Parallelization of the set partitioning method for optimally solving the vehicle routing problem

Stephanie Marie Wolf

Follow this and additional works at: https://trace.tennessee.edu/utk_gradthes

Recommended Citation
https://trace.tennessee.edu/utk_gradthes/10987
To the Graduate Council:

I am submitting herewith a thesis written by Stephanie Marie Wolf entitled "Parallelization of the set partitioning method for optimally solving the vehicle routing problem." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

Michael Leuze, Major Professor

We have read this thesis and recommend its acceptance:

Chuck Noon, Brad Vander Zanden

Accepted for the Council:

Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)
To the Graduate Council:

I am submitting herewith a thesis written by Stephanie Marie Wolf entitled Parallelization of the Set Partitioning Method for Optimally Solving the Vehicle Routing Problem. I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

Michael Leuze, Major Professor

We have read this thesis and recommend its acceptance:

Charles E. Moore

Accepted for the Council:

Lawrence Minkel

Associate Vice Chancellor
and Dean of the Graduate School
Parallelization of the Set Partitioning Method for Optimally Solving the Vehicle Routing Problem

A Thesis

Presented for the

Master of Science Degree

The University of Tennessee, Knoxville

Stephanie Marie Wolf

May 1996
Acknowledgments

I would like to thank Mike Leuze for being my main official advisor and discussing the project with me along the way. I would like to thank Chuck Noon for being my secondary advisor from the department of Business Management. Thanks for the informational meetings and the materials and the suggestions provided. Thanks to Dr. Brad Vander Zanden for being on my committee and giving me his time. Thanks also to Christian Halloy for serving as my unofficial advisor in this project. Thanks for listening, advising, and helping me keep to a schedule with this project.

Thanks also to my family and friends who have encouraged and pressed me along the way with comments like, "I cannot understand why you are not finished with your thesis yet."

Thanks especially to my fiance, Sean Houmes, who encouraged me to follow my dream and who tried to motivate me when motivation did not seem to come easily for me. Thanks Sean, for helping me finally get here. I love you dearly.
Abstract

This paper will present the Vehicle Routing Problem in its basic form as well as describe the particular type of problem the solution method addresses. The Set Partitioning method has been used to solve this problem optimally. The Set Partitioning method consists of three steps. The serial versions of code in each step could be parallelized to obtain a speedup at each step, and an overall speedup for the whole project.

The second step, which this paper focuses on, has been parallelized for this project and will be described in greater detail. Parallelization strategies and considerations will also be discussed.

The third step of the Set Partitioning method, which involves integer programming, may simply require the use of an efficient parallel integer program solver for speedup in this final step. Complexity of each step will be presented as well as ideas for future research.
Contents

1 Introduction to the Vehicle Routing Problem (VRP) 1
   1.1 Traveling Salesman Problem (TSP) ....................... 2
   1.2 Types of Vehicle Routing Problems (VRPs) ................ 3
   1.3 Assumptions made for this specific VRP .................. 6
   1.4 Complexity of the VRP .................................. 8
   1.5 Mathematical Representation of the VRP .................. 9
   1.6 Types of Solution Methods ............................... 11

2 Different Approaches to Solving the VRP 14
   2.1 Exact Algorithms ....................................... 15
   2.2 Heuristics .............................................. 15
       2.2.1 Savings or Insertion Methods ......................... 16
       2.2.2 Cluster First, Route Second .......................... 16
       2.2.3 Route First, Cluster Second .......................... 17
       2.2.4 Improvement or Exchange Procedures ................. 17
2.3 Combination Methods ........................................... 18
  2.3.1 Mathematical Programming Coupled With Branch and Bound 18
  2.3.2 Set Partitioning Method ................................... 18

3 The Set Partitioning Method .................................... 19
  3.1 Set Partitioning - Step 1 .................................... 21
  3.2 Set Partitioning - Step 2 .................................... 22
  3.3 Set Partitioning - Step 3 .................................... 23
  3.4 Optimality of the Set Partitioning Method ................. 23

4 Introduction to Mathematical Programming and Important Terms 25

5 Introduction to Enumeration Trees .............................. 28

6 Explanation of the Serial Code ................................. 31
  6.1 Overview of the Serial Code ................................ 31
  6.2 Important Algorithms in the Serial Code ................. 33
    6.2.1 rreduce ................................................. 33
    6.2.2 onetree ................................................ 34
    6.2.3 Heuristic4 ............................................. 34

7 Parallelization of the Code .................................... 35
  7.1 Motivation for Parallelization of the Code ............... 35
  7.2 Motivation for Using Parallel Virtual Machines(PVM) .... 36
7.3 Motivation for using PVM over a small network .......... 36
7.4 Motivation for using PVM over the IBM SP2 .......... 37
7.5 Issues involved in parallelizing code .................. 37
7.6 Serial Enumeration Trees into Parallel Pruning ........ 38
7.7 Parallelization Strategies for this Code .............. 39
    7.7.1 First Ideas ..................................... 39
    7.7.2 Next Ideas ....................................... 44
    7.7.3 Best Ideas ....................................... 52
    7.7.4 Reordering Might Prune More Routes ............... 54

8 Testing of the Parallel Codes and Results ............. 59
     8.1 Why Parallel Methods 1 through 3 should NOT be used .... 59
     8.2 Why Parallel Methods 4 and 5 should NOT be used ........ 60
     8.3 Why Parallel Methods 6 and 7 should NOT be used ........ 60
     8.4 Serial versus Parallel versions 10 and 11 .............. 61
     8.5 Cities Must Be Ordered Min to Max ..................... 61
     8.6 Serial versus Parallel version 9 ....................... 62
     8.7 The Best Method - Serial versus Parallel version 8 ....... 62
     8.8 Comparing PVM code over a LAN versus PVM code on the SP2 . 63
     8.9 Different numbers of Processors on the IBM SP2 .......... 64
     8.10 Conclusions ....................................... 64
9 Areas for Future Research

Bibliography

Appendices

Appendix A - Serial Code ........................................ 72
Appendix B - Best Parallel Code .................................. 126
Appendix C - Test Cases ............................................. 190
Appendix D - PVM Hints ........................................... 213
Appendix E - SP2 Hints ............................................ 218
Appendix F - World Wide Web Addresses ...................... 221

Vita .............................................................. 223
List of Tables

7.1 Deviance in the Number of Routes .................................. 55

8.1 Serial versus Massive Communication Parallel Codes ............. 60
8.2 Serial versus Non-Pruning Parallel Codes ............................ 61
8.3 Serial versus Pruning Parallel Codes ................................. 63
8.4 Serial Time on a SPARC5 versus the SP2 ............................ 64
8.5 Speedup of PVM over 4 Processors on the SP2 ...................... 65
List of Figures

5.1 An explanation about spanning trees .......................... 29

6.1 trees ......................................................... 32

7.1 Binary Tree with Number of Cities Levels .................. 41
7.2 \(2^{\text{Number of Cities}}\) Tree with 1 Level ................. 42
7.3 Loop and Spawn, Loop and Re-Spawn ....................... 43
7.4 Round Robin .................................................. 46
7.5 First Come First Served ....................................... 47
7.6 Divide the Work ................................................. 48
7.7 Every Nth Route ............................................... 50
7.8 Bad Load Balancing for Every Nth Route .................... 51
7.9 Good Load Balancing for Divide and Prune .................. 53
7.10 Divide and Prune and Sections ............................. 55
7.11 Good Load Balancing for Divide and Prune ................. 56
List of Important Abbreviations and Terms
<table>
<thead>
<tr>
<th>VRP</th>
<th>Vehicle Routing Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSP</td>
<td>Traveling Salesman Problem</td>
</tr>
<tr>
<td>CVRP</td>
<td>Capacitated Vehicle Routing Problem (one with weight constraints for the trucks).</td>
</tr>
<tr>
<td>VRPWTW</td>
<td>Vehicle Routing Problem with time windows (at least some customers must be reached within a certain time frame.)</td>
</tr>
<tr>
<td>Gap</td>
<td>Maximum total time allowed for a route</td>
</tr>
<tr>
<td>lowerbound</td>
<td>The optimal travel time for a route solved using an exact TSP algorithm over the set of cities for a route.</td>
</tr>
<tr>
<td>dual variable</td>
<td>Value associated with converting a minimization problem to a maximization problem or a maximization problem to a minimization problem, whichever could be more easily solved to optimality.</td>
</tr>
<tr>
<td>nz</td>
<td>The reduced cost of a route, or the difference between the summation of dual variables for a route and that route's lowerbound.</td>
</tr>
<tr>
<td>reduced cost of a route (nz)</td>
<td>The difference between the summation of dual variables for a route and that route's lowerbound.</td>
</tr>
<tr>
<td>NumberofCities</td>
<td>The Number of Cities, not including the depot</td>
</tr>
<tr>
<td>NV</td>
<td>Number of Vehicles in the VRP</td>
</tr>
<tr>
<td>TimeLimit</td>
<td>The maximum amount of time allowed for a single route</td>
</tr>
<tr>
<td>primal problem</td>
<td>original linear programming problem</td>
</tr>
<tr>
<td>dual problem</td>
<td>a problem constructed by converting a primal into a different form</td>
</tr>
<tr>
<td>heuristic</td>
<td>an algorithm guaranteed to find a solution (if one exits) in a reasonable amount of time</td>
</tr>
<tr>
<td>exact algorithm</td>
<td>is guaranteed to find a solution, yet may run much longer than a reasonable amount of time</td>
</tr>
<tr>
<td>optimal solution</td>
<td>a best solution (Remember that sometimes there may be more than one best solution.)</td>
</tr>
<tr>
<td>route</td>
<td>an ordered cluster of cities visited by one truck</td>
</tr>
<tr>
<td>constraint</td>
<td>criteria for considering a route as a possible route in optimal solution to the VRP</td>
</tr>
<tr>
<td>feasible route</td>
<td>route which meets all of the constraints of the VRP</td>
</tr>
<tr>
<td>eligible route</td>
<td>feasible route which is also economical because its reduced cost is less than the Gap</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction to the Vehicle Routing Problem (VRP)

The basic vehicle routing problem generally refers to the optimal scheduling of several trucks to several different cities, or customers. Optimal scheduling usually refers to minimizing costs, which may often be a function of the total distance traveled by the trucks. This basic VRP could be thought of as two separate problems - dividing the cities among routes, and the ordering of cities on those routes. Once the cities are assigned to a route, the problem of ordering those cities on that route can be modeled as a Traveling Salesman Problem (TSP).
1.1 Traveling Salesman Problem (TSP)

Many people are familiar with a particular scheduling problem known as the traveling salesman problem (TSP). The traveling salesman problem can be represented by a graph in which the nodes, or vertices, in the graph represent cities, and the arcs, or edges, of the graph represent roads between the cities. A salesman wishes to start at his home city and visit every remaining city exactly once, and return to his home city. He would like to know which route has the least cost (which may simply be the least total Euclidean distance, \( D = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \)).

One could simply enumerate, or list, every possible route and then choose the best route. The problem with this method is that the number of routes increases significantly for each new city, or node, added to the problem. The number of possible paths in a completely connected graph which contains \( \text{NumberOfCities} \) nodes (which include the salesman’s home city) is \((\text{NumberOfCities} - 1)!\), or the \((\text{NumberOfCities} - 1) \ast (\text{NumberOfCities} - 2) \ast (\text{NumberOfCities} - 3) \ldots \ast 1 \ast 1\).

When the salesman begins his journey from one city, he has the choice of visiting one of \( \text{NumberOfCities} - 1 \). He then has \( \text{NumberOfCities} - 2 \) cities to choose from for his next visit. And, this continues until there is no choice as to which remaining city the salesman must visit because there is only one unvisited city remaining. Then, there is only one choice of where the salesman will go after all the unvisited cities have been visited - home. There are many different orderings
of cities the salesman could pick. He would like to optimize his cost by minimizing the total distance he travels.

The Traveling Salesman Problem, or TSP, has been shown to be an NP-Complete problem. [11] This means that for large cases, it is conjectured that the problem cannot be solved in polynomial time. In other words, as the problem size of the TSP slightly increases, the amount of time required to solve the TSP drastically increases. Suppose one has a fleet of vehicles (s)he has to dispatch to service customers or deliver or pick up goods. One would always like to be able to dispatch the vehicles in an optimal manner. The Vehicle Routing Problem (VRP) is basically finding an optimal solution to the distribution of vehicles in order to accomplish a particular goal (such as minimizing the total cost of all the routes).

1.2 Types of Vehicle Routing Problems (VRPs)

Although the basic definition of the VRP is one with a single depot and multiple vehicles, there are several types of VRPs. The first type is often referred to as the Chinese Postman Problem. This type of Vehicle Routing Problem (VRP) involves one postman covering all arcs of a representative graph for a problem. For example, suppose one must dispatch one or several vehicles to accomplish the street-cleaning in a town.

The capacitated Chinese postman problem is a slight variation on the Chinese
postman problem. In this problem, there is a fleet of vehicles (rather than just one postman) which must cover every arc of the representation graph.

There are also the multiple traveling salesman problems (m-TSP). These involve several salesman from different cities. All other cities in the problem must be visited once by one of the salesman. Each salesman’s home city is visited twice by that particular salesman. This problem could also be thought of as a multiple depot with a single vehicle per depot type of problem.

Another type of Vehicle Routing Problem is referred to as the multiple depot, multiple vehicle problem. This type of problem involves multiple depots or warehouses each dispatching one or several truck(s) to deliver a demanded quantity of their product to the customers. Suppose one owns a medical supply company which has several warehouses which are responsible for the delivery of a known quantity of a particular medical supply to several different hospitals and pharmacies. One wants to find the least cost method of delivering its supplies to all of its customers (or demand points). One must contemplate which warehouse should service each particular customer and decide which routes would minimize the total distance covered by routes at each particular depot.

Sometimes a person may not know a priori, or beforehand, how much of its product a customer will request. Suppose a local distributing company has one warehouse which distributes alcohol to local bars, restaurants, and package liquor places. The dispatcher, or person creating the routes for the vehicles, must use
probability and a weight demand history from each customer to determine the probable demand from each customer. Then, routes can be determined based on customer location and estimated customer demand. This type of problem is called single depot multiple vehicle using stochastic demands. "Stochastic" refers to probability methods used to determine the product demand for each customer.

The type of vehicle routing problem presented in this paper is simply referred to as the single depot multiple vehicle problem. This type of VRP is also considered the basic VRP. The customer demands are known a priori, or before the routes are created. An example of such a problem is a food supplier shipping requested food amounts to several different grocery stores. Another example is a warehouse which ships a predetermined amount of sheet metal to several different sites. This problem requires no probability methods to determine the weight demand at each customer because their demands are already known. Thus, this problem contains less uncertainty than the single depot with stochastic methods. Also, this problem is slightly easier than the multiple depot, multiple vehicle problem because the distribution of customers per depot has already been determined. All customers in the problem are serviced by the only depot in the problem. The single depot, multiple vehicle problem could be thought of as the last step in the solution to a multiple depot, multiple vehicle problem. First, distribute the customers by depot. Then, solve the single depot, multiple vehicle problem over each depot. (Note, however, that there are other methods of solving a multiple depot, multiple
vehicle problem.)

1.3 Assumptions made for this specific VRP

First of all, this specific type of VRP tends to require many considerations. These might include the number of vehicles, the number of drivers, whether all trucks have the same capacity, maximum number of cities on a route, maximum distance of a route, maximum time allowed for a route, time windows in which certain customers have to be reached, whether multiple day trips are allowed, overtime wages for drivers, and whether the weight demands are known before the routes are created. Not all of these considerations are addressed in the VRP presented within this paper.

