Possible Point: +7.370000e+001
Relative Gap: +1.000000e-001

CONOPT 3 24.3.3 r48116 Released Sep 19, 2014 WEI x86 64bit/MS Windows
CONOPT 3 version 3.16C
Copyright (C) ARKI Consulting and Development A/S
Bagsvaerdvej 246 A
DK-2880 Bagsvaerd, Denmark
** Warning ** The number of nonlinear derivatives equal to zero
in the initial point is large (= 56 percent).
A better initial point will probably help the optimization.
** Optimal solution. Reduced gradient less than tolerance.
CONOPT time Total 0.084 seconds
of which: Function evaluations 0.017 = 20.2%
1st Derivative evaluations 0.021 = 25.0%

<table>
<thead>
<tr>
<th>LOWER</th>
<th>LEVEL</th>
<th>UPPER</th>
<th>MARGINAL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----</td>
<td>EQU obj</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>----</td>
<td>EQU time</td>
<td>-INF</td>
<td>75.000</td>
</tr>
<tr>
<td>----</td>
<td>VAR x equals 1 if arc is traveled for kth time</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LOWER</th>
<th>LEVEL</th>
<th>UPPER</th>
<th>MARGINAL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.1.1.1</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>1.2.1.7</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>1.4.1.2</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>1.4.2.10</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>2.3.1.4</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>2.3.2.8</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>3.0.1.12</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>3.1.1.9</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>3.4.1.5</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>4.1.1.6</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>4.2.1.3</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>4.3.1.11</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LOWER</th>
<th>LEVEL</th>
<th>UPPER</th>
<th>MARGINAL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----</td>
<td>VAR z</td>
<td>-INF</td>
<td>67.000</td>
</tr>
<tr>
<td>z objective function value</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Code for Maximizing Weight – Six Node Problem:

//Function Prototypes
int findNextTempNode (int j_value);
int findLastNode (int j_value);

//Constant Variables
int NODE_SIZE = 6;
int TRAVEL_DIS = 6;
int BUFFER_SIZE = 30;
int TIME_CONST = 160;
int W_INC = 5;

//Global Variables
int node[6] = {0, 1, 2, 3, 4, 5};
int X[6][6]; //X, Z, W, and t are =
int Xtemp[6][6];
int Z[6][6];

int W[6][6] = {
    {0, 5, 4, 9, 4, 6},
    {5, 0, 6, 5, 6, 7},
    {4, 6, 0, 4, 6, 7},
    {9, 5, 4, 0, 5, 4},
    {4, 6, 6, 5, 0, 7},
    {6, 7, 7, 4, 7, 0}
};

int t[6][6] = {
    {1000, 4, 6, 8, 6, 4},
    {4, 1000, 7, 6, 6, 5},
    {6, 7, 1000, 6, 5, 7},
    {8, 6, 6, 1000, 5, 4},
    {6, 6, 5, 5, 1000, 6},
    {4, 5, 7, 4, 6, 1000}
};

int currentNode = 0;
int totalWeight = 0;
int totalTime = 0;
int tempNode[5];
int loopnum = 0;
int currentPathSum = 0;
int tempPathSum = 0;
int tempTotalTime = 0;

int tempPathNodes[5];
int currentPathNodes[6];
//Main Function
int main (void) {

    int a, b, c, d, e;
    int l, m, n, o;
    int k;
    int p, q, r, s;
    int ii, jj, kk;
    int count = 0;

    printf("Number of Nodes: %d\n", NODE_SIZE);
    printf("Temp Travel Length: %d\n", TRAVEL_DIS);
    printf("Size of Buffer: %d\n", BUFFER_SIZE);
    printf("Total Time Limit: %d\n", TIME_CONST);
    printf("Weight for Untraveled Nodes: %d\n", W_INC);

    while ( totalTime < TIME_CONST ) {

        count = count + 1;
        tempPathNodes[0] = currentNode;
        currentPathSum = 0;

        for (a = 0; a < NODE_SIZE; a++) {
            loopnum = 0;
            tempTotalTime = totalTime;
            tempNode[loopnum] = currentNode;
            tempPathSum = 0;

            if (tempNode[loopnum] == a) continue;
            if (((Z[tempNode[loopnum]][a] + BUFFER_SIZE) > tempTotalTime) &&
                ((X[tempNode[loopnum]][a]) != 0)) continue;
            if ((tempTotalTime + t[tempNode[loopnum]][a]) > TIME_CONST) continue;
            Xtemp[tempNode[loopnum]][a] = X[tempNode[loopnum]][a];
            l = tempNode[loopnum];
            p = Z[tempNode[loopnum]][a];

            if ((totalTime > (TIME_CONST/2)) &&
                ((Xtemp[tempNode[loopnum]][a]) == 0) &&
                ((X[tempNode[loopnum]][a]) == 0)){
                tempPathSum += W_INC;
            }
            findNextTempNode(a);
            tempPathNodes[1] = tempNode[loopnum + 1];

            for (b = 0; b < NODE_SIZE; b++) {
                loopnum = 1;
if (tempNode[loopnum] == b) continue;
if (((Z[tempNode[loopnum]][b] + BUFFER_SIZE) > tempTotalTime) &&
(((Xtemp[tempNode[loopnum]][b]) != 0) || ((X[tempNode[loopnum]][b]) != 0)))
continue;
if ((tempTotalTime + t[tempNode[loopnum]][b]) > TIME_CONST) continue;
Xtemp[tempNode[loopnum]][b] = X[tempNode[loopnum]][b];
m = tempNode[loopnum];
q = Z[tempNode[loopnum]][b];
if ((totalTime > (TIME_CONST/2)) && ((Xtemp[tempNode[loopnum]][b]) == 0) &&
((X[tempNode[loopnum]][b]) == 0)){
tempPathSum += W_INC;
}
findNextTempNode(b);
tempPathNodes[2] = tempNode[loopnum +1];

for (c = 0; c < NODE_SIZE; c++) {
    loopnum = 2;
    if (tempNode[loopnum] == c) continue;
    if (((Z[tempNode[loopnum]][c] + BUFFER_SIZE) > tempTotalTime) &&
(((Xtemp[tempNode[loopnum]][c]) != 0) || ((X[tempNode[loopnum]][c]) != 0)))
continue;
    if ((tempTotalTime + t[tempNode[loopnum]][c]) > TIME_CONST) continue;
    Xtemp[tempNode[loopnum]][c] = X[tempNode[loopnum]][c];
n = tempNode[loopnum];
r = Z[tempNode[loopnum]][c];
    if (findNextTempNode(c) == 0) continue;
    tempPathNodes[3] = tempNode[loopnum +1];

    for (d = 0; d < NODE_SIZE; d++) {
        loopnum = 3;
        if (tempNode[loopnum] == d) continue;
        if (((Z[tempNode[loopnum]][d] + BUFFER_SIZE) > tempTotalTime) &&
(((Xtemp[tempNode[loopnum]][d]) != 0) || ((X[tempNode[loopnum]][d]) != 0)))
continue;
        if ((tempTotalTime + t[tempNode[loopnum]][d]) > TIME_CONST) continue;
        Xtemp[tempNode[loopnum]][d] = X[tempNode[loopnum]][d];
o = tempNode[loopnum];
s = Z[tempNode[loopnum]][d];
        if (findNextTempNode(d) == 0) continue;
        tempPathNodes[4] = tempNode[loopnum +1];