Specific Considerations for this VRP are:

1. There is one depot and multiple trucks.

2. Each truck has a maximum capacity, or weight limit. Thus, this problem is a capacitated VRP. And, each truck has the SAME capacity.

3. A minimum weight is imposed on each truck in order to assure that all weight demands of the customers can be met. \( \text{MinW} = \text{Max}(\text{TotalW} - (\text{NV} - 1) \times \text{MaxW}, 0) \).

4. There are no customers which HAVE to be reached within a certain time frame. Thus, there are no time windows for this VRP. However, there IS
a TimeLimit for the entire route. The total time route time consists of the optimal route time (distance), or lower bound, and the amount of time required to stop at each city. In this VRP, the stopping time is a constant. Total route time must be less than or equal to the TimeLimit for the route to be feasible.

5. The weight demand of each city, or customer, is known a priori, or before the routes are determined.

6. No customer requests more weight than the maximum weight limit of a truck. And, the VRP does not contain more weight than can be carried by all of the trucks together.

7. Trucks do not make multiple day trips from the depot to the customers. Thus, one truck is associated with exactly one route (which is a feasible route). A feasible route is a grouping and ordering of cities which meets all constraints of the VRP.

8. Every truck has 1 route through at least one city.

9. The distances between cities are assumed to be symmetric. (The distance from A to B is considered to be the same as the distance from B to A.) Often in real world situations, the distances between two places may not be symmetric due to congestion or one way streets.
Finding all feasible routes requires first determining the total weight for the route. Does the weight for the set of cities which will make up the route meet the minimum and maximum weight constraints? If both weight constraints are met for a cluster of cities, a TSP is applied to the set of cities to order the route and determine the optimal distance traveled by the truck on that route. That distance, or lower bound, is then used to calculate the total route time. Does the total route time meet the TimeLimit constraint? If the time constraint for the route is also met, an economic consideration is If the answer is "yes," the route is a feasible one and the reduced cost of the route is computed and compared to the Gap variable. The reduced cost of a route, or nz value, is the lower bound of the route subtracting the sum of the dual variables corresponding to the customers on the route. If the reduced cost exceeds the gap, the route is deemed uneconomical and disregarded, so the algorithm begins checking another route. This continues until all feasible routes have been determined.

1.4 Complexity of the VRP

The Vehicle Routing Problem could be thought of as a two-part problem in which one first decides which truck should visit which cities and then finds the optimal route for that truck through those cities. The second portion of this problem, as stated here, is, in actuality, a TSP (which finds the optimal ordering of cities as
well as the total cost of a route). If one solved this first part of the VRP in a simple exact manner - listing all possible routes, determining feasibility of those routes, determining the optimal costs of the feasible routes, and then grouping routes together such that every city, or customer, is on exactly one route, the complexity of the VRP would be $2^{NumberofCities} - 1$, where NumberofCities is the number of cities in the VRP not including the depot, and the route with no cities on it is not even considered. The time this algorithm would take to run is some function of $2^{NumberofCities}$. Note that as the NumberofCities variable grows slightly, the time this algorithm requires to run greatly increases. Remember that the second part of this algorithm is a TSP, whose complexity is $N!$, with a relatively small $N$ (approximately NumberofCities/N).

1.5 Mathematical Representation of the VRP

Below is a mathematical representation of the VRP along with an explanation of the mathematical equations and the notation used.

The variables which will be used within the mathematical representation are:

$C_j = \text{Total cost, or Euclidean distance, of route } j.$

$X_j = \text{Binary vector of cities which contains a 1 in a position corresponding to a city visited on route } j \text{ and contains a 0 in that position if the city is NOT visited on route } j.$
$S_j$ = Binary variable which is 1 if route $j$ is in the solution. Otherwise, $S_j$ is 0.

NC = Vector containing all 1's in positions 1 through the number of cities

W = weight demand vector

Max = Maximum weight bound on a truck

Min = minimum weight bound on a truck

STOPTIME = Vector containing the constant amount of stopping time required at a city. There are NumberofCities positions in this vector. TimeLimit = The total amount of time a route can take

Now, the Vehicle Routing Problem can be stated mathematically in the following manner:

Minimize

$$\sum_j C_j \cdot S_j$$

subject to

$$\sum_j S_j \cdot X_j = NC$$

and

$$\forall j, S_j \in \{0,1\}$$

With Constraints

$$Min \leq W \cdot X_j \leq Max$$

$$C_j + STOPTIME \cdot X_j \leq TimeLimit [22]$$
In other words, "minimize" the total cost of all of the routes in the solution "subject to" each city must appear on exactly one route in the solution, and each route is represented by a binary array such that the position in the array is a 1 if the city is visited by the route and the position in the array is 0 if the city is not visited on that particular route.

In order for a route to be feasible, it must first meet the minimum and maximum weight constraints of a truck as well as meet the route time (TimeLimit) constraint. [16]

1.6 Types of Solution Methods

There are mainly two types of solution methods for NP-complete problems such as the Traveling Salesman Problem (TSP) and the Vehicle Routing Problem (VRP). It is conjectured that NP-complete problems cannot be solved in polynomial time for larger cases. These two solution methods are 1) exact methods and 2) heuristic(good) methods. The exact methods will return an actual optimal solution to a problem if one exists. However, these methods require much time, and as previously stated, the number of possible, not necessarily feasible, routes for any given VRP is $2^{\text{Number of Cities}} - 1$. (The non-route, or route which visits no cities at all, need not be considered).

The heuristic methods, on the other hand, run in a more reasonable amount
of time and return a solution. Good heuristics tend to return good solutions. Perhaps a good solution returned by a good heuristic may actually be an optimal solution. But, an exact method would need to be used to determine this and the exact method could be very time-consuming. A VRP may be too large to find an exact solution within a given time frame, or even a lifetime, for that matter. So, which type of solution is better? Good heuristics are used to find a good solution to a given problem in real time. And, often the good solution returned is good enough for the problem at hand.

But, ideally, one would like to obtain an optimal solution to a particular problem in a relatively short amount of time. For NP-Complete problems, it is conjectured that there will always be instances of the problem for which finding an optimal solution in a relatively short amount of time is impossible. Combination methods, which have been created in an attempt to do just that, could be considered a third type of solution method.

Part of the vehicle routing involves graph covering. This means that every customer must be on at least one route. However, set-partitioning is also involved. Every customer also must be on only one route. Thus, each customer must be on exactly one route. No customers can be left out of the solution to the VRP. A Set Partitioning method, which is a combination method, has been posed which optimally solves the VRP in a relatively short amount of time. This Set Partitioning Method will be described in greater detail later. The serial code which imple-
mented this method, although quicker than most exact methods, was still taking over two weeks to optimally solve a problem with 50 cities and 3 trucks (with a large Gap variable) when run on a fast workstation, the DEC Alpha! There has to be a better way! Thus parallelization was needed to divide the work among processors and thus improve the performance of this algorithm even more. One could hope for speedup which would scale with the number of processors used. This means that with N processors, one could ideally hope for the parallel code to run N times as fast as the serial code ran on one processor.
Chapter 2

Different Approaches to Solving the VRP

Here are several approaches to solving the VRP. These fall into the three method types previously mentioned. The exact methods give the optimal solution in a large amount of time for large cases. Often the complexity of an exact algorithm is $2^N$, where $N$ is the number of cities (not including the depot from which the trucks are dispatched). These methods may be fine for smaller data sets. However, on larger data sets, the algorithms may run for years and still not finish. The good heuristic methods give good, but not necessarily optimal, solutions in a much shorter amount of time. The problem with heuristics is that they MAY NOT obtain an optimal solution or even be close to one. Therefore, combination methods have been created in an attempt to achieve an optimal solution in a
relatively short amount of time. Thus, these seem to obtain the best of both worlds.

2.1 Exact Algorithms

Again, exact solutions take an extremely long time to run for larger data sets. However, if and when the algorithm actually finishes, one is guaranteed to have an optimal solution. (There may, in some cases be more than one optimal solution. But, one will be found eventually if one exists). Some exact algorithms include Branch and Bound, Dynamic Programming and Cutting Plane algorithms. More information on these methods can be found in Christofides, et. al. [17] These exact solutions are not as popular as the heuristic methods described below.

2.2 Heuristics

Heuristic algorithms take less time to run than the exact algorithms and produce a solution. A good heuristic tends to return a good solution. Even though that good solution MIGHT actually be the optimal one, one would have to run an exact algorithm to prove that the good solution actually is optimal. Often good solutions are good enough (especially considering the amount of time required to determine a best, or optimal solution for the problem).

The following are a few heuristic algorithms used for solving the VRP.
2.2.1 Savings or Insertion Methods

The Savings, or Insertion, Methods search the list of unassigned cities in order to find the best (or often, nearest,) one to add to the current route one is creating. If adding the city to the route will not cause the weight of the route to exceed the maximum weight constraint for a truck, add the city to the route, and remove the city from the list of unassigned cities. When no more cities can be added to a route without exceeding the maximum weight constraint for a truck, consider this route complete and begin creating the next route from the remaining cities. Continue until all cities have been assigned to a route.

Those who are more familiar with the Traveling Salesman Problem may notice that this method tends to mimic the nearest neighbors algorithm for solving the TSP. (One begins at the salesman’s home city and travels to the closest unvisited city, and from that city, travels to the closest unvisited city. One continues in this manner until all cities have been visited and then returns to the home city.)

2.2.2 Cluster First, Route Second

This method first partitions the destinations, or cities, into clusters based on the distance between the cities or some other important measure of cost. One must make sure none of the clusters will exceed the maximum weight of a route. A destination or two may have to be shifted to another cluster if the total weight demand of the cluster is more than the maximum weight a truck can carry. Then,
the clusters are treated as miniature TSP problems, and a good route is created over each cluster.

2.2.3 Route First, Cluster Second

This method first finds one good ordered route as if there were only one route through all of the cities, or if this were a single TSP. Then, this one good ordered route is divided into NV smaller routes where NV is the number of vehicles and the smaller ordered routes are under the maximum weight of a truck. If a route exceeds the maximum weight of the truck, then one or two of the destinations, or cities, will be shifted to an adjacent route (one next to the route of the city being shifted).

2.2.4 Improvement or Exchange Procedures

Improvement, or exchange, procedures first find a solution. Then improvement costs are evaluated. Then, routes may exchange cities depending on the amount of cost improvement acquired by an exchange of cities. The exchanges or improvements tend to occur over a given number of time steps to produce at least a good solution.
2.3 Combination Methods

As previously mentioned, combination methods are a means of obtaining an optimal solution in a relatively small amount of time.

2.3.1 Mathematical Programming Coupled With Branch and Bound

There are several general methods which use mathematical programming and Branch and Bound Algorithms. These have been proven to solve the VRP optimally. Branch and Bound algorithms get their name from branching to a certain part of a solution and solving for an upper or lower bound on that portion of the solution. A few mathematical programming concepts will be explained in this paper.

2.3.2 Set Partitioning Method

The Set Partitioning Method assumes there is a graph representing cities, or customers, joined together by arcs (highways, dirt roads, over the river by ferry). The graph is partitioned, or divided up into NV (number of vehicles) subgraphs. One truck is assigned to each set of customers, or subgraph. The ordering of the cities visited is solved as a TSP. This particular combination method of solving the Vehicle Routing Problem (VRP) is the focus of this paper and will be discussed in greater detail in the following chapter.
Chapter 3

The Set Partitioning Method

As previously stated, the Set Partitioning Method is a combination method. First, a good heuristic is used to obtain a good, but not necessarily optimal solution for the VRP. Then, that solution is tested and either found to actually be optimal, or other routes are found that should be considered. If additional routes are added, an Integer Program (IP) is run over the set of routes to obtain an optimal solution.

The reason for using the Set Partitioning Method is that it gives an actual optimal solution in a reasonable amount of time. (that is, compared to exact methods).

The Set Partitioning Method consists of three main steps. Step 1 requires using a good heuristic to obtain a good solution and then using linear programming to obtain the initial dual variables and the gap for the second step of this method. The complexity of the heuristic may vary. But, most heuristics tend to be polynomial in nature.
The heuristic is a column based generation scheme with insertion and deletion. One begins with a set of columns. New columns are added if they are found to possibly contain a route that may be in an optimal solution. When the stopping condition is met, if the stopping condition was large enough, the optimal solution will be returned. Otherwise, if one finds a reduced cost (nz value) lower than -1 or -2 in step two, this indicates that the stopping condition had not been correct in the first step. However, the LP usually produces solution variables as rational numbers - not integers. That is why the integer program (IP) must be run in step 3. For more specific information on the column generation scheme and/or the Set Partitioning Method, see “A Set-Partitioning-Based Exact Algorithm for the Vehicle Routing Problem” by Yogesh Agarwal, et. al. [22]

The complexity of integer and linear programming depend on the number of solution variables as well as constraint variables. Also, the complexity of solving an LP or IP will depend partially on the method of solving those equations.

Step 2 of the Set Partitioning Method, the main focus of this paper, uses the calculated dual variables and the gap produced in step 1 to find all feasible routes which meet both weight constraints as well as the total route time (or TimeLimit) constraint. The minimum cost of each feasible route is also found. The total number of possible routes is of order $2^N$. Thus, this step is very time-consuming. Parallelization of this portion has been used to distribute the work among several processes and thereby decrease the run time of the code. Note that
the algorithm is still of the order $2^N$. The parallelization of the code is an attempt to speed up the serial code. However, as mentioned before, some problems may not be solvable in a *reasonable* amount of time.

Step 3 of the Set Partitioning Method is to use integer programming.

Once again, the complexity of integer programming is dependent upon the number of solution and constraint variables in a given problem.

### 3.1 Set Partitioning - Step 1

In step 1, a heuristic is used in an attempt to achieve a good solution. The chosen heuristic method was linear programming and column generation using an insert/deletion method as described above.

After a good solution has been found, a linear program solver is run on the set of routes to obtain the dual variables as well as the gap. The dual variables of a Linear Program (LP) are those which can be used to convert a problem solving for a minimum value to a problem solving for a maximum value OR vice-versa. These dual variables will be utilized in the second step. They are useful in determining the reduced cost of a route. The gap is the maximum reduced cost, or gap between the lower and upper bounds of a problem. If a route's reduced cost exceeds the gap, the route is considered uneconomical and ineligible to be in an optimal solution for the VRP. This gap can be used to narrow down the
number of eligible routes. The total time taken on a route is the amount of time
required to stop at one city multiplied by the number of cities on the route plus
some function of the distance traveled. Thus, the total time of a route is the total
stop time of the route added to the total travel time of the route.

3.2 Set Partitioning - Step 2

Given are the list of city (and depot) coordinates, city (and depot) weight de-
mands, city (and depot) dual variables and the time gap for any route. Now, one
must find all feasible, eligible routes. These are routes which meet the weight re-
quirements for the truck as well as meet the route time (or TimeLimit) constraint
for a route. Yet, for a route to be considered part of the optimal solution, the total
reduced cost of the route must be less than the given Gap. Total reduced cost is
the total cost of a route, or lowerbound, minus the sum of the dual variables for
the route.

This paper deals with this step 2 of the Set Partitioning Method. The serial
and parallel approaches to finding all feasible routes are presented later in this
paper.

If no extra route is generated that had not already been produced by the
heuristic in Step 1, Step 1 had ACTUALLY produced the optimal solution! If,
on the other hand, a route is produced that was not in the solution given by
the heuristic in Step 1 of the Set Partitioning Method, that heuristic had NOT produced an optimal solution. The new route should be added in to the old set of routes produced by the heuristic.