        for (e = 0; e < NODE_SIZE; e++) {
            loopnum = 4;
            if (tempNode[loopnum] == e) continue;
if (((Z[tempNode[loopnum]][e] + BUFFER_SIZE) > tempTotalTime) &&
(((Xtemp[tempNode[loopnum]][e]) != 0) || ((X[tempNode[loopnum]][e]) != 0)))
continue;
if ((tempTotalTime + t[tempNode[loopnum]][e]) > TIME_CONST) continue;
Xtemp[tempNode[loopnum]][e] = X[tempNode[loopnum]][e];
if (findLastNode(e) == 0) continue;
tempPathSum -= W[tempNode[loopnum]][e];
}
if (tempPathSum > currentPathSum) {
currentPathSum = tempPathSum;
for (k = 0; k < (loopnum + 1); k++) {
currentPathNodes[k] = tempPathNodes[k];
}
currentPathNodes[loopnum + 1] = d;
}
tempPathSum -= W[o][d];
tempTotalTime -= t[o][d];
Xtemp[o][d] = 1;
Xtemp[d][o] = 1;
Z[o][d] = s;
Z[d][o] = s;
}
if (tempPathSum > currentPathSum) {
currentPathSum = tempPathSum;
for (k = 0; k < (loopnum + 1); k++) {
currentPathNodes[k] = tempPathNodes[k];
}
currentPathNodes[loopnum + 1] = c;
}
tempPathSum -= W[n][c];
tempTotalTime -= t[n][c];
Xtemp[n][c] = 1;
Xtemp[c][n] = 1;
Z[n][c] = r;
Z[c][n] = r;
}
if (tempPathSum > currentPathSum) {
currentPathSum = tempPathSum;
for (k = 0; k < (loopnum + 1); k++) {
currentPathNodes[k] = tempPathNodes[k];
}
currentPathNodes[loopnum + 1] = b;
}
tempPathSum -= W[m][b];
if ((totalTime > (TIME_CONST/2)) && ((Xtemp[tempNode[loopnum]][b]) == 0) && ((X[tempNode[loopnum]][b]) == 0)){
tempPathSum -= W_INC;
}
tempTotalTime -= t[m][b];
Xtemp[m][b] -= 1;
Xtemp[b][m] -= 1;
Z[m][b] = q;
Z[b][m] = q;
}

if ( tempPathSum > currentPathSum ) {
currentPathSum = tempPathSum;
for (k = 0; k < (loopnum + 1); k++) {
currentPathNodes[k] = tempPathNodes[k];
}
currentPathNodes[loopnum + 1] = a;
}
tempPathSum -= W[l][a];
if ((totalTime > (TIME_CONST/2)) && ((Xtemp[tempNode[l]][a]) == 0) && ((X[tempNode[l]][a]) == 0)){
tempPathSum -= W_INC;
}
tempTotalTime -= t[l][a];
Xtemp[l][a] -= 1;
Xtemp[a][l] -= 1;
Z[l][a] = p;
Z[a][l] = p;

}

printf("Current Path Nodes: %d ", currentPathNodes[0]);
for (k = 1; k < TRAVEL_DIS - 1; k++){
printf("%d ", currentPathNodes[k]);
}
printf("\t");
totalWeight += W[currentNode][currentPathNodes[1]];
totalTime += t[currentNode][currentPathNodes[1]];
Z[currentNode][currentPathNodes[1]] = totalTime;

Z[currentPathNodes[1]][currentNode] = totalTime;

X[currentNode][currentPathNodes[1]] += 1;
X[currentPathNodes[1]][currentNode] += 1;
printf("X(%d, %d) = %d\n", currentNode, currentPathNodes[1],
X[currentNode][currentPathNodes[1]], currentNode, currentPathNodes[1],
Z[currentNode][currentPathNodes[1]]);
printf("totalWeight: %d\ntotalTime: %d\n", totalWeight, totalTime);

//Find MIN
int min = 100;
for (ii = 0; ii < NODE_SIZE; ii++){
if ((t[currentPathNodes[1]][ii] < min) && (ii != currentNode)){
min = t[currentPathNodes[1]][ii];
}
currentNode = currentPathNodes[1];
if (totalTime + min > TIME_CONST){
break;
}
}
printf("Total Weight = %d\n", totalWeight);
printf("Total Time = %d\n", totalTime);
for (jj = 0; jj < NODE_SIZE; jj++){
for (kk = 0; kk < NODE_SIZE; kk++){
if ((X[jj][kk] == 0) && (X[kk][jj] == 0) && (jj != kk)){
printf("X(%d, %d) = 0\n", jj, kk);
}
}
}
return 0;

//Functions
int findNextTempNode (int j_value) {
tempPathSum += ((W[tempNode[loopnum]][j_value]));
tempTotalTime += t[tempNode[loopnum]][j_value];
Xtemp[tempNode[loopnum]][j_value] += 1;
Xtemp[j_value][tempNode[loopnum]] += 1;
Z[tempNode[loopnum]][j_value] = tempTotalTime;
Z[j_value][tempNode[loopnum]] = tempTotalTime;
tempNode[loopnum + 1] = j_value;
}

int findLastNode (int j_value) {
int k;
if ( ( tempTotalTime >= (Z[tempNode[loopnum]][j_value] + BUFFER_SIZE)) ||
((Xtemp[tempNode[loopnum]][j_value]) == 0) ) && (TIME_CONST > (totalTime +
tempTotalTime)) ){
}


tempPathSum += ((W[tempNode[loopnum]][j_value]));
if ( tempPathSum > currentPathSum ) {
currentPathSum = tempPathSum;
for (k = 0; k < (TRAVEL_DIS - 1); k++) {
currentPathNodes[k] = tempPathNodes[k];
}
currentPathNodes[TRAVEL_DIS - 1] = j_value;
}
return 1;
}
return 0;
Code for Max Weight minus Time – Six Node Problem:

// Function Prototypes
int findNextTempNode (int j_value);
int findLastNode (int j_value);

// Constant Variables
int NODE_SIZE = 6;
int TRAVEL_DIS = 6;
int BUFFER_SIZE = 30;
int TIME_CONST = 160;
int W_INC = 5;

// Global Variables
int nodes[] = {0, 1, 2, 3, 4, 5};
int X[6][6];
int Xtemp[6][6];
int Z[6][6];
int W[6][6];
int t[6][6];
int currentNode = 0;
totalWeight = 0;
totalTime = 0;
tempNode[5];
int loopnum = 0;
tempPathSum = 0;
tempPathSum = 0;
tempTotalTime = 0;
tempPathNodes[5];
t currentPathNodes[6];