3.3 Set Partitioning - Step 3

After all feasible routes have been determined, run an Integer Programming (IP) solver over the set of routes to solve the Set-Partitioning (SP) portion of the problem and thus produce an optimal solution to the VRP.

3.4 Optimality of the Set Partitioning Method

The theorem and corollary presented here are based on a well-known result from Pierce and Lasky. [20]

- **Theorem** - In a Set Partitioning Problem, the cost coefficients, $c_j$ are replaced by the reduced costs $r_j = C_j - u \cdot X_j$, where $u$ is the dual vector, the Set Partitioning Problem minimizing the reduced costs will have the same optimal solutions as the Set Partitioning problem which minimizes the total cost. The reduced costs are obtained when the LP is run in step 1 of the Set Partitioning Method. Note that these variables will be non-negative because they must be non-negative in order for the LP solution to be optimal. [22]
- Corollary - Suppose LB is the value of the optimal solution produced by the LP and UB is a known upper bound on the Set Partitioning solution. The optimal solution to the Set Partitioning Problem cannot contain any column with a reduced cost greater than UB - LB. UB - LB is the gap. When all columns with the reduced cost less than or equal to the gap (after the LP has been run and the dual variables which make up the u vector are given) and the Set Partitioning Problem is solved over the columns which were generated, the resulting solution is the optimal solution to the VRP. [22]
Chapter 4

Introduction to Mathematical Programming and Important Terms

This chapter will give a very brief introduction into mathematical programming. The focus will be on linear and integer programming.

A linear program is comprised of these components: an objective function, decision variables, and constraints.

The objective function is the optimization problem. This can either be a maximization or minimization problem. Decision variables are the different variables which are set in an attempt to obtain the objective. Constraints are conditions which limit the problem. [6]
A linear program could be defined in this manner:

Optimize: \( \max Y = 50X - 20Z \)

Subject to: \( X \leq 30 \)
\( Z \geq 2 \)
\( 2X = Z \)

and, \( X + Z \leq 60 \) [6]

The original linear program (LP) is also known as the primal. The dual of an LP is related to the LP in the following manner. The dual has as many decision variables as the primal has constraints; the dual has as many constraints as the primal has decision variables. If the primal involved maximization, the dual involves minimization. If the primal involved minimization, the dual involves maximization. Primals and duals come in pairs. The dual of a dual of a primal is the primal. [6]

A dual vector contains the values necessary to turn a maximization problem to a minimization problem or vice versa.

The dual values are used to determine an upper bound for the VRP. The lower bound, or total optimal cost of a route, is found using a TSP over the cities to be included in a route. The \( nz \) value, or reduced cost of a route will be the lower bound minus the summation of the dual values corresponding to each of the customers on a route.
The Gap, which is given, is the maximum gap between the upper and lower bounds for a VRP. The nz value of a route in the VRP MUST be less than or equal to the Gap variable. Otherwise, the route is considered ineligible. If the Gap is rather large, more feasible routes will also have appropriate nz values and remain eligible. Reducing the Gap, however, will limit the number of routes returned by Step 2 of the Set Partitioning Method.

Integer programming is any mathematical programming (including linear programming) which requires the decision variables to have integer values. [9] Since whether a city is on a route or not must be a 1 or a 0 and whether a route is in the Set Partitioning solution must be a 1 or a 0, ultimately an integer program (IP) must be run over the set of routes to solve the (SP) portion of the VRP and thus produce an optimal solution to the VRP.
Chapter 5

Introduction to Enumeration Trees

Enumeration trees could also be referred to as decision trees. At every step in the algorithm (or in the tree) before the algorithm stops (or a leaf node has been reached in the tree), a decision must be made. The number of choices can be represented as a number of branches extending from the node where the decision was being made. Each node represents all decisions made up to that point in the tree.

The Enumeration trees used in the serial algorithm are binary, which means there are two options represented by either a 1 or a 0 at each decision. A 1 represents a decision to place the city on the route. A 0, on the other hand, represents a decision not to place the city on the route. There is no other choice -
a city is either on a route or NOT on that same route. Each route is represented by a binary array (containing only 1’s and 0’s). Each leaf, or terminal, node represents an entire route.

The trees can be used in order to explain the concept of pruning, or cutting out, some branches which will not even need to be considered because the routes on those branches will not be feasible due to weight constraints. (Figure 5.1)

Figure 5.1: An explanation about spanning trees

(In figure 5.1, A is the parent node of B and C and is an ancestor to every node in the tree except itself. All the other nodes are considered descendents of...
A. The interior nodes are B, C, D, E, F and G. The leaf, or terminal nodes are H through O. B-D-E-H-I-J-K is a subtree of the entire tree.

When a node and its ancestors in the tree have already exceeded the maximum weight capacity for a truck, its descendent nodes do not need to be checked because most of the nodes children will have heavier routes. One of the child nodes at each sub-level will have the exact same weight as the ancestor node has. Skipping the routes which one knows will be infeasible before they are checked is referred to as pruning the tree. Pruning the tree will help one’s application run quicker since many branches and leaves will never have to be considered.
Chapter 6

Explanation of the Serial Code

6.1 Overview of the Serial Code

The serial code was a composite program created by Dr. Charles E. Noon. This code ordered the cities by weight demand from minimum to maximum weights. A Euclidean distance matrix is then calculated for the entire set of cities. The distances are assumed to be symmetric. The distance from A and B is the same as the distance from B to A. (Although this would not necessarily have to be the case.) The theory is to travel down a binary Enumeration tree with branches of 0 (do not include the city on the route) and 1 (do include the city on the route). Backtracking through the tree occurs whenever a route becomes infeasible or ineligible. However, the serial code actually checks the routes in a binary fashion setting the last position (or city with the maximum weight first). Routes which
do not satisfy the minimum truck weight are thrown out of consideration. Routes over the maximum weight capacity of a truck are thrown out of consideration and are the basis for pruning, or cutting out, a portion of the tree. (Figure 6.1)

Suppose the route at node D is already above the maximum weight for a truck. Prune at D because all of D’s children are at least as heavy as D.

D represents route 1 1 .....  
D would be the last node checked in that branch of the tree. 

Figure 6.1: trees

If a route falls within the weight bounds of the trucks, a call is made to onetree. onetree figures a minimum cost, or lower bound, for the route. If this minimum cost plus the stop time multiplied by the number of cities included on the route is less than the gap, call the subroutine Heuristic4. Heuristic4 figures a minimum cost for a route as well as orders the route near-optimally.
Again, calculate the total route time. If that time falls within the TimeLimit and the reduced cost is less than the gap, then the route is actually eligible to be in an optimal solution and should be printed into a file for further use in step 3.

### 6.2 Important Algorithms in the Serial Code

The Serial Code contains three slightly different algorithms that adjust the lower bounds on a route.

#### 6.2.1 rreduce

The rreduce algorithm was created by Charles Noon (University of Tennessee Knoxville) and Thomas Chan (SMU) in October 1989 and has been slightly modified by Stephanie Wolf in 1995. The algorithm produces lower bounds on the LP relaxation of the perfect 2-matching problem. The approach was based on “A Multiplier Adjustment Approach for the Set Partitioning Problem” by Thomas Chan and Candace Yano. (1988) The problem to be bounded is this: minimize $\text{Cost} \cdot X$ where $A \cdot X = 2$, and $0 \leq X \leq 1$.

$A$ is the node edge incidence matrix for the city graph. Cost is calculated using a NumberofCities by NumberofCities symmetric distance matrix. Dual feasible reduced costs are also calculated and copied into a matrix called Dist. The final vector of dual variables will be written into the array called “BestU”.

The rreduce algorithm is used once in the code to initially set the lower
bounds for the problem. onetree and heuristic4 are used in the beginning as well as later in the code to test the feasibility of any route which falls between the given weight constraints for a single truck.

6.2.2 onetree

onetree iteratively constructs a minimum spanning tree over a subset of customers by adding edges in a least cost next fashion. Then, the total cost of the spanning is calculated. This is a greedy algorithm which creates a lower bound to the Traveling Salesman Problem (TSP) portion of the VRP.

6.2.3 Heuristic4

Heuristic4 solves a TSP over a given subset of customers. This algorithm also includes a subroutine which finds a best route by first placing up to two cities on a route and then inserting each new city on the route into its best position on the route. The best position for the city is the one which incurs the least cost (or, in this case, distance). If the N-1 route was optimal, and N is inserted in its best position in the route, the resulting N route will also be optimal. This ordering is done to obtain the least cost, or lower bound, of a route.
Chapter 7

Parallelization of the Code

7.1 Motivation for Parallelization of the Code

The serial code was running for over two weeks when the input problem contained 50 cities and 3 trucks (and a large Gap variable) on a DEC Alpha workstation. Dividing up work among processors such that each processor’s work is independent of the other processors’ work, one should gain a speedup in the run-time for the code. This code easily lends itself to parallel application because each route can be checked separately, simultaneously. Whether one route in one subtree is found feasible has no bearing on whether another route from a completely separate subtree will be found feasible. However, remember that infeasibility due to exceeding the maximum weight limit of a route DOES have a bearing on the routes in that route’s own subtree. Routes can be pruned based on the infeasibility
Ideally, one would like linear speedup with the increase of processors. This means that with $N$ processors each doing $1/N$ of the work, the parallel code should run in $1/N$ of the time required for the serial code to run. However, this will not occur due to communications and other overheads. So, one attempts to get a good, speedup for the code.

### 7.2 Motivation for Using Parallel Virtual Machines (PVM)

Parallel Virtual Machines, PVM, is a well known software which is portable over many architectures. PVM is known as “The poor man’s parallel machine.” This software allows one to use several different workstations or other machines in a parallel manner through message passing routines and functions.

PVM code tends to be portable, which means that if a PVM application runs over a heterogeneous local area network, the application should also run over a parallel machine, such as the IBM SP2, with little adjustment made to the code.

### 7.3 Motivation for using PVM over a small network

This project is being done in conjunction with the University of Tennessee's Management Science program, which has a small cluster of machines. PVM will soon
be ported onto their machines. Then, the PVM could be used for future projects as well.

7.4 Motivation for using PVM over the IBM SP2

This researcher has previous experience on the IBM SP2 and was curious about the results of PVM on the SP2 compared to PVM over a heterogeneous network (one containing different types of machines). The SP2, or now, simply “SP”, is a cluster of RISC6000 processors most often connected by a high performance switch. PVM, being free-ware (software that is free), is often available on an SP2. Also often available is PVMe, which is IBM’s extended version of PVM. However, PVMe costs extra and may not be found on some SP2 machines, such as the 16 node SP2 at Oak Ridge National Labs. The PVM code created for the heterogeneous Local Area Network could be ported to the SP2 with ease. Not many changes were needed to run the same code in PVM over the SP2 after running the same PVM code over a heterogeneous LAN.

7.5 Issues involved in parallelizing code

There are several issues involved in creating parallel code. Three main issues are load balancing, communication versus computation, and speedup. Load balancing occurs when the processes share the work fairly evenly. One tries to balance the
load of his or her code effectively across the different machines. Excessive communication slows down parallel code. But, intensive computation which does not depend on or affect the results of the intensive computation on other processors, on the other hand, runs quite quickly in parallel. Sometimes the communication begins taking more time than the computation. Speedup refers to how much speed one gains through the use of more processors. The optimal number of processors varies with the code. Ideally, one would like a speedup of $N$, which would mean that distributing the work among $N$ processors would cause the parallel code to run $N$ times as fast as the serial code on one processor. Often, communication, data dependency (in order to do some of the calculations results are required from other calculations), and other possible overheads (time spent doing something which requires extra calculations and is only used in the parallel code, such as section manipulation in the code from Parallel Methods 7 and 9) can reduce the actual speedup.

### 7.6 Serial Enumeration Trees into Parallel Pruning

The serial version of the code would travel partially down the spanning tree and then backtrack when a route was infeasible due to its weight, and another route was tried. Basically, the pattern created by the serial code resembled binary counting beginning with all 0's and a 1 in the last position. And pruning, which
skips branches in the serial code, skips binary numbers in the parallel code. (which is actually the same as cutting branches.) See figure 5.2.

The authors of the serial code were focused on the spanning trees. The author of the parallel code preferred to visualize a binary representation of the numbers. But, like most mathematics, there is more than one way of correctly viewing a problem and more than one way of obtaining the same correct answer.

7.7 Parallelization Strategies for this Code

This section of this paper describes the author’s thought process and the stages of consideration of ideas for parallel code. Several parallel ideas were considered before the codes’ missing pruning process was included. Any non-pruning algorithm, parallel or not, will be far too time consuming for larger VRPs. Such an algorithm will check $2^{N_{\text{number of cities}}}$ routes. The run time of the non-pruning parallel code increases significantly as the Number of Cities in the VRP slightly increases.

7.7.1 First Ideas

These first two initial ideas would consume too much memory. These are listed mainly to show some methods of attacking a problem in a parallel fashion. The third idea unnecessarily wastes time spawning new worker codes, allowing those codes to exit, and re-spawning new worker codes. Remember these are not very good methods for solving the VRP. A parallel version of code which requires too
much memory space to run is NOT feasible and gains nothing over a serial code which may run an extremely long time but, at least, runs.

1. Binary Tree with \textit{NumberOfCities} Levels (Figure 7.1)

This first programming idea consists of three different codes, one for the first, or master node, one for the intermediate children nodes, and one for the leaf, or terminal children nodes. Each non-leaf node spawns two children and passes important information down to them. The leaf nodes are nodes at level \textit{NumberOfCities} of the binary tree. All of the work is done at the master and leaf nodes. The intermediate nodes are simply for spawning new children nodes and passing information down to them. After performing those two tasks, the intermediate codes exit.

This code uses up too much memory when one code is used with three options of what to do: parent part, middle part, or leaf part. So, the codes were separated into three different codes of different sizes. (37,711Bytes; 4,115Bytes; 26,595Bytes) Yet, these still use a lot of space in memory quickly. One way to reduce some of the memory required is to cut out the middle codes, which mainly spawned and sent important information down to the leaf nodes. (This would save \textit{NumberOfCities} * 4,115\text{Bytes}). This idea is explained next.
Figure 7.1: Binary Tree with Number of Cities Levels
2. $2^{Number\ of\ Cities}$ Tree with 1 Level (Figure 7.2)

The first node spawns $2^{Number\ of\ Cities}$ children. This eliminates the intermediate nodes of the previous method because they do not do anything extremely useful. The space required by this code is $1(mastercode) \times 37,314\ Bytes + 2^{Number\ of\ Cities} \times 26,595\ Bytes$. This still requires more memory than we would like to be using. As the number of cities increases, the amount of memory required significantly increases as well. There has to be a better way!

Figure 7.2: $2^{Number\ of\ Cities}$ Tree with 1 Level

42
3. Loop and Spawn, Loop and Re-Spawn (Figure 7.3)

Spawning refers to beginning another code from within a code. Often code is written in a master/worker manner. In these cases, the master spawns children, or worker processes, performs some calculations, and manages the I/O. The worker processes mainly check the different routes for feasibility and eligibility.

In this set of codes, the master spawns a predetermined number, $N$, of children nodes. They work, send back information, and exit. The master then
spawns N new children. They work, send back information, exit. This process continues until every possible route has been processed. This method does not use as much memory as the first two methods. But, this method does waste time in re-spawning often. One could make better use of his or her time by coding in one of the following methods instead.

7.7.2 Next Ideas

These ideas are definitely better than the first ideas. They do not consume as much memory. However, there is still a problem. These codes do not emulate the pruning process in the serial code. This means that for larger cases, the complexity of the algorithm is still $2^{\text{NumberOfCities}}$ because the weight of every possible route is always checked against at least one weight constraint. These first two methods (4 and 5) are also more time consuming due to the fact that several very small messages are being sent back and forth between the master and worker programs. As long as one does not run into a PVM problem with memory, one should pack larger messages and send fewer large messages rather than a greater number of small messages. (However, if memory DOES become a problem, one may consider sending smaller messages.)

4. Round Robin (Figure 7.4)

This fourth programming method consists of two codes - a master code and a worker code. The master spawns a fixed number of children, or worker,
Master
Sends each worker a route
Waits for message from
Worker 1
Sends each worker a route

Worker 1
Checks a rt
Requests more rts for everyone

Worker 2
Checks a rt

Worker 3
Checks a rt

Worker 4
Checks a rt

Figure 7.4: Round Robin
processes and passes each worker a single route. Every time the first process finishes a route and sends a message indicating that to the master, the master sends one new route to each of the N processes until all possible routes have been checked. The first child spawned is similar to a spokesperson for a group. The first child asks the master for new routes for everyone when that first child is finished with his current route. The problem with this could be that the first child obtains a series of infeasible routes, which allows it to finish quickly, while all other workers have a queue, or long list, of routes to check. Another possible problem could be that some other child is often given infeasible routes, that child, or worker, remains idle until the first child asks for new routes. This means that there is a strong possibility the load balancing will not be very good with this method. No, this is not good enough! Read on.

5. First Come First Served (Figure 7.5)

In this case, again, there are two codes - a master code and a worker code. The master spawns a predetermined number of children, or worker processes and passes each worker a single route. Every time any process finishes a route, that worker process sends a message back to the master. The master then sends that worker the next new route. This continues until all possible routes have been checked.
Master
Sends each worker a rt
Waits for request for a rt
Sends a new rt to the
worker which requests it.

Worker 1
Checks a rt
Asks for a rt

Worker 2
Checks a rt
Asks for a rt

Worker 3
Checks a rt
Asks for a rt

Worker 4
Checks a rt
Asks for a rt

Figure 7.5: First Come First Served
This method should have been quicker and have less processor idle time than the last method. However, the number of communications from the workers to the master is now N times the number of those communications in the previous method, where N is the number of worker processes. One may see the performance of this code is not quite as good as the performance of the previous method due to the increased amount of small messages being sent.

6. Divide the Work into Chunks (Figure 7.6)

After each worker finishes his set of routes, he sends his set of feasible, eligible routes to the master, who will process the routes for the output files.

Figure 7.6: Divide the Work
In this case, there are still two codes - a master code and a worker code. The master spawns a predetermined number of children, or worker processes, and divides up the routes equally by sending each worker process its identification code which describes the subset of routes that this worker will be responsible for checking. Each process then generates a certain set number \( \left( \frac{2^{\text{Number of Cites}}}{N} \right) \) of routes to check.

The main problem with this code is that it still does not prune, or cut, routes. Therefore, a pruning process should be added to this code. That will become parallel method number 8 below.

7. Every Nth Route (Figure 7.7)

This time, there were three codes - a master code, a worker code, and an I/O code. The I/O process was one specialized worker set aside to manage the I/O for the master.

The master sends the first child route number 1, the second child route number 2, and so on until the Nth child receives route number N. The master also sends each child the total number of worker processes, N. Every process checks its first route and then checks the route which is N more than the last route. This process continues until all possible routes have been checked.

The I/O node receives the printable routes from the workers and orders them before the routes are printed into files.
Master
Sends every worker his first route, and waits for the I/O node to send the feasible rts to the output file.

Worker 1
First, check the given route. Then check the route which is 4 more than the last one send route to I/O

Worker 2
First, check the given route. Then check the route which is 4 more than the last one send route to I/O

Worker 3
First, check the given route. Then check the route which is 4 more than the last one send route to I/O

Worker 4
First, check the given route. Then check the route which is 4 more than the last one send route to I/O

I/O node
Receives feasible routes from the workers, orders the routes, and prints to a file.

Remember that choosing the number of processors in EveryNth to be a power of two, as is shown here will lead to bad load balancing as the next figure will show. A prime number would have been a better choice for the number of processors to be used with this set of code.

Figure 7.7: Every Nth Route
For better load balancing, be certain to choose \( N \) such that \( N \) is NOT 2 raised to some power. (Figure 7.8) When one does choose \( N \) to be some power of 2, the last digit or two in the route is set to be some value which never changes. When the cities are ordered from minimum weight to maximum weight, that means that the maximum weight is always set for a process. One process will get all of the heaviest routes, many of which may be infeasible. Thus the load balancing with this method is not very good when \( N \) is a power of 2 and the cities are ordered from minimum to maximum weight.

<table>
<thead>
<tr>
<th>Worker 1</th>
<th>Worker 2</th>
<th>Worker 3</th>
<th>Worker 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>0010</td>
<td>0011</td>
<td>0100</td>
</tr>
<tr>
<td>0101</td>
<td>0110</td>
<td>0111</td>
<td>1000</td>
</tr>
<tr>
<td>1001</td>
<td>1010</td>
<td>1011</td>
<td>1000</td>
</tr>
<tr>
<td>1101</td>
<td>1110</td>
<td>1111</td>
<td>1100</td>
</tr>
</tbody>
</table>

Choosing the number of worker processors to be a power of two for the EveryNth codes is a bad idea if the cities are ordered from minimum to maximum weight demand. The last position or two in the array will be set (and those positions are the heaviest). This is a simple example with four cities which shows how the work would be divided among the workers. Notice that worker 3 gets the heaviest routes. The heaviest routes are more likely to be infeasible due to the maximum weight constraint of a route. Thus, worker 3 will probably finish more quickly than the other workers. Most of the work, in this case falls to worker 4 and worker 1. Therefore, the load balancing is not very good.

Figure 7.8: Bad Load Balancing for Every Nth Route
Regardless, this method still does not prune routes. And, one could NOT add pruning to this particular method. This method seems to “jump” into the tree at different points. And, when a process finds a route which exceeds the maximum weight limit, that does NOT mean the binary representation of that route plus a binary representation of N would produce a binary route which also exceeds the maximum weight limit. Nice try, but the methods definitely have to include pruning to be good.

7.7.3 Best Ideas

These next methods actually involve pruning, so they are much better than the other methods presented.

8. Divide and Prune (Figure 7.9)

This idea is the same as the divide method (6) above, but pruning is added. First, give each worker process an id number or section of the binary tree. For best results, a binary number of processors should be used (4, 8, 16, 32, etc.) This will evenly divide the work into subtrees. When the cities are ordered from minimum to maximum weight, as they were in the serial code, this divide and prune method balances the load fairly well. Each worker process has only the minimum positions set and must go through several combinations of the heavier weight demand cities. Therefore, one process doesn’t get most of the heavy infeasible routes.
This is an example of Divide with 4 workers. Each worker prunes his own routes based on the maximum truck capacity. When the cities are ordered from minimum to maximum weight demand, the cities set by the workers will be the minimum weight cities. Each worker will go through all maximum weight combinations. Therefore, Divide with Prune and this ordering of cities balances the load among worker processes.

Figure 7.9: Good Load Balancing for Divide and Prune
9. Divide and Prune with Sections (Figure 7.10)

This code is not as good as the previous code (method 8) because the positions which are fixed are heavier than the positions which are fixed in method 8 when the cities are ordered from minimum to maximum weight demand. The master process gives each worker an id number. Each worker then establishes a power of 2 number of sections. The routes are set up in this manner: section number, worker id number, rest of binary value for the route. For the minimum to maximum weight ordering, the overhead associated with manipulating sections on each worker process is probably more time-consuming than time-saving due to the poor load balancing.

7.7.4 Reordering Might Prune More Routes

One should be able to prune slightly more routes by ordering the cities by weight demands from maximum to minimum rather than from minimum to maximum. (Figure 7.11) Including minimum pruning could also prune additional non-feasible routes.

So, one ought to see a significant speed-up by simply reordering the routes. However, a problem has arisen when these codes are run on cases which produce 6,000 or more routes, there is a slight deviance (less than 0.1%) in the number of routes obtained through this method as opposed to the same methods ordered in the opposite manner listed above. The two previous methods (8 and 9) obtained
Divide and Prune without sections:

worker1: 00......... worker2: 01...........
worker3: 10......... worker4: 11...........

Divide and Prune with Sections:

worker1: XX00.... worker2: XX01.......
worker3: XX10.... worker4: XX01.......

where the XX takes on the values 00, 01, 10, 11

When the cities are ordered from maximum to minimum weight demand, these sections help balance the load among the workers.

Figure 7.10: Divide and Prune and Sections
Reordering should prune more routes:

Assume there are 5 cities with these weight demands: 5, 15, 12, 20, 18. MaxW = 23, MinW = 0

The left column shows the 11 routes checked when the cities are ordered from max to min. The column on the right shows the 21 routes checked when the cities are ordered from min to max. Ordering the cities from maximum to minimum weight demand results in checking 10 fewer infeasible routes than when the cities are ordered from minimum to maximum weight demands!

Figure 7.11: Good Load Balancing for Divide and Prune
the same amount of routes as the serial code had generated for the cases. (Table 7.1)

Table 7.1: Deviance in the Number of Routes

<table>
<thead>
<tr>
<th>Case</th>
<th>Rts (min to max)</th>
<th>Rts (max to min)</th>
<th>Change in # of Routes</th>
</tr>
</thead>
<tbody>
<tr>
<td>12-3</td>
<td>213</td>
<td>213</td>
<td>0</td>
</tr>
<tr>
<td>12-2</td>
<td>1252</td>
<td>1252</td>
<td>0</td>
</tr>
<tr>
<td>15-5</td>
<td>88</td>
<td>88</td>
<td>0</td>
</tr>
<tr>
<td>15-3</td>
<td>81</td>
<td>81</td>
<td>0</td>
</tr>
<tr>
<td>19-2</td>
<td>8547</td>
<td>8560</td>
<td>+13</td>
</tr>
<tr>
<td>20-6</td>
<td>25</td>
<td>25</td>
<td>0</td>
</tr>
<tr>
<td>20-4</td>
<td>156</td>
<td>156</td>
<td>0</td>
</tr>
<tr>
<td>25-8</td>
<td>91</td>
<td>91</td>
<td>0</td>
</tr>
<tr>
<td>29-3</td>
<td>10740</td>
<td>10739</td>
<td>-1</td>
</tr>
<tr>
<td>32-4</td>
<td>65657</td>
<td>65710</td>
<td>+53</td>
</tr>
<tr>
<td>32-3</td>
<td>6337</td>
<td>6332</td>
<td>-5</td>
</tr>
</tbody>
</table>

10. Partition and Prune (Cities ordered max to min)

This programming idea has the cities are ordered from maximum weight to minimum weight. Divide the work among the worker processors by sending each one his own worker id and therefore his partition of the work as in method 8. The main problem with this method when the cities are order in this fashion is that the extremely heavy routes will all fall to one processor. That processor will not have as many feasible routes as the others and will finish much quicker than the other routes. Also, the processor with the next highest id number may have more infeasible routes than the rest of the processors which are still working. Most of the work load may fall to a
few processors. How could the load be better balanced among the worker processes?

11. Divide and Prune with Sections (Cities ordered max to min)

In this final method, again, the cities are ordered from maximum weight to minimum weight in order to prune more routes and gain time. Due to the load balancing problem in the previously mentioned method, sections were added to the worker nodes as in method 9. The workers were each sent an id number. They were in charge of managing their own sections themselves. The routes are set up in this manner: section number, id number, rest of the binary value for the route. This method should create a better balance of the load and the overhead per worker process may be justified in this case.

However, since the reordering of cities produced a different number of routes in cases which produced over 6,000 routes, these last two methods were not used. So, parallel method 8, Divide and Prune with the cities ordered minimum to maximum weight demand, was considered to be the best code. If the reordering had resulted in the same number of routes for all cases, parallel method 11, Divide and Prune with Sections and with cities ordered from maximum to minimum weight demand, may have been the better method.
Chapter 8

Testing of the Parallel Codes

and Results

8.1 Why Parallel Methods 1 through 3 should NOT be used

The first three ideas were a starting place, but they were not acceptable ideas because they proved very quickly that they would use up far too much memory or unnecessarily waste time spawning, exiting, and re-spawning code. As that the amount of memory available on the machines one is working on is still an issue, these first ideas were quickly tossed aside, but used as stepping stones towards building the ideas that followed. The main goal of parallelizing the code is to have a code which runs and can gain a speedup in time. Thus, new ideas were
considered.

8.2 Why Parallel Methods 4 and 5 should NOT be used

Both of these two methods lost a lot of time to the massive amount of small messages being sent back and forth between the processes. (See Table 8.1.) Also, these did not include any pruning which meant that every route was at least checked whether it met one of the weight requirements. Therefore, the complexity, or order, of these algorithms is $2^{\text{Number Of Cities}}$.

<table>
<thead>
<tr>
<th>Case</th>
<th>Serial Time (sec)</th>
<th>Round Robin (sec)</th>
<th>First Come (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>12-3</td>
<td>0.43</td>
<td>4.20</td>
<td>6.00</td>
</tr>
<tr>
<td>12-2</td>
<td>1.88</td>
<td>4.53</td>
<td>5.74</td>
</tr>
<tr>
<td>15-5</td>
<td>0.25</td>
<td>77.12</td>
<td>48.67</td>
</tr>
<tr>
<td>15-3</td>
<td>0.68</td>
<td>78.23</td>
<td>47.67</td>
</tr>
<tr>
<td>19-2</td>
<td>44.93</td>
<td>573.17</td>
<td>695.52</td>
</tr>
<tr>
<td>20-6</td>
<td>0.72</td>
<td>836.13</td>
<td>1251.13</td>
</tr>
<tr>
<td>20-4</td>
<td>4.87</td>
<td>831.65</td>
<td>1326.68</td>
</tr>
<tr>
<td>25-8</td>
<td>2.75</td>
<td>several hrs</td>
<td>several hrs</td>
</tr>
</tbody>
</table>

8.3 Why Parallel Methods 6 and 7 should NOT be used

Although these codes are much better than the two previous codes, they also do not use pruning. The timings from these two methods are listed in Table 8.2. The larger cases still will not run in a reasonable amount of time.
Table 8.2: Serial versus Non-Pruning Parallel Codes

<table>
<thead>
<tr>
<th>Case</th>
<th>Serial Time (sec)</th>
<th>Non-Pruning Divide (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>12-3</td>
<td>0.43</td>
<td>2.33</td>
</tr>
<tr>
<td>12-2</td>
<td>1.88</td>
<td>11.23</td>
</tr>
<tr>
<td>15-5</td>
<td>0.25</td>
<td>1.25</td>
</tr>
<tr>
<td>15-3</td>
<td>0.68</td>
<td>1.37</td>
</tr>
<tr>
<td>20-6</td>
<td>0.72</td>
<td>0.53</td>
</tr>
<tr>
<td>20-4</td>
<td>4.87</td>
<td>3.0</td>
</tr>
<tr>
<td>25-8</td>
<td>2.75</td>
<td>1.95</td>
</tr>
</tbody>
</table>

Pruning is essential to gaining speedup in the parallel codes. Therefore pruning can be added to the parallel method 6, which divides the work among the routes. Pruning cannot be added to the parallel method 7 due to the manner in which each worker obtains its next route.