// Main Funciton
int main (void) {

int a, b, c, d, e;
int l, m, n, o;
int aa, bb, cc, dd, ee;
int aaa, bbb, ccc, ddd;
int k;
int p, q, r, s;
int ii, jj, kk;
int count = 0;

printf("Number of Nodes: \%d\n", NODE_SIZE);
printf("Temp Travel Length: \%d\n", TRAVEL_DIS);
printf("Size of Buffer: \%d\n", BUFFER_SIZE);
printf("Total Time Limit: \%d\n", TIME_CONST);
printf("Weight for Untraveled Nodes: \%d\n", W_INC);

while ( totalTime < TIME_CONST ) {
    count = count + 1;
tempPathNodes[0] = currentNode;
currentPathSum = 0;

    for (a = 0; a < NODE_SIZE; a++) {
        loopnum = 0;
tempTotalTime = totalTime;
tempNode[loopnum] = currentNode;
tempPathSum = 0;
        if (tempNode[loopnum] == a) continue;
        if (((Z[tempNode[loopnum]][a] + BUFFER_SIZE) > tempTotalTime) &&
            ((X[tempNode[loopnum]][a]) != 0)) continue;
        if ((tempTotalTime + t[tempNode[loopnum]][a]) > TIME_CONST) continue;
        Xtemp[tempNode[loopnum]][a] = X[tempNode[loopnum]][a];
l = tempNode[loopnum];
p = Z[tempNode[loopnum]][a];
        if (((totalTime > (TIME_CONST/3)) && ((Xtemp[tempNode[loopnum]][a]) == 0) &&
            ((X[tempNode[loopnum]][a]) == 0))
            tempPathSum += W_INC;
    }
    findNextTempNode(a);
tempPathNodes[1] = tempNode[loopnum + 1];
aaa = tempPathSum;

    for (b = 0; b < NODE_SIZE; b++) {
        tempPathSum = aaa;
        loopnum = 1;
        if (tempNode[loopnum] == b) continue;
    }
}
if (((Z[tempNode[loopnum]][b] + BUFFER_SIZE) > tempTotalTime) &&
(((Xtemp[tempNode[loopnum]][b]) != 0) || ((X[tempNode[loopnum]][b]) != 0)))
continue;
if (((tempTotalTime + t[tempNode[loopnum]][b]) > TIME_CONST) continue;
Xtemp[tempNode[loopnum]][b] = X[tempNode[loopnum]][b];
m = tempNode[loopnum];
q = Z[tempNode[loopnum]][b];
if (((totalTime > (TIME_CONST/3)) && ((Xtemp[tempNode[loopnum]][b]) == 0) &&
((X[tempNode[loopnum]][b]) == 0)){
 tempPathSum += W_INC;
}
findNextTempNode(b);
tempPathNodes[2] = tempNode[loopnum +1];
bbb = tempPathSum;

for (c = 0; c < NODE_SIZE; c++) {
tempPathSum = bbb;
loopnum = 2;
if (tempNode[loopnum] == c) continue;
if (((Z[tempNode[loopnum]][c] + BUFFER_SIZE) > tempTotalTime) &&
(((Xtemp[tempNode[loopnum]][c]) != 0) || ((X[tempNode[loopnum]][c]) != 0)))
continue;
if (((tempTotalTime + t[tempNode[loopnum]][c]) > TIME_CONST) continue;
Xtemp[tempNode[loopnum]][c] = X[tempNode[loopnum]][c];
n = tempNode[loopnum];
r = Z[tempNode[loopnum]][c];
if (findNextTempNode(c) == 0) continue;
tempPathNodes[3] = tempNode[loopnum +1];
ccc = tempPathSum;

for (d = 0; d < NODE_SIZE; d++) {
tempPathSum = ccc;
loopnum = 3;
if (tempNode[loopnum] == d) continue;
if (((Z[tempNode[loopnum]][d] + BUFFER_SIZE) > tempTotalTime) &&
(((Xtemp[tempNode[loopnum]][d]) != 0) || ((X[tempNode[loopnum]][d]) != 0)))
continue;
if (((tempTotalTime + t[tempNode[loopnum]][d]) > TIME_CONST) continue;
Xtemp[tempNode[loopnum]][d] = X[tempNode[loopnum]][d];
o = tempNode[loopnum];
s = Z[tempNode[loopnum]][d];
if (findNextTempNode(d) == 0) continue;
tempPathNodes[4] = tempNode[loopnum +1];
ccc = tempPathSum;

for (e = 0; e < NODE_SIZE; e++) {
tempPathSum = ddd;
loopnum = 4;
if (tempNode[loopnum] == e) continue;
if (((Z[tempNode[loopnum]][e] + BUFFER_SIZE) > tempTotalTime) && ((Xtemp[tempNode[loopnum]][e]) != 0) || (X[tempNode[loopnum]][e]) != 0))
continue;
if ((tempTotalTime + t[tempNode[loopnum]][e]) > TIME_CONST) continue;
Xtemp[tempNode[loopnum]][e] = X[tempNode[loopnum]][e];
if (findLastNode(e) == 0) continue;
ee = W[tempNode[loopnum]][e] - t[tempNode[loopnum]][e];
tempPathSum -= ee;
if (tempPathSum > currentPathSum)
{
currentPathSum = tempPathSum;
for (k = 0; k < (loopnum + 1); k++)
currentPathNodes[k] = tempPathNodes[k];
currentPathNodes[loopnum + 1] = d;
}
dd = W[tempNode[o]][d] - t[tempNode[o]][d];
tempPathSum -= dd;
tempTotalTime -= t[o][d];
Xtemp[o][d] -= 1;
Xtemp[d][o] -= 1;
Z[o][d] = s;
Z[d][o] = s;
if (tempPathSum > currentPathSum)
{
currentPathSum = tempPathSum;
for (k = 0; k < (loopnum + 1); k++)
currentPathNodes[k] = tempPathNodes[k];
currentPathNodes[loopnum + 1] = c;
}
cc = W[tempNode[n]][c] - t[tempNode[n]][c];
tempPathSum -= cc;
tempTotalTime -= t[n][c];
Xtemp[n][c] -= 1;
Xtemp[c][n] -= 1;
Z[n][c] = r;
Z[c][n] = r;
if (tempPathSum > currentPathSum)
currentPathSum = tempPathSum;
for (k = 0; k < (loopnum + 1); k++) {
    currentPathNodes[k] = tempPathNodes[k];
}
currentPathNodes[loopnum + 1] = b;
}

bb = W[tempNode[m]][b] - t[tempNode[m]][b];
tempPathSum -= bb;
if ((totalTime > (TIME_CONST/3)) && ((Xtemp[tempNode[loopnum]][b]) == 0) &&
((X[tempNode[loopnum]][b]) == 0)){
tempPathSum -= W_INC;
}
tempTotalTime -= t[m][b];
Xtemp[m][b] -= 1;
Xtemp[b][m] -= 1;
Z[m][b] = q;
Z[b][m] = q;
}

if ( (tempPathSum > currentPathSum ) ) {
currentPathSum = tempPathSum;
for (k = 0; k < (loopnum + 1); k++) {
currentPathNodes[k] = tempPathNodes[k];
}
currentPathNodes[loopnum + 1] = a;
}
aa = W[tempNode[l]][a] - t[tempNode[l]][a];
if ((totalTime > (TIME_CONST/3)) && ((Xtemp[tempNode[l]][a]) == 0) &&
((X[tempNode[l]][a]) == 0)){
tempPathSum -= W_INC;
}tempPathSum -= aa;
tempTotalTime -= t[l][a];
Xtemp[l][a] -= 1;
Xtemp[a][l] -= 1;
Z[l][a] = p;
Z[a][l] = p;

printf("Current Path Nodes: %d ", currentPathNodes[0]);
for (k = 1; k < TRAVEL_DIS - 1; k++){
printf("%d ", currentPathNodes[k]);
}
printf("\t");
totalWeight += W[currentNode][currentPathNodes[1]];  
totalTime += t[currentNode][currentPathNodes[1]];  
Z[currentNode][currentPathNodes[1]] = totalTime;  

Z[currentPathNodes[1]][currentNode] = totalTime;  

X[currentNode][currentPathNodes[1]] += 1;  
X[currentPathNodes[1]][currentNode] += 1;  

printf("X(%d, %d) = %d\n", currentNode, currentPathNodes[1],  
X[currentNode][currentPathNodes[1]], currentNode, currentPathNodes[1],  
Z[currentNode][currentPathNodes[1]]);  

printf("totalWeight: %d\n", totalWeight, totalTime);  

//Find MIN  
int min = 100;  
for (ii = 0; ii < NODE_SIZE; ii++)  
if ((t[currentPathNodes[1]][ii] < min) && (ii != currentNode))  
min = t[currentPathNodes[1]][ii];  
}  
}  
currentNode = currentPathNodes[1];  
if (totalTime + min > TIME_CONST)  
break;  
}  
}  

printf("Total Weight = %d\n", totalWeight);  
printf("Total Time = %d\n", totalTime);  

for (jj = 0; jj < NODE_SIZE; jj++)  
for (kk = 0; kk < NODE_SIZE; kk++)  
if ((X[jj][kk] == 0) && (X[kk][jj] == 0) && (jj != kk))  
printf("X(%d, %d) = 0\n", jj, kk);  
}  
}  

return 0;  
}  

//Functions  
int findNextTempNode (int j_value) {  
int g;  
g = W[tempNode[loopnum]][j_value] - t[tempNode[loopnum]][j_value];  
tempPathSum += g;  
tempTotalTime += t[tempNode[loopnum]][j_value];
Xtemp[tempNode[loopnum]][j_value] += 1;
Xtemp[j_value][tempNode[loopnum]] += 1;
Z[tempNode[loopnum]][j_value] = tempTotalTime;
Z[j_value][tempNode[loopnum]] = tempTotalTime;
tempNode[loopnum + 1] = j_value;
}

int findLastNode (int j_value) {
int k;
int h;

if ( ((tempTotalTime >= (Z[tempNode[loopnum]][j_value] + BUFFER_SIZE)) ||
  ((Xtemp[tempNode[loopnum]][j_value]) == 0) ) && (TIME_CONST > (totalTime +
tempNode[loopnum][j_value]) ) )
  //Probable FLAG
  h = W[tempNode[loopnum]][j_value] - t[tempNode[loopnum]][j_value];
tempPathSum += h;
if ( tempPathSum > currentPathSum )
  currentPathSum = tempPathSum;
for (k = 0; k < (TRAVEL_DIS - 1); k++)
  currentPathNodes[k] = tempPathNodes[k];
}
currentPathNodes[TRAVEL_DIS - 1] = j_value;
}
return 1;
}
return 0;
}
Code for Max Weight – 20 Node Problem:
//Function Prototypes
int findNextTempNode (int j_value);
int findLastNode (int j_value);

//Constant Variables
int NODE_SIZE = 20;
int TRAVEL_DIS = 6;
int BUFFER_SIZE = 800;
int TIME_CONST = 5000;
int W_INC = 100;

//Global Variables
int nodes[] = {0, 1, 2, 3, 4, 5};
int X[20][20]; //X, Z, W, and t are size NODE_SIZE
int Xtemp[20][20];
int Z[20][20];
int W[20][20] = {
    {0, 9, 5, 7, 5, 4, 11, 10, 11, 10, 12, 5, 12, 5, 11, 6, 6, 5, 12, 11},
    {9, 0, 7, 5, 5, 6, 9, 9, 10, 11, 12, 4, 5, 11, 12, 8, 6, 5, 12, 12},
    {5, 7, 0, 8, 11, 7, 7, 8, 5, 11, 10, 11, 12, 6, 7, 9, 4, 6, 5},
    {7, 5, 8, 0, 4, 6, 8, 8, 4, 11, 9, 9, 5, 6, 6, 4, 7, 11, 7},
    {5, 5, 11, 4, 0, 4, 6, 8, 11, 5, 8, 12, 4, 7, 6, 11, 10, 9, 9, 9},
    {4, 6, 7, 6, 4, 0, 6, 7, 4, 6, 11, 10, 10, 7, 6, 4, 10, 7, 6, 9},
    {11, 9, 7, 8, 6, 6, 0, 4, 6, 6, 9, 9, 5, 11, 8, 9, 9, 10, 7, 7},
    {10, 9, 8, 8, 8, 7, 4, 0, 9, 12, 11, 10, 11, 10, 11, 7, 11, 10, 5, 6, 7},
    {11, 10, 5, 4, 11, 4, 6, 9, 0, 8, 8, 4, 7, 11, 11, 7, 9, 12, 4, 9},
    {10, 11, 14, 5, 6, 12, 8, 0, 5, 11, 6, 4, 9, 5, 10, 4, 10, 6},
    {12, 12, 10, 11, 8, 11, 9, 11, 8, 5, 0, 12, 7, 12, 7, 8, 9, 7, 10, 10},
    {5, 4, 10, 9, 12, 10, 9, 10, 4, 11, 12, 0, 6, 8, 4, 8, 8, 8, 4, 9},
    {12, 5, 11, 9, 4, 10, 5, 10, 7, 6, 7, 6, 0, 6, 7, 9, 9, 6, 5, 6},
    {5, 11, 12, 5, 7, 7, 11, 11, 11, 4, 12, 8, 6, 0, 4, 12, 6, 8, 12, 5},
    {11, 12, 6, 6, 6, 6, 8, 7, 11, 9, 7, 4, 7, 4, 0, 5, 7, 7, 11, 11},
    {6, 8, 7, 6, 11, 4, 9, 11, 7, 5, 8, 8, 9, 12, 5, 0, 9, 6, 5, 5},
    {6, 6, 9, 4, 10, 10, 9, 10, 9, 10, 9, 8, 9, 6, 7, 9, 0, 9, 8, 11},
    {5, 5, 4, 7, 9, 7, 10, 5, 12, 4, 7, 8, 6, 8, 7, 6, 9, 0, 11, 4},
    {12, 12, 6, 11, 9, 6, 7, 6, 4, 10, 10, 4, 5, 12, 11, 5, 8, 11, 0, 10},
    {11, 12, 5, 7, 9, 7, 9, 7, 9, 6, 10, 9, 6, 5, 11, 5, 11, 4, 10, 0}};

int t[20][20] = {
    {1000, 7, 5, 11, 12, 12, 9, 12, 8, 7, 11, 7, 9, 7, 9, 5, 9, 12, 5, 8},
    {7, 1000, 11, 4, 7, 5, 5, 10, 10, 7, 6, 10, 12, 5, 12, 5, 10, 10, 8, 9},
    {5, 11, 1000, 12, 4, 9, 4, 10, 5, 9, 11, 12, 9, 4, 10, 8, 10, 9, 6, 7},
    {11, 4, 12, 1000, 5, 8, 7, 10, 6, 11, 5, 11, 4, 9, 9, 5, 4, 12, 11, 9},
    {12, 7, 4, 5, 1000, 12, 11, 6, 5, 7, 8, 12, 8, 6, 7, 8, 8, 9, 8, 11},
    {12, 5, 9, 8, 12, 1000, 5, 4, 8, 4, 7, 4, 10, 8, 7, 9, 10, 6, 6, 11},
    {5, 4, 8, 8, 7, 4, 5, 12, 10, 8, 9, 11, 0, 10, 7, 9, 9, 7, 9, 6, 11, 5, 11, 4, 10, 0}};
int currentNode = 0;
int totalWeight = 0;
int totalTime = 0;
int tempNode[5];
int loopnum = 0;
int currentPathSum = 0;
int tempPathSum = 0;
int tempTotalTime = 0;
int tempPathNodes[5];
int currentPathNodes[6];
int path[627];