8.4 Serial versus Parallel versions 10 and 11

These ordered the cities from maximum weight to minimum weight in an attempt to prune more infeasible routes. However, these were not tested due to the difference in the number of routes for the larger cases when the cities were ordered from maximum to minimum weight demand.
8.5 Cities Must Be Ordered Min to Max

When cities were ordered from maximum to minimum weight demand, they produced a different number of routes than when the cities were ordered from minimum to maximum weight. That was the only change to the code except, of course, the pruning methods were slightly different depending on the ordering of the cities. The parallel code should emulate the serial code. In the serial code, the cities were ordered from minimum to maximum weight demand. The serial code produced the same routes as the parallel code when the cities were ordered in the same manner (from minimum to maximum weight demand). Thus, the minimum to maximum weight demand ordering of the cities was the one that was used. Thus, parallel method 8, Divide and Prune, was considered the best parallel method presented.

8.6 Serial versus Parallel version 9

Remember that this method used sections when they were not really required to balance the load. Therefore, this set of codes was also not tested.

8.7 The Best Method - Serial versus Parallel version 8

Once again, this was the very best parallel method presented. The timings of this algorithm are as follows (Table 8.3).
Table 8.3: Serial versus Pruning Parallel Codes

<table>
<thead>
<tr>
<th>Case</th>
<th>Serial Time on SP2 (sec)</th>
<th>Divide Master over 4 PROCS on SP2 (sec)</th>
<th>Divide Child (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>12-3</td>
<td>0.12</td>
<td>0.37</td>
<td>0.18</td>
</tr>
<tr>
<td>12-2</td>
<td>0.78</td>
<td>2.80</td>
<td>0.29</td>
</tr>
<tr>
<td>15-5</td>
<td>0.06</td>
<td>0.21</td>
<td>0.18</td>
</tr>
<tr>
<td>15-3</td>
<td>0.22</td>
<td>0.19</td>
<td>0.52</td>
</tr>
<tr>
<td>19-2</td>
<td>15.41</td>
<td>24.31</td>
<td>23.80</td>
</tr>
<tr>
<td>20-6</td>
<td>0.28</td>
<td>0.06</td>
<td>1.68</td>
</tr>
<tr>
<td>20-4</td>
<td>2.03</td>
<td>0.67</td>
<td>6.90</td>
</tr>
<tr>
<td>25-8</td>
<td>1.28</td>
<td>0.32</td>
<td>8.68</td>
</tr>
<tr>
<td>29-3</td>
<td>7558.14</td>
<td>3.44</td>
<td>3020.16</td>
</tr>
<tr>
<td>32-4</td>
<td>23229.77</td>
<td>13.60</td>
<td>6881.93</td>
</tr>
</tbody>
</table>

8.8 Comparing PVM code over a LAN versus PVM code on the SP2

The SP2 is a collection of RS6000 processors connected through a high performance switch. The RS6000 tends to be two to four times quicker over the than the SPARC5s of which the LAN was comprised at least for the serial codes. (See Table 8.4.) The SP2 processors also had more memory available on them than the machines in the LAN. The testing of PVM on the SP2 should also show the SP2 to be about twice as fast as the collection of SPARC5s. However, there were some memory problems with the LAN for even the relatively small case with 19 cities and 2 trucks which produces 8,547 routes. Thus, a comparison of PVM over the LAN of SPARC5s versus PVM over the SP2 has not been included.
Table 8.4: Serial Time on a SPARC5 versus the SP2

<table>
<thead>
<tr>
<th>Case</th>
<th>Serial Time on SPARC5 (sec)</th>
<th>Serial Time on the SP2 (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>12-3</td>
<td>0.37</td>
<td>0.12</td>
</tr>
<tr>
<td>12-2</td>
<td>1.57</td>
<td>0.78</td>
</tr>
<tr>
<td>15-5</td>
<td>0.20</td>
<td>0.06</td>
</tr>
<tr>
<td>15-3</td>
<td>0.75</td>
<td>0.22</td>
</tr>
<tr>
<td>19-2</td>
<td>44.55</td>
<td>15.41</td>
</tr>
<tr>
<td>20-6</td>
<td>0.78</td>
<td>0.28</td>
</tr>
<tr>
<td>20-4</td>
<td>5.75</td>
<td>2.03</td>
</tr>
<tr>
<td>25-8</td>
<td>2.75</td>
<td>1.28</td>
</tr>
<tr>
<td>29-3</td>
<td>20199.8</td>
<td>7558.14</td>
</tr>
<tr>
<td>32-4</td>
<td>51361.6</td>
<td>23229.77</td>
</tr>
</tbody>
</table>

8.9 Different numbers of Processors on the IBM SP2

Since the smaller cases tend to run rather quickly, only the larger cases would be used to determine the effect of adding more processors to the applications. The performance is expected to increase with the number of processors.

8.10 Conclusions

Many smaller cases tend to run rather quickly in serial. Those cases tended to run a little slower in parallel.

However, for intermediate cases, using an SP with 4 processors showed an impressive improvement of time over the serial code on one SP2 processor. See Table 8.5.
Table 8.5: Speedup of PVM over 4 Processors on the SP2

<table>
<thead>
<tr>
<th>Case</th>
<th>Serial Time on SP2 (sec)</th>
<th>Divide Maximum Child (4 PROCS on SP2) (sec)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>29-3</td>
<td>7558.14</td>
<td>3020.16</td>
<td>2.5</td>
</tr>
<tr>
<td>32-4</td>
<td>23229.77</td>
<td>6881.93</td>
<td>3.38</td>
</tr>
</tbody>
</table>

Even larger cases were expected to obtain similar results. Adding more processors should also result in an even better speedup.
Chapter 9

Areas for Future Research

Currently, there are several suggestions for future areas of research associated with this project. Some may seem more suited to Management Science, whereas some may be more suited to Computer Science.

First of all, one could try using parallel genetic algorithms as a good algorithm for the first step of the Set Partitioning method. Also, a parallel LP solver could be utilized as part of that step. If a parallel LP solver already exists, it should be used at the point; otherwise, a parallel LP solver could be constructed for this step.

As for the second step of this method, one could experiment with compiler flags, different parallel machines, the use of C versus FORTRAN, the use of PVM versus PVMe on the SP2.

For the third step, a parallel IP solver could be used if one has not already
been used. Again, if a parallel IP solver already exists, it should be used at the point; otherwise, a parallel IP solver could be constructed for this step.

Lastly, one could package the three steps together on some machine and market the whole package as something both useful and incredible!

Also, someone in Management Science could run several tests to determine the parallel codes' performance as a function of number of trucks, cities, TimeLimit, and the Gap variable. How the code scales with the number of processors could also be investigated.

A Business Management person could also test the limits of the parallel codes on the SP2. How large of a case can be run in reasonable time, (whatever that is decided to be)?

“There is something fascinating about science. One gets such wholesale returns of conjecture out of such a trifling investment of fact.” Mark Twain

Whatever the extended research may be, this researcher hopes the next researcher finds his or her experience to be as learning a one as this project has been for this researcher.
Bibliography
Appendix A - Serial Code
PROGRAM ENUMERATE

INTEGER MAXPTS
PARAMETER(MAXPTS = 101)
COMMON/MATRIX/DIST, NUMVTX, LOWERB
INTEGER DIST(MAXPTS, MAXPTS), NUMVTX, LOWERB
INTEGER DISTANCE(MAXPTS, MAXPTS), NODEOF(MAXPTS)
INTEGER X(101), Y(101), NV, STOPTIME, TIMELIMIT
INTEGER N, MINW, MAXW, T, I, J, K, W(101), WB(101), L, S(101)
INTEGER COUNT, CARDINALITY, ORIG_LABEL(101), 0
REAL DUAL(101), DUALT, Z, GAP
integer indual(101), indualt, r100000, i100
real tocost
COUNT=0
CARDINALITY=0

OPEN (15, FILE = 'ENUM.IN', STATUS = 'UNKNOWN')
OPEN (16, FILE = 'ENUM.OUT', STATUS = 'UNKNOWN')
OPEN (17, File = 'cols.mps', status = 'unknown')
READ(15,*) N, MAXW, NV, STOPTIME, TIMELIMIT, GAP
T=0
print *, 'starting to run enumerate > enumerate.out'
DO 1 I=1,N
ORIG_LABEL(I)=I
READ(15,*) X(I), Y(I), W(I)
PRINT *, X(I), Y(I), W(I)
1 T=T+W(I)
do 201 i=1,N
read(15,*)dual(i)
dual(i)=dual(i)*1000.
201 indual(i)=int(dual(i))
N=N-1
MINW=T-(NV-1)*MAXW
MINW=MAX(0, MINW)
c read(15,*) minw
PRINT *, MINW, MAXW
DUALT=DUAL(N+1)
indualt=indual(n+1)
write(17,71)
format('NAME',5x,'SETPARLP')
write(17,72)
format('ROWS')
write(17,73)
format(1x,'N',2x,'COST')
do 78 i=1,n
  i100=i+100
  write(17,74)i100
format(1x,'E',2x,'ND',i3)
continue
write(17,755)
format(1x,'E',2x,'VEHI')
write(17,77)
format('COLUMNS')

2  L=0
3  L=L+1
   IF(L.EQ.N) GO TO 4
   IF(W(L).LE.W(L+1)) GO TO 3
   0=ORIG_LABEL(L)
   I=W(L)
   J=X(L)
   K=Y(L)
   Z=DUAL(L)
   ijk=indual(L)
   ORIG_LABEL(L)=ORIG_LABEL(L+1)
   W(L)=W(L+1)
   X(L)=X(L+1)
   Y(L)=Y(L+1)
   DUAL(L)=DUAL(L+1)
   indual(L)=indual(L+1)
   ORIG_LABEL(L+1)=0
   W(L+1)=I
   X(L+1)=J
   Y(L+1)=K
   DUAL(L+1)=Z
0. indual(L+1)=ijk
GO TO 2

4   PRINT *,(W(I),I=1,N)
    print *,(dual(i),i=1,n)
    do 901 i=1,n+1
        do 901 j=1,n+1
        distance(i,j)=int(1000*((x(i)-x(j))**2+(y(i)-y(j))**2)**.5)
        if(distance(i,j).eq.0) then
            print *,'WOW WOW dist is zero ',i,j
        end if
    901    dist(i,j)=distance(i,j)
        do 902 i=1,n+1
            distance(i,i)=99999999
    902    dist(i,i)=distance(i,i)
    numvtx=n+1
    call HEURISTIC_4(numvtx,dist,lowerb)
    print *, 'TSP = ',lowerb
    call rreduce
    print *, 'MAM = ',lowerb
    call onetree
    print *, '1-TREE = ',lowerb

W(N+1)=MAXW*N
DO 5 I=1,N-1
    T=T-W(I)
5    WB(I)=T
    WB(N)=-MAXW*N
    WB(N+1)=-MAXW*N

L=0
T=0
10   L=L+1
    S(L)=0
    IF(WB(L)+T.GE.MINW) GO TO 10
20   S(L)=1
    T=T+WB(L)
    indualt=indualt+indual(L)
    CARDINALITY=CARDINALITY+1
NODEOF(CARDINALITY)=L
DIST(1,CARDINALITY+1)=DISTANCE(N+1,L)
DIST(CARDINALITY+1,1)=DISTANCE(L,N+1)
DO 25 I=1,CARDINALITY-1
   DIST(I+1,CARDINALITY+1)=DISTANCE(NODEOF(I),L)
   DIST(CARDINALITY+1,I+1)=DISTANCE(L,NODEOF(I))
25
DIST(CARDINALITY+1,CARDINALITY+1)=999999999
NUMVTX=CARDINALITY+1

IF(T.LE.MAXW) GO TO 40
30 IF(S(L).EQ.0) GO TO 20
   T=T-W(L)
   indualt=indualt-indual(L)
   CARDINALITY=CARDINALITY-1
   L=L-1
   IF(L.EQ.O) THEN
      WRITE(16,1601)COUNT
      WRITE(17,101)
      FORMAT('LAST ROUTE =', I6)
      WRITE(17,103)
      DO 103 I=1,N
         I100=I+100
         WRITE(17,102)I100
         FORMAT('RHS', 2X, 'NDM3.2X, 1.0')
      103 CONTINUE
      WRITE(17,115)FLOAT(NV)
      FORMAT('RHS', 2X, 'VEHI', 4X, F3.1)
      WRITE(17,105)
      FORMAT('BOUNDS')
      DO 106 I=1,COUNT
         WRITE(17,107)100000+I
         FORMAT('BV', 6X, 'ENDS', 2X, 'R', I6)
      106 CONTINUE
      WRITE(17,104)
      FORMAT('ENDATA')
      STOP
   END IF
   GO TO 30
40 IF(T.LT.MINW) GO TO 10
   IF(NUMVTX.GE.2) THEN
if(numvtx.eq.2) then
lowerb=dist(1,2)+dist(2,1)
IF(LOWERB+1000*CARDINALITY*STOPTIME.GT.1000*TIMELIMIT) GO TO 35
else
call onetree
IF(LOWERB+1000*CARDINALITY*STOPTIME.GT.1000*TIMELIMIT) GO TO 35
call HEURISTIC_4(numvtx,dist,lowerb)
IF(LOWERB+1000*CARDINALITY*STOPTIME.GT.1000*TIMELIMIT) GO TO 35
end if
if(lowerb-indualt.lt.int(gap)) then
nz=lowerb-indualt
COUNT=COUNT+1
WRITE(16,*)COUNT,nz,LOWERB,(s(i),i=1,l)
WRITE(16,1616)COUNT,nz,LOWERB,(s(i),i=1,l)
1616 FORMAT(i6,2i8, 50(i2))
tocost=lowerb/1000.
ri00000=count+100000
write(17,119) ri00000,tocost
119 format(1x,'R',i6,2x,'COST',2x,f8.2)
do 333 ijk=1,1
if(s(ijk).eq.1) then
i100=orig_label(ijk)+100
write(17,44)r100000,i100
write(17,44)i100
44 format(2x,'ND',i3,2x,'1.0')
end if
333 continue
write(17,157)r100000
157 format(1x,'R',i6,2x,'VEHI',4x,'1.0')
end if
END IF
if(t+w(l+l).gt.maxw) go to 35
GO TO 10
END
THIS IS A PROTOTYPE CODE (OCT. 1989) WRITTEN AND DESIGNED BY
CHARLES E. NOON (UNIV OF TENNESSEE) AND THOMAS CHAN (S.M.U.) FOR
PRODUCING LOWER BOUNDS ON THE LP RELAXATION OF THE PERFECT
2-MATCHING PROBLEM. THE APPROACH IS BASED ON "A MULTIPLIER
ADJUSTMENT APPROACH FOR THE SET PARTITIONING PROBLEM" BY
THOMAS CHAN AND CANDACE YANO (TECH. PAPER, DEPT. OF CSE, S.M.U.,
DECEMBER, 1988). NO PART OF THIS CODE MAY BE COPIED OR
DISTRIBUTED WITHOUT WRITTEN CONSENT OF THE AUTHORS.

THE PROBLEM TO BE BOUNDED IS GIVEN AS,
MINIMIZE CX
AX = 2
0 <= X <= 1
WHERE "A" IS THE NODE EDGE INCIDENCE MATRIX FOR A GRAPH. THE
EDGE COSTS "C" ARE TO BE INPUT AS A COMPLETE N BY N DISTANCE
MATRIX.

NOTE: THE DIMENSIONS ARE SET FOR 100 NODES.

**********
THE FOLLOWING FOUR LINES SHOULD BE COPIED AND PLACED IN
THE PART OF YOUR CODE WHERE AN INTEGER DISTANCE MATRIX IS STORED.
THE COMPLETE SYMMETRIC DISTANCE MATRIX (ALL N SQUARED VALUES)
SHOULD BE WRITTEN INTO DIST(I,J) AND THE NUMBER OF VERTICES
SHOULD BE ASSIGNED TO NUMVTX.