// Main Function
int main (void) { 
int a, b, c, d, e;
int l, m, n, o;
int k;
int p, q, r, s;
int ii, jj, kk, ll;
int count = 0;

printf("Number of Nodes: %d\n", NODE_SIZE);
printf("Temp Travel Length: %d\n", TRAVEL_DIS);
printf("Size of Buffer: %d\n", BUFFER_SIZE);
printf("Total Time Limit: %d\n", TIME_CONST);
printf("Weight for Untraveled Nodes: %d\n", W_INC);

while ( totalTime < TIME_CONST ) { 
path[count] = currentNode;
count = count + 1;
tempPathNodes[0] = currentNode;
currentPathSum = 0;
for (a = 0; a < NODE_SIZE; a++) {
    loopnum = 0;
    tempTotalTime = totalTime;
    tempNode[loopnum] = currentNode;
    tempPathSum = 0;
    if (tempNode[loopnum] == a) continue;
    if (((Z[tempNode[loopnum]][a] + BUFFER_SIZE) > tempTotalTime) &&
        ((X[tempNode[loopnum]][a]) != 0)) continue;
    if ((tempTotalTime + t[tempNode[loopnum]][a]) > TIME_CONST) continue;
    Xtemp[tempNode[loopnum]][a] = X[tempNode[loopnum]][a];
    l = tempNode[loopnum];
    p = Z[tempNode[loopnum]][a];
    if ((totalTime > (TIME_CONST/4)) && ((Xtemp[tempNode[loopnum]][a]) == 0) &&
        ((X[tempNode[loopnum]][a]) == 0))
        tempPathSum += W_INC;
    findNextTempNode(a);
    tempPathNodes[1] = tempNode[loopnum + 1];
}
for (b = 0; b < NODE_SIZE; b++) {
    loopnum = 1;
    if (tempNode[loopnum] == b) continue;
    if (((Z[tempNode[loopnum]][b] + BUFFER_SIZE) > tempTotalTime) &&
        (((Xtemp[tempNode[loopnum]][b]) != 0) || ((X[tempNode[loopnum]][b]) != 0)))
        continue;
    if ((tempTotalTime + t[tempNode[loopnum]][b]) > TIME_CONST) continue;
    Xtemp[tempNode[loopnum]][b] = X[tempNode[loopnum]][b];
    m = tempNode[loopnum];
    q = Z[tempNode[loopnum]][b];
    if ((totalTime > (TIME_CONST/4)) && ((Xtemp[tempNode[loopnum]][b]) == 0) &&
        ((X[tempNode[loopnum]][b]) == 0))
        tempPathSum += W_INC;
    findNextTempNode(b);
    tempPathNodes[2] = tempNode[loopnum + 1];
}
for (c = 0; c < NODE_SIZE; c++) {
    loopnum = 2;
    if (tempNode[loopnum] == c) continue;
    if (((Z[tempNode[loopnum]][c] + BUFFER_SIZE) > tempTotalTime) &&
        (((Xtemp[tempNode[loopnum]][c]) != 0) || ((X[tempNode[loopnum]][c]) != 0)))
        continue;
    if ((tempTotalTime + t[tempNode[loopnum]][c]) > TIME_CONST) continue;
    Xtemp[tempNode[loopnum]][c] = X[tempNode[loopnum]][c];
    n = tempNode[loopnum];
}
r = Z[tempNode[loopnum]][c];
if (findNextTempNode(c) == 0) continue;
tempPathNodes[3] = tempNode[loopnum + 1];

for (d = 0; d < NODE_SIZE; d++) {
    loopnum = 3;
    if (tempNode[loopnum] == d) continue;
    if (((Z[tempNode[loopnum]][d] + BUFFER_SIZE) > tempTotalTime) &&
        (((Xtemp[tempNode[loopnum]][d]) != 0) || ((X[tempNode[loopnum]][d]) != 0)))
        continue;
    if ((tempTotalTime + t[tempNode[loopnum]][d]) > TIME_CONST) continue;
    Xtemp[tempNode[loopnum]][d] = X[tempNode[loopnum]][d];
    o = tempNode[loopnum];
    s = Z[tempNode[loopnum]][d];
    if (findNextTempNode(d) == 0) continue;
    tempPathNodes[4] = tempNode[loopnum + 1];
}

for (e = 0; e < NODE_SIZE; e++) {
    loopnum = 4;
    if (tempNode[loopnum] == e) continue;
    if (((Z[tempNode[loopnum]][e] + BUFFER_SIZE) > tempTotalTime) &&
        (((Xtemp[tempNode[loopnum]][e]) != 0) || ((X[tempNode[loopnum]][e]) != 0)))
        continue;
    if ((tempTotalTime + t[tempNode[loopnum]][e]) > TIME_CONST) continue;
    Xtemp[tempNode[loopnum]][e] = X[tempNode[loopnum]][e];
    if (findLastNode(e) == 0) continue;
    tempPathSum -= W[tempNode[loopnum]][e];
}

if ( tempPathSum > currentPathSum ) {
    currentPathSum = tempPathSum;
    for (k = 0; k < (loopnum + 1); k++) {
        currentPathNodes[k] = tempPathNodes[k];
    }
    currentPathNodes[loopnum + 1] = d;
    tempPathSum -= W[o][d];
    tempTotalTime -= t[o][d];
    Xtemp[o][d] = 1;
    Xtemp[d][o] = 1;
    Z[o][d] = s;
    Z[d][o] = s;
}

if ( tempPathSum > currentPathSum ) {
    currentPathSum = tempPathSum;
}
for (k = 0; k < (loopnum + 1); k++) {
    currentPathNodes[k] = tempPathNodes[k];
}
currentPathNodes[loopnum + 1] = c;

tempPathSum -= W[n][c];
tempTotalTime -= t[n][c];
Xtemp[n][c] = 1;
Xtemp[c][n] = 1;
Z[n][c] = r;
Z[c][n] = r;
}

if ( tempPathSum > currentPathSum ) {
    currentPathSum = tempPathSum;
    for (k = 0; k < (loopnum + 1); k++) {
        currentPathNodes[k] = tempPathNodes[k];
    }
currentPathNodes[loopnum + 1] = b;
}
tempPathSum -= W[m][b];

if ((totalTime > (TIME_CONST/4)) && ((Xtemp[tempNode[loopnum]][b]) == 0) &&
    ((X[tempNode[loopnum]][b]) == 0)){
tempPathSum -= W_INC;
}
tempTotalTime -= t[m][b];
Xtemp[m][b] = 1;
Xtemp[b][m] = 1;
Z[m][b] = q;
Z[b][m] = q;
}

if ( tempPathSum > currentPathSum ) {
    currentPathSum = tempPathSum;
    for (k = 0; k < (loopnum + 1); k++) {
        currentPathNodes[k] = tempPathNodes[k];
    }
currentPathNodes[loopnum + 1] = a;
}
tempPathSum -= W[l][a];
if ((totalTime > (TIME_CONST/4)) && ((Xtemp[tempNode[l]][a]) == 0) &&
    ((X[tempNode[l]][a]) == 0)){
tempPathSum -= W_INC;
}
tempTotalTime -= t[l][a];
Xtemp[l][a] -= 1;
Xtemp[a][l] -= 1;
Z[l][a] = p;
Z[a][l] = p;
}

totalWeight += W[currentNode][currentPathNodes[1]];
totalTime += t[currentNode][currentPathNodes[1]];
Z[currentNode][currentPathNodes[1]] = totalTime;
Z[currentPathNodes[1]][currentNode] = totalTime;
X[currentNode][currentPathNodes[1]] += 1;
X[currentPathNodes[1]][currentNode] += 1;

printf("X(%d, %d) = %d\n", currentNode, currentPathNodes[1],
X[currentNode][currentPathNodes[1]]);
printf("totalWeight: %d \t totalTime: %d\n", totalWeight, totalTime);

//Find MIN
int min = 100;
for (ii = 0; ii < NODE_SIZE; ii++){
if ((t[currentPathNodes[1]][ii] < min) && (ii != currentNode)){
  min = t[currentPathNodes[1]][ii];
}
}
currentNode = currentPathNodes[1];
if (totalTime + min > TIME_CONST){
  break;
}
}

for (jj = 0; jj < NODE_SIZE; jj++){
for (kk = 0; kk < NODE_SIZE; kk++){
if ((X[jj][kk] == 0) && (X[kk][jj] == 0) && (jj != kk)){
  printf("X(%d, %d) = 0\n", jj, kk);
}
}
}
return 0;

//Functions
int findNextTempNode (int j_value) {
tempPathSum += ((W[tempNode[loopnum]][j_value]));
tempTotalTime += t[tempNode[loopnum]][j_value];
Xtemp[tempNode[loopnum]][j_value] += 1;
Xtemp[j_value][tempNode[loopnum]] += 1;
Z[tempNode[loopnum]][j_value] = tempTotalTime;
Z[j_value][tempNode[loopnum]] = tempTotalTime;
tempNode[loopnum + 1] = j_value;
}

int findLastNode (int j_value) {
    int k;
    if ( ((tempTotalTime >= (Z[tempNode[loopnum]][j_value] + BUFFER_SIZE)) || ((Xtemp[tempNode[loopnum]][j_value]) == 0) ) && (TIME_CONST > (totalTime + t[tempNode[loopnum]][j_value]))) {
        //Probable FLAG
        tempPathSum += ((W[tempNode[loopnum]][j_value]));
        if ( tempPathSum > currentPathSum ) {
            currentPathSum = tempPathSum;
            for (k = 0; k < (TRAVEL_DIS - 1); k++) {
                currentPathNodes[k] = tempPathNodes[k];
            }
            currentPathNodes[TRAVEL_DIS - 1] = j_value;
        }
    }
    return 1;
}
return 0;
Output for Max Weight – Four Node Problem:
Number of Nodes: 4
Temp Travel Length: 6
Size of Buffer: 15
Total Time Limit: 60
Weight for Untraveled Nodes: 10
Current Path Nodes: 0 3 2 0 3   \(X(0, 3) = 1\)   \(Z(0, 3) = 7\)
totalWeight: 9   totalTime: 7
Current Path Nodes: 3 1 2 0 3   \(X(3, 1) = 1\)   \(Z(3, 1) = 14\)
totalWeight: 14   totalTime: 14
Current Path Nodes: 1 2 0 3 2   \(X(1, 2) = 1\)   \(Z(1, 2) = 19\)
totalWeight: 20   totalTime: 19
Current Path Nodes: 2 3 0 3 2   \(X(2, 3) = 1\)   \(Z(2, 3) = 24\)
totalWeight: 26   totalTime: 24
Current Path Nodes: 3 0 1 2 0   \(X(3, 0) = 2\)   \(Z(3, 0) = 31\)
totalWeight: 35   totalTime: 31
Current Path Nodes: 0 1 2 0 3   \(X(0, 1) = 1\)   \(Z(0, 1) = 35\)
totalWeight: 40   totalTime: 35
Current Path Nodes: 1 2 0 3 2   \(X(1, 2) = 2\)   \(Z(1, 2) = 40\)
totalWeight: 46   totalTime: 40
Current Path Nodes: 2 0 3 2 2   \(X(2, 0) = 1\)   \(Z(2, 0) = 48\)
totalWeight: 52   totalTime: 48
Current Path Nodes: 0 3 2 2 2   \(X(0, 3) = 3\)   \(Z(0, 3) = 55\)
totalWeight: 61   totalTime: 55
Current Path Nodes: 3 2 2 2 2   \(X(3, 2) = 2\)   \(Z(3, 2) = 60\)
totalWeight: 67   totalTime: 60
Total Weight = 67
Total Time = 60
Time to complete program = 0.000181 seconds
Output for Max Weight minus Time – Four Node Problem:

Number of Nodes: 4
Temp Travel Length: 6
Size of Buffer: 15
Total Time Limit: 60
Weight for Untraveled Nodes: 10

Current Path Nodes: 0 3 2 1 3
   X(0, 3) = 1
   Z(0, 3) = 7
   totalWeight: 9
   totalTime: 7

Current Path Nodes: 3 2 3 2 3
   X(3, 2) = 1
   Z(3, 2) = 12
   totalWeight: 15
   totalTime: 12

Current Path Nodes: 2 0 1 2 3
   X(2, 0) = 1
   Z(2, 0) = 20
   totalWeight: 21
   totalTime: 20

Current Path Nodes: 0 1 2 3 0
   X(0, 1) = 1
   Z(0, 1) = 24
   totalWeight: 26
   totalTime: 24

Current Path Nodes: 1 3 2 1 3
   X(1, 3) = 1
   Z(1, 3) = 31
   totalWeight: 31
   totalTime: 31

Current Path Nodes: 3 2 1 0 3
   X(3, 2) = 2
   Z(3, 2) = 36
   totalWeight: 37
   totalTime: 36

Current Path Nodes: 2 1 0 3 2
   X(2, 1) = 1
   Z(2, 1) = 41
   totalWeight: 43
   totalTime: 41

Current Path Nodes: 1 0 3 2 2
   X(1, 0) = 2
   Z(1, 0) = 45
   totalWeight: 48
   totalTime: 45

Current Path Nodes: 0 2 3 2 2
   X(0, 2) = 2
   Z(0, 2) = 53
   totalWeight: 54
   totalTime: 53

Current Path Nodes: 2 3 3 2 2
   X(2, 3) = 3
   Z(2, 3) = 58
   totalWeight: 60
   totalTime: 58