INTEGER MAXPTS
PARAMETER(MAXPTS = 101)
COMMON/MATRIX/DIST,NUMVTX,LOWERB
INTEGER DIST(MAXPTS,MAXPTS),NUMVTX,LOWERB

**********
AFTER THE DISTANCE MATRIX AND NUMVTX IS LOADED INTO THE
COMMON BLOCK /MATRIX/, THE SUBROUTINE CAN BE CALLED BY
A SIMPLE "CALL RREDUCE" COMMAND. THE LOWER BOUND WILL BE GIVEN

SUBROUTINE RREDUCE
C***************************************************************
C THE FOLLOWING FOUR LINES SHOULD BE COPIED AND PLACED IN
C THE PART OF YOUR CODE WHERE AN INTEGER DISTANCE MATRIX IS STORED.
C THE COMPLETE SYMMETRIC DISTANCE MATRIX (ALL N SQUARED VALUES)
C SHOULD BE WRITTEN INTO DIST(I,J) AND THE NUMBER OF VERTICES
C SHOULD BE ASSIGNED TO NUMVTX.
C
INTEGER MAXPTS
PARAMETER(MAXPTS = 101)
COMMON/MATRIX/DIST,NUMVTX,LOWERB
INTEGER DIST(MAXPTS,MAXPTS),NUMVTX,LOWERB

**********
AFTER THE DISTANCE MATRIX AND NUMVTX IS LOADED INTO THE
COMMON BLOCK /MATRIX/, THE SUBROUTINE CAN BE CALLED BY
A SIMPLE "CALL RREDUCE" COMMAND. THE LOWER BOUND WILL BE GIVEN

78
C AS THE "LOWERB" VALUE AND THE DUAL FEASIBLE REDUCED COSTS WILL C
BE WRITTEN BACK INTO THE DIST(I,J) MATRIX. THEREFORE, YOU MAY C
READ THE REDUCED COSTS OUT OF THE /MATRIX/ COMMON BLOCK AND WORK C
ON THEM. THE INTERNAL ARRAY BESTU( ) IS THE FINAL VECTOR OF DUAL C
VARIABLE VALUES CORRESPONDING TO THE AX=2 CONSTRAINTS.
C
PARAMETER(NUMROW = 900, NUMCOL = 450000)
INTEGER C,R,C1,CADD1,P1,PADD1,SUM,CUTOFF,NCOUNT,NROW
INTEGER LB,NSOL,NCOL,TOTW,ITER,CUT,UB,TOTAL,TEMP,TEMPT
INTEGER ROWST(NUMROW),ROWEND(NUMROW),AROW(NUMCOL+NUMCOL)
INTEGER ACOL(NUMCOL+NUMCOL),U(NUMROW),BESTU(NUMROW)
INTEGER COST(NUMCOL),ACOST(NUMCOL),COUNT(NUMROW)
INTEGER NCOVER(NUMROW),SOL(NUMCOL),TABUR1(NUMROW)
LOGICAL FOUND,FOUNDA
MXITER = 10
NROW = NUMVTX
NCOL=NROW*(NROW-1)/2
DO 10 R = 1,NROW
COUNT(R) = 0
10 CONTINUE

C THE FOLLOWING LOOP IS WHERE THE UPPER TRIANGLE OF
C THE DISTANCE MATRIX IS READ.
C
C=0
INDEX=1
NLESS1=NROW-1
DO 30 I1=1,NLESS1
I1AND1=I1+1
DO 20 I2=I1AND1,NROW
C=C+1
COST(C)=DIST(I1,I2)
ACOL(INDEX) = I1
ACOL(INDEX+1) = I2
COUNT(I1) = COUNT(I1) + 1
COUNT(I2) = COUNT(I2) + 1
ACOST(C) = COST(C)
INDEX = INDEX + 2
20 CONTINUE
30 CONTINUE
C
C FURTHER INITIALIZATION.
C
TOTAL = 0
DO 40 R = 1,NROW
ROWST(R) = TOTAL + 1
TOTAL = TOTAL + COUNT(R)
ROWEND(R) = TOTAL
COUNT(R) = ROWST(R) - 1
NCOVER(R) = 0
U(R) = 0
TABUR1(R) = 0
40 CONTINUE
INDEX = 0
DO 50 C = 1,NCOL
P1 = 2 * C
I1 = ACOL(P1-1)
I2 = ACOL(P1)
COUNT(I1) = COUNT(I1) + 1
AROW(COUNT(I1)) = C
COUNT(I2) = COUNT(I2) + 1
AROW(COUNT(I2)) = C
50 CONTINUE
LB = 0
NSOL = 0
TOTAL = 0
ITER = 0
C
C END OF INITIALIZATION.
C BEGIN ITERATIONS.
C
60 ITER = ITER + 1
IF(ITER .EQ. 1) THEN
TOTW=0
FOUND = .TRUE.
GO TO 220
END IF
CUTOFF = ITER - 1
M = NSOL
FOUND = .FALSE.
DO 200 J = M,1,-1
C = SOL(J)
IF(ACOST(C) .GT. 0) GO TO 200
P1 = 2 * C
I1 = ACOL(P1-1)
I2 = ACOL(P1)
IF(NCOVER(I2) .LT. NCOVER(I1)) THEN
  I = I1
  I1 = I2
  I2 = I
END IF
IF(NCOVER(I1) .NE. 1) GO TO 200
FOUND = .TRUE.
MINIM = 10**9
R = I1
DO 70 L = ROWST(R),ROWEND(R)
  TEMP = ACOST(AROW(L))
  IF((TEMP .LT. MINIM) .AND.(TEMP .GT. 0)) THEN
    MINIM = TEMP
    GADD1 = AROW(L)
  END IF
70 CONTINUE
U(R) = U(R) + MINIM
LB = LB + MINIM
DO 90 L = ROWST(R),ROWEND(R)
  GADD1 = AROW(L)
  ACOST(CADD1) = ACOST(CADD1) - MINIM
  IF(ACOST(CADD1) .EQ. 0) THEN
    PADD1 = 2 * CADD1
    IADD1 = ACOL(PADD1-1)
    IADD2 = ACOL(PADD1)
    NCOVER(IADD1) = NCOVER(IADD1) + 1
    NCOVER(IADD2) = NCOVER(IADD2) + 1
    DO 80 IADD = 1,NSOL
      IF(SOL(IADD) .EQ. GADD1) GO TO 90
80 CONTINUE
  NSOL = NSOL + 1
  SOL(NSOL) = GADD1
END IF
90 CONTINUE
TABUR1(I1) = ITER
IF((NCOVER(I2) .GT. 2).AND.(TABUR1(I2) .LT. CUTOFF)) THEN
  MIN = ACOST(C)
R=I2
SUM=MIN
U(R) = U(R) + SUM
LB = LB + SUM
DO 100 L = ROWST(R),ROWEND(R)
  CADD1 = AROW(L)
  IF(ACOST(CADD1) .LE. 0) THEN
    FOUNDA = .TRUE.
  ELSE
    FOUNDA = .FALSE.
  END IF
  ACOST(CADD1) = ACOST(CADD1) - SUM
  IF((ACOST(CADD1) .GT. 0) .AND.(FOUND)) THEN
    PADD1 = 2 * CADD1
    IADD1 = ACOL(PADD1-1)
    IADD2 = ACOL(PADD1)
    NCOVER(IADD1) = NCOVER(IADD1) - 1
    NCOVER(IADD2) = NCOVER(IADD2) - 1
  END IF
100 CONTINUE
END IF
200 CONTINUE
IF(.NOT. FOUND) GO TO 500
NCOUNT = 0
TOTW = 0
DO 210 I = 1,NSOL
  C = SOL(I)
  IF(ACOST(C) .LE. 0) THEN
    NCOUNT = NCOUNT + 1
    SOL(NCOUNT) = C
    TOTW = TOTW + ACOST(C)
  END IF
210 CONTINUE
NSOL = NCOUNT
220 DO 290 R = 1,NROW
  IF(NCOVER(R) .NE. 0) GO TO 290
  MINIM = 10**9
  DO 230 L = ROWST(R),ROWEND(R)
    TEMP = ACOST(AROW(L))
    IF(TEMP .LT. MINIM) THEN
      MINIM = TEMP
    ELSE
      NCOVER(R) = 0
    END IF
230 CONTINUE
500 CONTINUE
82
CADD1 = AROW(L)
END IF

230 CONTINUE
U(R) = U(R) + MINIM
LB = LB + MINIM
DO 240 L = ROWST(R),ROWEND(R)
CADD1 = AROW(L)
ACOST(CADD1) = ACOST(CADD1) - MINIM
IF(ACOST(CADD1) .EQ. 0) THEN
PADD1 = 2 * CADD1
IADD1 = ACOL(PADD1-1)
IADD2 = ACOL(PADD1)
NCOVER(IADD1) = NCOVER(IADD1) + 1
NCOVER(IADD2) = NCOVER(IADD2) + 1
DO 250 IADD = 1,NSOL
IF(SOL(IADD) .EQ. CADD1) GO TO 240
250 CONTINUE
NSOL = NSOL + 1
SOL(NSOL) = CADD1
END IF

240 CONTINUE
290 CONTINUE
IF(.NOT. FOUND) GO TO 500
TEMPT = 2 * LB + TOTW
IF(TEMPT .GT. TOTAL) THEN
TOTAL = TEMPT
DO 300 R=1,NROW
BESTU(R) = U(R)
300 CONTINUE
END IF
IF(ITER .LT. MXITER) GO TO 60
C
C ITERATIONS COMPLETED.
C BEGIN CALCULATION OF REDUCED COSTS AND TERMINATION.
C
500 NTOTU=0
DO 510 I=1,NROW
NTOTU=NTOTU+BESTU(I)
510 CONTINUE
NTOTS=0
DO 530 C=1,NCOL

83
P1 = 2 * C
I1 = ACOL(P1-1)
I2 = ACOL(P1)
NREDUC=COST(C)-BESTU(I1)-BESTU(I2)
IF(NREDUC.LT.0) THEN
  NTOTSI=NTOTSI+NREDUC
  NREDUC=0
END IF
ccccc 3-7-95 DIST(I1,I2) = NREDUC
ccccc 3-7-95 DIST(I2,I1) = NREDUC
530 CONTINUE
LOWERB=2*NTOTU+NTOTS1
PRINT *, 'FINAL LOWERBOUND = ', LOWERB, ' = 2 * ', ntotu, ' + ', ntotsi
RETURN
END

SUBROUTINE ONETREE

INTEGER MAXPTS
PARAMETER(MAXPTS = 101)
COMMON/MATRIX/DIST,NUMVTX,LOWERB
INTEGER DIST(MAXPTS,MAXPTS),NUMVTX,LOWERB
INTEGER FF(MAXPTS),UU(MAXPTS),QQ(MAXPTS)
INTEGER DEGREE(MAXPTS)
COMMON/DEG/DEGREE

J = 1
MIN1 = DIST(1,NUMVTX)
DO I = 2,NUMVTX - 1
  IF(MIN1 .GT. DIST(I,NUMVTX)) THEN
    MIN1 = DIST(I,NUMVTX)
    J = I
  END IF
END DO

IF(J .NE. 1) THEN
  K = 1
ELSE

84
K = 2
END IF

MIN2 = DIST(K, NUMVTX)
DO I = K+1, NUMVTX - 1
  IF(MIN2 .GT. DIST(I, NUMVTX) .AND. I .NE. J) THEN
    MIN2 = DIST(I, NUMVTX)
    K = I
  END IF
END DO

NUMVTX = NUMVTX - 1

TIME0 = CTIME()
CALL PRIM(QQ, UU, FF)
DO I = 1, NUMVTX
  DEGREE(I) = 0
END DO
DO I = 2, NUMVTX
  DEGREE(FF(I)) = DEGREE(FF(I)) + 1
  DEGREE(I) = DEGREE(I) + 1
END DO
DEGREE(J) = DEGREE(J) + 1
DEGREE(K) = DEGREE(K) + 1
DEGREE(NUMVTX+1) = 2
TIME1 = CTIME() - TIME0
WRITE(20,11) NUMVTX+1
FORMAT(IX,'NUMBER OF NODES = ',I3)
WRITE(20,12) TIME1
FORMAT(IX,'CPU TIME = ',F5.2)
WRITE(20,*)
WRITE(20,13)
FORMAT(IX, ' F(I) : PREDECESSOR OF NODE I : ')
WRITE(20,'(10I6)')( FF(I), I = 1, NUMVTX)
WRITE(20,'(2I6)') J, K
WRITE(20,*)
WRITE(20,14)
FORMAT(IX, ' W(I,F(I)) : WEIGHT OF EDGES : ')
WRITE(20,'(10I6)')( UU(I), I = 1, NUMVTX)
WRITE(20,'(2I6)') MIN1, MIN2

85
J = 0
DO I = 1,NUMVTX
    J = J + UU(I)
END DO

WRITE(20,*)
WRITE(*,15) J+MIN1+MIN2
WRITE(*,15) J+MIN1+MIN2
FORMAT(1X,' TOTAL WEIGHT = ',I9)
lowerb=j+min1+min2
NUMVTX = NUMVTX+1
ID = 0
DO I = 1,NUMVTX
    PRINT*,' DEGREE OF',I,' = DEGREE(I)
    ID = ID + DEGREE(I)
END DO
PRINT*, ' TOTAL DEGREE = ',ID
END ! END OF ONETREE

C ********************************************************** SUBROUTINE PRIM **********************************************************
C *** (SHORTEST SPANNING TREE PROBLEM) ***
C *** THE PROGRAM IS BASED ON THE PAPER ***
C *** P. M. CAMERINI, G. M. GALBIATI, F. MAFFIOLI ***
C *** "ALGORITHMS FOR FINDING OPTIMUM TREES: ***
C *** DESCRIPTIONS, USE AND EVALUATION", ***
C *** ANNALS OF OPERATIONS RESEARCH 7, 1988. ***
C ***
C ********************************************************** SUBROUTINE PRIM (Q,U,F) **********************************************************
C SUBROUTINE PRIM COMPUTES A SPANNING TREE OF MINIMUM TOTAL
C WEIGHT BY PRIM'S METHOD
C THE GRAPH MUST BE SIMPLE AND COMPLETE
C
C WARNINGS
C
C MATRIX OF ARC WEIGHTS MUST BE COMPLETE AND SYMMETRIC
SUBROUTINE PRIM DOES NOT CHECK COMPLETENESS NOR SYMMETRY

OUTPUT ARGUMENTS

F(I)   PREDECESSOR OF NODE I IN THE OUTWARDS ORIENTED TREE WITH ROOT
       IF N IS LESS THAN OR EQUAL TO 1, PRIM RETURNS WITH F(1)=0
       (DIM. NRD)

WORKING ARGUMENTS

Q(.)   VECTOR OF ISOLATED NODES (DIM. NRD)
U(.)   VECTOR OF CURRENT NODE WEIGHS (DIM. NRD)

LOCAL VARIABLES

K      ISOLATED NODE CHOSEN
I      DO-LOOP INDEX
J      DO-LOOP INDEX
JJ     CONTAINER OF Q(J)
QO     CARDINALITY OF Q

CALLED FUNCTION

PMIN   RETURNS A NODE OF MINIMUM WEIGHT

TYPE AND DIMENSION STATEMENTS

INTEGER MAXPTS
PARAMETER(MAXPTS = 101)
COMMON/MATRIX/DIST,NUMVTX,LOWERB
INTEGER DIST(MAXPTS,MAXPTS),NUMVTX,LOWERB
INTEGER F(MAXPTS),Q(MAXPTS),QO,PMIN,U(MAXPTS)

DATA STRUCTURES INITIALIZATION

F(1) = 0
QO = NUMVTX - 1
IF ( NUMVTX . LE . 1 ) GO TO 40
DO 1000 I=2,NUMVTX
   Q(I-1) = I
   F(I) = 1
   U(I) = DIST(I,I)
1000 CONTINUE

C MAIN ITERATION: DETERMINATION OF AN ARC OF MINIMUM WEIGHT
C CONNECTING A NODE IN THE TREE TO AN EXTERNAL NODE. THIS
C ARC IS THEN ADDED TO THE TREE.
C
20 IF (QO .LE. 1) GO TO 40
   K = PMIN(U,Q,QO)
C
C UPDATING OF LABELS = CLEANUP OPERATION
C
DO 1030 J=1,QO
   JJ = Q(J)
   IF (U(JJ) .LE. DIST(K,JJ)) GO TO 1030
   U(JJ) = DIST(K,JJ)
   F(JJ) = K
1030 CONTINUE
GO TO 20
40 RETURN
END

INTEGER FUNCTION PMIN (U,Q,QO)

FUNCTION PMIN EXTRACTS FROM Q AND RETURNS AN ISOLATED NODE K
OF MINIMUM WEIGHT U(K) IF QO IS GREATER THAN 0, ELSE RETURNS 0.

INPUT ARGUMENTS

U(.) VECTOR OF NODE WEIGHTS

INPUT/OUTPUT ARGUMENTS

Q(.) VECTOR OF ISOLATED NODES

QO CARDINALITY OF VECTOR Q

LOCAL VARIABLES

88
C
C I  DO-LOOP INDEX
C II  CURRENT NAME FOR Q(I)
C P  POSITION OF CURRENT OPTIMAL NODE
C
C TYPE AND DIMENSION STATEMENTS
C
INTEGER MAXPTS
PARAMETER(MAXPTS = 101)
COMMON/MATRIX/DIST,NUMVTX,LOWEBR
INTEGER DIST(MAXPTS,MAXPTS),NUMVTX,LOWEBR

INTEGER Q(MAXPTS),Q0,P,U(MAXPTS)

C
IF ( Q0 .GT. 0 ) GO TO 10
  PMIN = 0
RETURN
10 CONTINUE
  P = 1
  PMIN = Q(1)
  DO 1000 I=1,Q0
    II = Q(I)
    IF ( U(II) .GE. U(PMIN) ) GO TO 1000
      P = I
      PMIN = II
  1000 CONTINUE
  Q(P) = Q(Q0)
  Q0 = Q0 - 1
RETURN
END

********************************************************************************
********************************************************************************

SUBROUTINE HEURISTIC_4(nnn,ddd,min_tsp)

INTEGER*4 MAXPTS, BIGINT,min_tsp
PARAMETER (MAXPTS = 101, BIGINT = 999999)

89
INTEGER*4 MAXOPT
PARAMETER (MAXOPT = 30)
COMMON /POINTS/ X(MAXPTS), Y(MAXPTS), NUMPTS, MATSIZ
COMMON /DISTS / D(MAXPTS,MAXPTS), DM(MAXPTS,MAXPTS)
COMMON /LISTS / IROUTE(MAXPTS), ILIST(MAXPTS,3)
COMMON /OUTCOM/ FUNIT
COMMON/PRNSW/ KPRINT(10)
INTEGER*4 X, Y, NUMPTS, MATSIZ, NNN
INTEGER*4 D, DM, d(maxpts, maxpts)
INTEGER*4 IROUTE, ILIST
INTEGER*4 FUNIT

COMMON /keep/ dk(maxpts, maxpts)
integer*4 dk

C *** END OF BLOCK ***
C

DIMENSION TIMARR(20)
INTEGER*4 I, J, NTREES, LTREE, UTREE, OPTION
INTEGER*4 SEED, THRTOT, LIMTOT, DIFF
REAL SECOND(MAXOPT)
INTEGER*4 TOTALS(MAXOPT), MINUTE(MAXOPT), RNUMB(MAXOPT)
INTEGER*4 KSWIT(10)
CHARACTER*8 RNAME(MAXOPT)
CHARACTER*50 OFILE, SFILE

C
C *** BEGIN ***
C

C GET PROGRAM PARAMETERS AND GENERATING DISTANCE MATRIX
C

min_tsp = 999999999

kprint(1) = 1
kprint(2) = 1
numpts = nnn
DO 10 I = 1,NUMPTS
    d(i,i) = ddd(i,i)
    dk(i,i) = d(i,i)
    dm(i,i) = d(i,i)
10    continue

numptsless1=numpts-1
DO 30 I = 1,NUMPTSless1
    iand1=i+1
    DO 20 J = iand1,NUMPTS
        d(i,j) = ddd(i,j)
        d(j,i) = d(i,j)
        dk(i,j) = d(i,j)
        dk(j,i) = d(i,j)
        dm(i,j) = d(i,j)
        dm(j,i) = d(i,j)
20    continue
30    continue

LTREE = 12
UTREE = 12

print *,'going into first heuristic'

CALL SIMPTR
RNAME(l) = ' THREE '
print *,'back from simptr'
RNUMB(l) = 0
CALL THREE
print *,'back from three'
CALL TOURLN(TOTALS(l))

min_tsp=totals(l)
return
C print *, 'back from tourln'
C WRITE(1,'(1I0)') TOTALS(1)
    if(min_tsp.gt.totals(1)) min_tsp=totals(1)
C
C GET AND MEASURE THE "THROPT" TOUR.
C
    CALL SIMPTR
    RNAME(2) = ' THROPT'
    RNUMB(2) = 0
    CALL THROPT
    CALL TOURLN(TOTALS(2))
C WRITE(1,'(1I0)') TOTALS(2)
    if(min_tsp.gt.totals(2)) min_tsp=totals(2)
C
C GET AND MEASURE THE "OROPT" TOUR.
C
    CALL SIMPTR
    RNAME(3) = ' OROPT'
    RNUMB(3) = 0
    CALL OROPT
    CALL TOURLN(TOTALS(3))
C WRITE(1,'(1I0)') TOTALS(3)
    if(min_tsp.gt.totals(3)) min_tsp=totals(3)
OPTION = 3

C GET AND MEASURE VARIOUS "ARCOPT" TOURS.
C
DO 50 J=2,LTREE
    CALL GETSST
50 CONTINUE
DO 100 NTREES=LTREE, UTREE
    CALL GETSST
    CALL COPYMT(1,2)
    CALL SIMPTR
    CALL SMPTR2
    CALL REDOMT
    CALL MAKLST
    OPTION = OPTION + 1
    RNAME(OPTION) = ' ARCOPT'
    RNUMB(OPTION) = NTREES
92
CALL ARCOPT
CALL TOURLN(TOTALS(OPTION))
if(min_tsp.gt.totals(option))min_tsp=totals(option)
CALL COPYMT\n
100 CONTINUE
C
C PRINT OUT THE TOTAL CPU TIME FOR THE RUN, CLOSE FILES, AND QUIT
C
MOST = INT((OPTION - 1) / 5) + 1
DO 200 J = 1, MOST
   JLO = (5*(J-1))+1
   JHI = (5*J)
   IF (JHI.GT.OPTION) JHI = OPTION
   WRITE(FUNIT,1) (RNAME(I), I=JLO, JHI)
   WRITE(FUNIT,4) (RNUMB(I), I=JLO, JHI)
   WRITE(FUNIT,2) (TOTALS(I), I=JLO, JHI)
   WRITE(*,1) (RNAME(I), I=JLO, JHI)
   WRITE(*,2) (TOTALS(I), I=JLO, JHI)
   WRITE(FUNIT,*),
   WRITE(FUNIT,*),
   200 CONTINUE
   return
read *
CLOSE(FUNIT)
C
1 FORMAT(5A10)
2 FORMAT(5I10)
3 FORMAT(5(I3,'m',F5.2,'s'))
4 FORMAT(5(I8,2X))
57 FORMAT(10F7.2)
C
STOP
END

SUBROUTINE REDOMT
C ************************************************ REDOMAT ************************************************
C
C REARRANGES MATRIX "D".

C
C *** INSERTION COMMON BLOCK ***
C
INTEGER+4 MAXPTS, BIGINT
PARAMETER (MAXPTS = 101, BIGINT = 999999)
COMMON /POINTS/ X(MAXPTS), Y(MAXPTS), NUMPTS, MATSIZ
COMMON /DISTS / D(MAXPTS,MATXMPTS), DM(MAXPTS,MATXMPTS)
COMMON /LISTS / IROUTE(MAXPTS), ILIST(MAXPTS,3)
COMMON /OUTCOM/ FUNIT
COMMON/PRNSW/ KPRINT(10)
INTEGER+4 X, Y, NUMPTS, MATSIZ
INTEGER+4 D, DM
INTEGER+4 IROUTE, ILIST
INTEGER+4 FUNIT
C
C *** END OF BLOCK ***
C
LOGICAL WAY
INTEGER+4 I, J
C
C *** BEGIN ***
C
C
C IF(KPRINT(2).EQ.1) WRITE(1,*),' REDOMT'
C DO 50 I=1,NUMPTS
C D(ROUTE(I),LIST(ROUTE(I),2)) = BIGINT
C D(LIST(ROUTE(I),2),ROUTE(I)) = BIGINT
C 50 CONTINUE
C DO 60 I=1,NUMPTS
C DO 70 J=1,NUMPTS
C IF (I.EQ.J) GO TO 70
C IF (D(I,J).EQ.BIGINT) THEN
C D(I,J) = DM(I,J)
C ELSE
C D(I,J) = BIGINT
C ENDIF
C 70 CONTINUE
C 60 CONTINUE
C RETURN
C END
SUBROUTINE GETSST
C ************************************** GETSST **************************************
C
C THIS ROUTINE MAKES ONE MINIMAL SPANNING TREE FROM ALL THE ARCS IN DK() WHOSE VALUES ARE LESS THAN BIGINT.
C
C
C SHORTEST SPANNING TREE USING PRIM'S ALGOR
C REFERENCE CHRISTOFIDES "GRAPH THEORY: AN ALGORITHMIC APPROACH" PP 138-139
C
C
C NN ------ NUMBER OF NODES
C IT(N) --- START NODE OF ARC N IN 'SST'
C JT(N) --- END NODE OF ARC N IN 'SST'
C COST(N) - COST OF ARC N IN 'SST'
C IALPHA(J) CLOSEST NODE IN TREE TO NODE J NOT IN TREE
C = 0 IF NODE J ALREADY IN TREE
C BETA(J) - DISTANCE TO CLOSEST NODE IN TREE FROM NODE J
C NARC ---- NUMBER OF ARCS IN 'SST' NARC = NN - 1
C BETAMN -- DISTANCE TO CLOSEST NODE NOT IN TREE FROM TREE
C
C
C THIS PROGRAM USES THE DISTANCE MATRIX D(I,J) TO GET THE COSTS BETWEEN NODES I AND J
C
C *** INSERTION COMMON BLOCK ***
C
INTEGER*4 MAXPTS, BIGINT
PARAMETER (MAXPTS = 101, BIGINT = 999999)
COMMON /POINTS/ X(MAXPTS), Y(MAXPTS), NUMPTS, MATSIZ
COMMON /DISTS/ D(MAXPTS,MAXPTS), DM(MAXPTS,MAXPTS)
COMMON /LISTS/ IROUTE(MAXPTS), ILIST(MAXPTS,3)
COMMON /OUTCOM/ FUNIT
COMMON/PRNSW/ KPRINT(10)
INTEGER*4 X, Y, NUMPTS, MATSIZ
INTEGER*4 D, DM
INTEGER*4 IROUTE, ILIST
INTEGER*4 FUNIT
C
C *** END OF BLOCK ***
C
COMMON/KEEP/DK(MAXPTS,MAXPTS)
INTEGER DK
DIMENSION IT(200),JT(200),COST(200)
DIMENSION IALPHA(200),BETA(200)
INTEGER*4 BETAMN,BETA,COST

C
C *** BEGIN ***
C
C INITIALIZE IALPHA(J) AND BETA(J)
C
C IF(KPRINT(2) .EQ. 1) WRITE(1,*), ' GETSST'
DO 100 J = 2,NUMPTS
   IALPHA(J) = 1
   IF(DK(1,J) .GE. 0) BETA(J) = DK(1,J)
   IF(DK(1,J) .LT. 0) BETA(J) = BIGINT
CONTINUE
NARC = NUMPTS - 1
DO 3000 NA = 1,NARC
   BETAMN = BIGINT
   DO 2000 J = 2,NUMPTS
      IF(IALPHA(J) .EQ. 0) GO TO 2000
      IF(BETAMN .LE. BETA(J)) GO TO 2000
      IADD = J
      BETAMN = BETA(J)
   CONTINUE
   I = IALPHA(IADD)
   IALPHA(IADD) = 0
   IT(NA) = I
   JT(NA) = IADD
   DK(I,IADD) = -BETA(IADD) - 1
   DK(IADD,I) = -BETA(IADD) - 1
   COST(NA) = BETA(IADD)
   DO 2500 K = 2,NUMPTS
      IF(IALPHA(K) .EQ. 0) GO TO 2500
      IF(DK(IADD,K) .LT. 0) GO TO 2500
      IF(DK(IADD,K) .GE. BETA(K)) GO TO 2500
      IALPHA(K) = IADD
      BETA(K) = DK(IADD,K)
   CONTINUE
2500 CONTINUE
3000 CONTINUE
DO 4000 N = 1,NARC
   I = IT(N)
   J = JT(N)
   DK(I,J) = BIGINT
   DK(J,I) = BIGINT
   IF(KPRINT(8) .EQ. 1) WRITE(1,'(3I8)') I,J,D(I,J)
4000 CONTINUE

RETURN
END

SUBROUTINE COPYMT(MAT1,MAT2)
C *************♦******♦+**=)=**♦********** COPYMT ********************** ********
C
C THIS ROUTINE COPIES "MAT2" INTO "MAT1".
C
C 1 : "D"
C 2 : "DK"
C 3 : "DM"
C
INTEGER*4 MAXPTS, BIGINT
PARAMETER (MAXPTS = 101, BIGINT = 999999)
COMMON /DISTS / D(MAXPTS,MAXPTS), DM(MAXPTS,MAXPTS)
INTEGER*4 D, DM
COMMON /KEEP / DK(MAXPTS,MAXPTS)
INTEGER*4 DK
INTEGER*4 MAT1, MAT2, M2
INTEGER*4 I, J
COMMON /PRNSW/ KPRINT(IO)
C
C *** BEGIN ***
C
C IF(KPRINT(2) .EQ. 1) WRITE(1,*), COPYMAT'
DO 10 I=1, MAXPTS
   DO 20 J=1, MAXPTS
      IF (MAT2.EQ.1) M2 = D(I,J)
      IF (MAT2.EQ.2) M2 = DK(I,J)
      IF (MAT2.EQ.3) M2 = DM(I,J)
10 CONTINUE
20 CONTINUE
IF (MAT1.EQ.1) D(I,J) = M2
IF (MAT1.EQ.2) DK(I,J) = M2
IF (MAT1.EQ.3) DM(I,J) = M2
20 CONTINUE
10 CONTINUE
RETURN
END