Total Weight = 60
Total Time = 58
Time to complete program = 0.000271 seconds
Output for Max Weight – Six Node Problem:
Number of Nodes: 6
Temp Travel Length: 6
Size of Buffer: 30
Total Time Limit: 160
Weight for Untraveled Nodes: 5
Current Path Nodes: 0 3 1 2 5   X(0, 3) = 1      Z(0, 3) = 8
totalWeight: 9          totalTime: 8
Current Path Nodes: 3 1 5 2 4   X(3, 1) = 1      Z(3, 1) = 14
totalWeight: 14         totalTime: 14
Current Path Nodes: 1 2 5 4 3   X(1, 2) = 1      Z(1, 2) = 21
totalWeight: 20         totalTime: 21
Current Path Nodes: 2 4 1 5 0   X(2, 4) = 1      Z(2, 4) = 26
totalWeight: 26         totalTime: 26
Current Path Nodes: 4 1 5 4 3   X(4, 1) = 1      Z(4, 1) = 32
totalWeight: 32         totalTime: 32
Current Path Nodes: 1 5 0 3 4   X(1, 5) = 1      Z(1, 5) = 37
totalWeight: 39         totalTime: 37
Current Path Nodes: 5 0 3 4 5   X(5, 0) = 1      Z(5, 0) = 41
totalWeight: 45         totalTime: 41
Current Path Nodes: 0 3 1 2 5   X(0, 3) = 2      Z(0, 3) = 49
totalWeight: 54         totalTime: 49
Current Path Nodes: 3 4 5 2 1   X(3, 4) = 1      Z(3, 4) = 54
totalWeight: 59         totalTime: 54
Current Path Nodes: 4 5 2 1 3   X(4, 5) = 1      Z(4, 5) = 60
totalWeight: 66         totalTime: 60
Current Path Nodes: 5 2 1 5 0   X(5, 2) = 1      Z(5, 2) = 67
totalWeight: 73         totalTime: 67
Current Path Nodes: 2 4 1 5 0   X(2, 4) = 2      Z(2, 4) = 72
totalWeight: 79         totalTime: 72
Current Path Nodes: 4 1 3 0 5   X(4, 1) = 2      Z(4, 1) = 78
totalWeight: 85         totalTime: 78
Current Path Nodes: 1 5 0 3 4   X(1, 5) = 2      Z(1, 5) = 83
totalWeight: 92         totalTime: 83
Current Path Nodes: 5 3 0 5 2   X(5, 3) = 1      Z(5, 3) = 87
totalWeight: 96         totalTime: 87
Current Path Nodes: 3 0 5 2 1   X(3, 0) = 3      Z(3, 0) = 95
totalWeight: 105        totalTime: 95
Current Path Nodes: 0 1 2 4 5   X(0, 1) = 1      Z(0, 1) = 99
totalWeight: 110        totalTime: 99
Current Path Nodes: 1 2 3 1 5   X(1, 2) = 2      Z(1, 2) = 106
totalWeight: 116        totalTime: 106
Current Path Nodes: 2 0 5 1 3   X(2, 0) = 1      Z(2, 0) = 112
totalWeight: 120        totalTime: 112
Current Path Nodes: 0 4 1 5 0   X(0, 4) = 1      Z(0, 4) = 118
totalWeight: 124        totalTime: 118
Current Path Nodes: 4 2 5 1 0  X(4, 2) = 3  Z(4, 2) = 123  
   totalWeight: 130  totalTime: 123  
Current Path Nodes: 2 3 0 1 5  X(2, 3) = 1  Z(2, 3) = 129  
   totalWeight: 134  totalTime: 129  
Current Path Nodes: 3 0 5 1 2  X(3, 0) = 4  Z(3, 0) = 137  
   totalWeight: 143  totalTime: 137  
Current Path Nodes: 0 1 5 3 4  X(0, 1) = 2  Z(0, 1) = 141  
   totalWeight: 148  totalTime: 141  
Current Path Nodes: 1 4 3 5 0  X(1, 4) = 3  Z(1, 4) = 147  
   totalWeight: 154  totalTime: 147  
Current Path Nodes: 4 5 0 0 0  X(4, 5) = 2  Z(4, 5) = 153  
   totalWeight: 161  totalTime: 153  
Current Path Nodes: 5 1 1 0 0  X(5, 1) = 3  Z(5, 1) = 158  
   totalWeight: 168  totalTime: 158  
Total Weight = 168  
Total Time = 158  
Time to complete program = 0.001334 seconds
Output for Max Weight minus Time – Six Node Problem:

Number of Nodes: 6
Temp Travel Length: 6
Size of Buffer: 30
Total Time Limit: 160
Weight for Untraveled Nodes: 5

Current Path Nodes: 0 5 1 4 5  X(0, 5) = 1  Z(0, 5) = 4
totalWeight: 6  totalTime: 4

Current Path Nodes: 5 1 2 4 5  X(5, 1) = 1  Z(5, 1) = 9
totalWeight: 13  totalTime: 9

Current Path Nodes: 1 0 3 4 5  X(1, 0) = 1  Z(1, 0) = 13
totalWeight: 18  totalTime: 13

Current Path Nodes: 0 3 1 4 5  X(0, 3) = 1  Z(0, 3) = 21
totalWeight: 27  totalTime: 21

Current Path Nodes: 3 5 2 4 5  X(3, 5) = 1  Z(3, 5) = 25
totalWeight: 31  totalTime: 25

Current Path Nodes: 5 4 2 0 5  X(5, 4) = 1  Z(5, 4) = 31
totalWeight: 38  totalTime: 31

Current Path Nodes: 4 2 5 2 1  X(4, 2) = 1  Z(4, 2) = 36
totalWeight: 44  totalTime: 36

Current Path Nodes: 2 5 0 1 5  X(2, 5) = 1  Z(2, 5) = 43
totalWeight: 51  totalTime: 43

Current Path Nodes: 5 1 0 5 0  X(5, 1) = 2  Z(5, 1) = 48
totalWeight: 58  totalTime: 48

Current Path Nodes: 1 0 3 4 5  X(1, 0) = 2  Z(1, 0) = 52
totalWeight: 63  totalTime: 52

Current Path Nodes: 0 5 3 4 5  X(0, 5) = 2  Z(0, 5) = 56
totalWeight: 69  totalTime: 56

Current Path Nodes: 5 3 1 4 5  X(5, 3) = 2  Z(5, 3) = 60
totalWeight: 73  totalTime: 60

Current Path Nodes: 3 1 2 4 5  X(3, 1) = 1  Z(3, 1) = 66
totalWeight: 78  totalTime: 66

Current Path Nodes: 1 2 3 4 5  X(1, 2) = 1  Z(1, 2) = 73
totalWeight: 84  totalTime: 73

Current Path Nodes: 2 0 3 4 5  X(2, 0) = 1  Z(2, 0) = 79
totalWeight: 88  totalTime: 79

Current Path Nodes: 0 4 3 5 3  X(0, 4) = 1  Z(0, 4) = 85
totalWeight: 92  totalTime: 85

Current Path Nodes: 4 3 2 4 5  X(4, 3) = 1  Z(4, 3) = 90
totalWeight: 97  totalTime: 90

Current Path Nodes: 3 5 0 1 5  X(3, 5) = 3  Z(3, 5) = 94
totalWeight: 101  totalTime: 94

Current Path Nodes: 5 4 1 0 5  X(5, 4) = 2  Z(5, 4) = 100
totalWeight: 108  totalTime: 100

Current Path Nodes: 4 2 5 2 1  X(4, 2) = 2  Z(4, 2) = 105
totalWeight: 114  totalTime: 105
Current Path Nodes: 2 5 0 1 5  \( X(2, 5) = 2 \)  \( Z(2, 5) = 112 \)

totalWeight: 121  totalTime: 112

Current Path Nodes: 5 1 4 0 5  \( X(5, 1) = 3 \)  \( Z(5, 1) = 117 \)

totalWeight: 128  totalTime: 117

Current Path Nodes: 1 4 3 5 3  \( X(1, 4) = 1 \)  \( Z(1, 4) = 123 \)

totalWeight: 134  totalTime: 123

Current Path Nodes: 4 0 3 4 5  \( X(4, 0) = 2 \)  \( Z(4, 0) = 129 \)

totalWeight: 138  totalTime: 129

Current Path Nodes: 0 3 2 4 5  \( X(0, 3) = 2 \)  \( Z(0, 3) = 137 \)

totalWeight: 147  totalTime: 137

Current Path Nodes: 3 5 0 1 5  \( X(3, 5) = 4 \)  \( Z(3, 5) = 141 \)

totalWeight: 151  totalTime: 141

Current Path Nodes: 5 0 1 5 4  \( X(5, 0) = 3 \)  \( Z(5, 0) = 145 \)

totalWeight: 157  totalTime: 145

Current Path Nodes: 0 2 5 2 4  \( X(0, 2) = 2 \)  \( Z(0, 2) = 151 \)

totalWeight: 161  totalTime: 151

Current Path Nodes: 2 3 3 2 4  \( X(2, 3) = 1 \)  \( Z(2, 3) = 157 \)

totalWeight: 165  totalTime: 157

Total Weight = 165
Total Time = 157
Time to complete program = 0.001468 seconds
Output for Max Weight – 20 Node Problem:
Path: (0, 10, 1, 18, 0, 12, 2, 4, 11, 10, 13, 1, 9, 2, 13, 8, 17, 18, 13, 6, 0, 8, 4, 15, 7, 9, 0, 14, 1, 19, 14, 18, 3, 10, 5, 11, 2, 10, 7, 13, 15, 11, 9, 16, 7, 0, 19, 10, 18, 19, 16, 5, 12, 7, 1, 8, 14, 9, 18, 4, 16, 8, 7, 11, 3, 12, 15, 6, 10, 16, 6, 11, 19, 4, 17, 6, 1, 0, 3, 2, 16, 15, 8, 19, 5, 13, 11, 17, 16, 12, 8, 10, 4, 7, 17, 0, 10, 1, 18, 0, 12, 2, 4, 11, 10, 13, 1, 9, 2, 13, 8, 17, 18, 13, 6, 0, 8, 4, 15, 7, 9, 0, 14, 1, 19, 14, 18, 3, 10, 5, 11, 2, 10, 7, 13, 15, 11, 9, 16, 7, 0, 19, 10, 18, 19, 16, 5, 12, 7, 1, 15, 2, 6, 3, 7, 2, 5, 6, 9, 12, 10, 17, 13, 4, 6, 14, 2, 1, 5, 0, 16, 14, 5, 9, 3, 12, 14, 9, 18, 12, 11, 14, 7, 5, 18, 4, 9, 8, 18, 15, 3, 5, 17, 14, 4, 12, 1, 8, 11, 16, 8, 2, 3, 8, 5, 15, 14, 3, 13, 16, 3, 19, 15, 10, 14, 13, 19, 2, 18, 16, 1, 3, 4, 11, 18, 7, 15, 4, 0, 11, 17, 15, 0, 17, 2, 4, 17, 12, 13, 0, 6, 17, 19, 4, 5, 11, 6, 18, 17, 1, 6, 19, 11, 1, 0, 2, 12, 6, 8, 4, 1, 10, 9, 17, 3, 10, 13, 9, 19, 12, 7, 19, 14, 8, 13, 18, 14, 1, 18, 10, 9, 15, 13, 7, 6, 13, 2, 9, 18, 19, 10, 7, 9, 14, 0, 9, 8, 10, 0, 8, 7, 0, 12, 8, 17, 13, 1, 19, 8, 16, 19, 0, 18, 3, 11, 7, 1, 9, 6, 10, 2, 11, 4, 10, 14, 7, 15, 10, 17, 7, 18, 8, 1, 2, 7, 16, 2, 4, 15, 0, 6, 2, 18, 4, 7, 3, 6, 15, 1, 6, 14, 2, 15, 8, 3, 15, 12, 9, 16, 15, 11, 0, 1, 10, 3, 0, 13, 10, 9, 13, 14, 8, 13, 18, 14, 19, 7, 12, 14, 1, 18, 10, 11, 9, 15, 13, 7, 6, 13, 2, 9, 18, 17, 14, 0, 9, 7, 10, 0, 8, 7, 0, 12, 0, 16, 6, 11, 13, 1, 19, 0, 18, 6, 8, 10, 16, 8, 9, 1, 7, 11, 8, 2, 10, 6, 9, 14, 10, 15, 7, 14, 15, 18, 3, 1, 8, 19, 10, 5, 6, 19, 3, 13, 17, 8, 4, 7, 3, 6, 0, 15, 1, 6, 15, 19, 18, 7, 16, 18, 2, 7, 5, 18, 4, 15, 3, 2, 4, 11, 2, 16, 12, 3, 0, 1, 10, 3, 8, 15, 2, 6, 12, 10, 13, 8, 14, 1, 2, 19, 16, 15, 12, 7, 6, 14, 2, 13, 6, 17, 10, 7, 9, 2, 5, 14, 18, 9, 16, 14, 12, 8, 7, 13, 18, 10, 11, 9, 12, 13, 15, 9, 8, 10, 0, 18, 1, 19, 14, 0, 7, 1, 9, 19, 0, 8, 16, 10, 15, 7, 11, 19, 10, 6, 9, 10, 14, 9, 0, 12, 1, 8, 19, 7, 14, 17, 18, 7, 16, 6, 0, 13, 1, 6, 19, 12, 2, 10, 5, 12, 18, 19, 13, 17, 8, 4, 7, 3, 10, 1, 0, 2, 18, 3, 2, 7, 5, 13, 10, 12, 3, 14, 1, 2, 6, 18, 4, 13, 8, 14, 2, 13, 6, 12, 7, 9, 2, 4, 19, 16, 2, 11, 6, 14, 18, 9, 13, 18, 10, 7, 13, 11, 8, 2, 19, 17, END)
Total Weight = 5545
Total Time = 5000
Time to complete program = 4.887014 seconds

Output for Max Weight – 21 Node Problem:
Total Weight = 9048
Total Time = 7998
Time to complete program = 12.942527 seconds

Output for Max Weight – 22 Node Problem:
Total Weight = 9165
Total Time = 8000
Time to complete program = 18.958577 seconds

Output for Max Weight – 23 Node Problem:
Total Weight = 9543
Total Time = 8000
Time to complete program = 29.323686 seconds
References