SUBROUTINE MAKLST

C ************************************************************************** MAKMAT **************************************************************************
C
C THE ROUTINE FILLS THE "ARCLST" VARIABLES WITH THE PROPER VALUES
C FROM THE "D" MATRIX.
C
C  *** INSERTION COMMON BLOCK ***
C
INTEGER+4 MAXPTS, BIGINT
PARAMETER (MAXPTS = 101, BIGINT = 999999)
COMMON /POINTS/ X(MAXPTS), Y(MAXPTS), NUMPTS, MATSIZ
COMMON /DISTS / D(MAXPTS,M ATPS), DM(MAXPTS,M ATPS)
COMMON /LISTS / IROUTE(MAXPTS), ILIST(MAXPTS,3)
COMMON /OUTCOM/ FUNIT
COMMON/PRNSW/ KPRINT(10)
INTEGER+4 X, Y, NUMPTS, MATSIZ
INTEGER+4 D, DM
INTEGER+4 IROUTE, ILIST
INTEGER+4 FUNIT
C
C  *** END OF BLOCK ***
C
C ARC LIST VARIABLES
C
COMMON /ARCCOM/ MAP(MAXPTS),IBEG(MAXPTS),
+ ADJ(75*MAXPTS),C(75*MAXPTS),
+ COST(4,3)
INTEGER+4 MAP, IBEG, ADJ, C, COST
C
INTEGER+4 I, J, COUNT, LAST
C
C  *** BEGIN ***
C IF(KPRINT(2) .EQ. 1) WRITE(1,*),' MAKLST'
COUNT = 0
LAST = 0
DO 10 I=1, NUMPTS
  DO 20 J=1, NUMPTS
    IF (I.EQ.J) GO TO 20
    IF (D(I,J).GE.BIGINT) GO TO 20
    COUNT = COUNT + 1
    C(COUNT) = D(I,J)
    ADJ(COUNT) = J
    IF (COUNT.GT.(75*MAXPTS)) GO TO 30
    IF (LAST.NE.I) THEN
      LAST = I
      IBEG(I) = COUNT
    ENDIF
  CONTINUE
10 CONTINUE
IBEG(NUMPTS+1) = COUNT+1
J = NUMPTS
DO 50 K=1,NUMPTS
  I = ILIST(J,2)
  ILIST(J,3) = D(J,I)
  J = I
50 CONTINUE
GO TO 40
30 WRITE(FUNIT,*) '*** ERROR IN MAKLST'
WRITE(FUNIT,*) 'TOO MANY ARCS TO MAKE THE LISTS'
WRITE( * ,*) '*** ERROR IN MAKLST'
WRITE( * ,*) 'TOO MANY ARCS TO MAKE THE LISTS'
40 RETURN
END

SUBROUTINE OUTLST
C ************************************************ OUTLST ************************************************
C
C *** INSERTION COMMON BLOCK ***
C
INTEGER*4 MAXPTS, BIGINT
PARAMETER (MAXPTS = 101, BIGINT = 999999)
COMMON /POINTS/ X(MAXPTS), Y(MAXPTS), NUMPTS, MATSIZ
COMMON /DISTS / D(MAXPTS,MAXPTS), DM(MAXPTS,MAXPTS)
COMMON /LISTS / IROUTE(MAXPTS), ILIST(MAXPTS,3)
COMMON /OUTCOM/ FUNIT
COMMON/PRNSW/ KPRINT(10)
INTEGER*4 X, Y, NUMPTS, MATSIZ
INTEGER*4 D, DM
INTEGER*4 IROUTE, ILIST
INTEGER*4 FUNIT

C
C *** END OF BLOCK ***
C
C ARC LIST VARIABLES
C
COMMON /ARCCOM/ MAP(MAXPTS), IBEG(MAXPTS),
+ ADJ(75*MAXPTS), C(75*MAXPTS),
+ COST(4,3)
INTEGER*4 MAP, IBEG, ADJ, C, COST

C
INTEGER*4 I, J, COUNT, LAST

C *** BEGIN ***
C
IF(KPRINT(2) .EQ. 1) WRITE(1,*),'OUTLIST'
DO 10 I=1,NUMPTS
  TOP = IBEG(I)
  BOT = IBEG(I+1) - 1
  WRITE(FUNIT,*),'','
  WRITE(FUNIT,*),'NODE ',I
  DO 20 J=TOP, BOT
    WRITE(FUNIT,'(2I10)') ADJ(J), C(J)
20     CONTINUE
10  CONTINUE
RETURN
END

SUBROUTINE TOURLN(TOTAL)
C ********************************************** TOURLN **********************************************
C
C FIND THE LENGTH OF THE SOLUTION IN "IROUTE".

100
C
C *** INSERTION COMMON BLOCK ***
C
INTEGER*4 MAXPTS, BIGINT
PARAMETER (MAXPTS = 101, BIGINT = 999999)
COMMON /POINTS/ X(MAXPTS), Y(MAXPTS), NUMPTS, MATSIZ
COMMON /DISTS / D(MAXPTS,MAXPTS), DM(MAXPTS,MAXPTS)
COMMON /LISTS / IROUTE(MAXPTS), ILIST(MAXPTS,3)
COMMON /OUTCOM/ FUNIT
COMMON/PRNSW/ KPRINT(10)
INTEGER*4 X, Y, NUMPTS, MATSIZ
INTEGER*4 D, DM
INTEGER*4 IROUTE, ILIST
INTEGER*4 FUNIT
C
C *** END OF BLOCK ***
C
INTEGER*4 TOTAL, I
C
C *** BEGIN ***
C
WRITE(1,*) ' TOURLN'
WRITE(1,'(10I6)') (IROUTE(I), 1=1,NUMPTS)
TOTAL = 0
DO 10 I = 1,NUMPTS
   TOTAL = TOTAL + DM(I,ILIST(I,2))
10 CONTINUE
RETURN
END

SUBROUTINE SIMPTR
C **************************************** SIMPTR ****************************************
C
C GENERATE A TOUR FROM WHICH THE ROUTINES CAN WORK AND STORE
C THEM IN THE "IROUTE" AND "ILIST" ARRAYS.
C
C THE TOUR WILL BE 1-2-3-4--NUMPTS.
C
C *** INSERTION COMMON BLOCK ***
C
101
INTEGER*4 MAXPTS, BIGINT
PARAMETER (MAXPTS = 101, BIGINT = 999999)
COMMON /POINTS/ X(MAXPTS), Y(MAXPTS), NUMPTS, MATSIZ
COMMON /DISTS / D(MAXPTS,MAXPTS), DM(MAXPTS,MAXPTS)
COMMON /LISTS / IROUTE(MAXPTS), ILIST(MAXPTS,3)
COMMON /OUTCOM/ FUNIT
COMMON/PRNSW/ KPRINT(10)
INTEGER*4 X, Y, NUMPTS, MATSIZ
INTEGER*4 D, DM
INTEGER*4 IROUTE, ILIST
INTEGER*4 FUNIT

C
C  END OF BLOCK ***
C
C  BEGIN ***
C
IF(KPRINT(2) .EQ. 1) WRITE(1,*) ' SIMPTR'
J = NUMPTS
DO 30 I = 1,NUMPTS
   IROUTE(I) = I
   ILIST(J,2) = I
   ILIST(I,1) = J
   J = I
30 CONTINUE
RETURN
END

SUBROUTINE SMPTR2
C ******************************************** SMPTR2 ********************************************
C
C GENERATE A TOUR FROM WHICH THE ROUTINES CAN WORK AND STORE THEM IN THE "IROUTE" AND "ILIST" ARRAYS.
C
C THE TOUR WILL BE 1-2-3-4-...-NUMPTS.
C
C *** INSERTION COMMON BLOCK ***
C
INTEGER*4 MAXPTS, BIGINT
PARAMETER (MAXPTS = 101, BIGINT = 999999)
COMMON /POINTS/ X(MAXPTS), Y(MAXPTS), NUMPTS, MATSZ
COMMON /DISTS / D(MAXPTS,MAXPTS), DM(MAXPTS,MAXPTS)
COMMON /LISTS / IROUTE(MAXPTS), ILIST(MAXPTS,3)
COMMON /OUTCOM/ FUNIT
COMMON/PRNSW/ KPRINT(10)
INTEGER*4 X, Y, NUMPTS, MATSZ
INTEGER*4 D, DM
INTEGER*4 IROUTE, ILIST
INTEGER*4 FUNIT

C
C *** END OF BLOCK ***
C
INTEGER*4 I, J

C *** BEGIN ***
C
C IF(KPRINT(2) .EQ. 1) WRITE(1,*), ' SMPTR2'
K = 0
JJ = NUMPTS
DO 40 J = 1,10
   DO 30 I = 1,NUMPTS,10
      L = I + J - 1
      IF(L .GT. NUMPTS) GO TO 30
      K = K+ 1
      IROUTE(K) = L
      ILIST(JJ,2) = L
      ILIST(L,1) = JJ
      JJ = L
30     CONTINUE
40    CONTINUE
L = IROUTE(K)
ILIST(L,2) = 1
ILIST(1,1) = L
DO 50 K = 1,NUMPTS
   I = IROUTE(K)
C      IF(KPRINT(6) .EQ. 1) WRITE(1,51) K,I,ILIST(I,2),ILIST(I,1)
50    CONTINUE
51 FORMAT(4I6)
RETURN
END
SUBROUTINE ARCOPT
C ******************************************** ARCOPT ********************************************
IMPLICIT INTEGER*4 (A-Z)
C
INTEGER*4 MAXPTS, BIGINT
PARAMETER (MAXPTS = 101, BIGINT = 999999)
COMMON /POINTS/ X(MAXPTS), Y(MAXPTS), NUMPTS, MATSIZ
COMMON /DISTS / D(MAXPTS,MAXPTS), DM(MAXPTS,MAXPTS)
COMMON /LISTS / IROUTE(MAXPTS), ILIST(MAXPTS,3)
COMMON /OUTCOM/ FUNIT
COMMON/PRNSW/ KPRINT(IO)
INTEGER*4 X, Y, NUMPTS, MATSIZ
INTEGER*4 D, DM
INTEGER*4 IROUTE, ILIST
INTEGER*4 FUNIT
C
COMMON /LISTTT/ LIST(MAXPTS,2), LISTNO, ISAVD(MAXPTS), NEW
INTEGER*4 LIST, LISTNO, ISAVD, NEW
C
INTEGER*4 SAVMAX, DIST, DIST3, DIFF, KSW
C
COMMON /ARCCOM/ MAP(MAXPTS),IBEG(MAXPTS),
+ ADJ(75*MAXPTS),C(75*MAXPTS),
+ COST(4,3)
INTEGER*4 MAP, IBEG, ADJ, C, COST
C
C *** BEGIN ***
C
C IF(KPRINT(2) .EQ. 1) WRITE(1,*) ' ARCOPT'
NM    = NUMPTS - 1
KSW    = 5
SAVMAX = 0
DO 90 I = 1,NUMPTS
   II = IROUTE(I)
   MAP(II) = I
   JJ = ILIST(II,2)
   LIST(I,1) = II
   LIST(I,2) = JJ
90 CONTINUE
LISTNO = NUMPTS.

100 I1 = LIST(LISTNO,1)
I2 = ILIST(I1,2)
IF (I2 .EQ. LIST(LISTNO,2)) GO TO 300
I1 = LIST(LISTNO,2)
I2 = ILIST(I1,2)
IF (I2 .EQ. LIST(LISTNO,1)) GO TO 300
WRITE(6,55)
55 FORMAT(3X,'ERROR—LIST EMPTY')
STOP

300 LISTNO = LISTNO - 1
DO 325 L = 1,NUMPTS
   IF (MAP(L) .LT. MAP(I1)) MAP(L) = MAP(L) + NUMPTS
325 CONTINUE

IB1 = IBEG(I1)
IE1 = IBEG(I1+1) - 1
IB2 = IBEG(I2)
IE2 = IBEG(I2+1) - 1

DO 1000 IS1 = IB1, IE1
   IN1 = ADJ(IS1)
   IF (IN1 .EQ. I2) GO TO 1000

DO 800 IS2 = IB2, IE2
   IN2 = ADJ(IS2)
   IF (IN2 .EQ. I1) GO TO 800

J1 = IN1
K1 = IN2
J2 = ILIST(J1,2)
IF (MAP(K1).LT.MAP(J2)) GO TO 400
IF (J2 .EQ. I1) GO TO 400
K2 = ILIST(K1,2)
IF (D(J2,K2) .EQ.BIGINT) GO TO 400

DIFF = ILIST(I1,3) + ILIST(J1,3) + ILIST(K1,3) -
   (C(IS1) + C(IS2) + D(J2,K2))
IF (DIFF.LE.SAVMAX) GO TO 400
SAVMAX = DIFF
KSW = 1
COST(KSW,1) = C(IS1)
COST(KSW,2) = C(IS2)
COST(KSW,3) = D(J2,K2)
GO TO 700

400
J2 = IN1
K1 = IN2
J1 = ILIST(J2,1)
IF (MAP(K1).LT.MAP(J2)) GO TO 500
K2 = ILIST(K1,2)
IF (D(J1,K2) .EQ.BIGINT) GO TO 500

DIFF = ILIST(I1,3) + ILIST(J1,3) + ILIST(K1,3) - (C(IS1) + C(IS2) + D(J1,K2))
+ IF (DIFF.LE.SAVMAX) GO TO 500
SAVMAX = DIFF
KSW = 2
COST(KSW,1) = C(IS1)
COST(KSW,2) = C(IS2)
COST(KSW,3) = D(J1,K2)
GO TO 700

500
J2 = IN1
K2 = IN2
J1 = ILIST(J2,1)
K1 = ILIST(K2,1)
IF (MAP(K1).LT.MAP(J2)) GO TO 600
IF (D(J1,K1) .EQ.BIGINT) GO TO 600

DIFF = ILIST(I1,3) + ILIST(J1,3) + ILIST(K1,3) - (C(IS1) + C(IS2) + D(J1,K1))
+ IF (DIFF.LE.SAVMAX) GO TO 600
SAVMAX = DIFF
KSW = 3
COST(KSW,1) = C(IS1)
COST(KSW,2) = D(J1,K1)
COST(KSW,3) = C(IS2)
GO TO 700

600
K1 = IN1
J2 = IN2
J1 = ILIST(J2,1)
K2 = ILIST(K1,2)
IF (MAP(K1).LT.MAP(J2)) GO TO 700
IF (D(J1,K2) .EQ.BIGINT) GO TO 700

DIFF = ILIST(I1,3) + ILIST(J1,3) + ILIST(K1,3) -
      (C(IS1) + C(IS2) + D(J1,K2))
IF (DIFF.LE.SAVMAX) GO TO 700
SAVMAX = DIFF
KSW = 4
COST(KSW,1) = C(IS1)
COST(KSW,2) = C(IS2)
COST(KSW,3) = D(J1,K2)

C
C
C
C
C
C
C
C
C
C
C
C
C
C
C
C
C

700 IF (KSW.EQ.5) GO TO 709
IF (LISTNO.EQ.0) GO TO 709
NSW = 1
N1 = I1
N2 = I2
701 LADD = 0
DO 704 L = 1, LISTNO
   LP = L - LADD
   IF (LIST(L,1).EQ.N1.AND.LIST(L,2).EQ.N2) GO TO 702
   IF (LIST(L,2).EQ.N1.AND.LIST(L,1).EQ.N2) GO TO 702
   LIST(LP,1) = LIST(L,1)
   LIST(LP,2) = LIST(L,2)
   GO TO 704
702 LADD = 1
704 CONTINUE
LISTNO = LISTNO - LADD
GO TO (706, 707, 709), NSW
706 NSW = 2
N1 = J1
N2 = J2
GO TO 701
707 NSW = 3
N1 = K1
N2 = K2
SUBROUTINE AREARR MAKES THE ARC SWAPS FOR ARCOPT

CALL AREARR(I1,J1,I2,K1,J2,K2,KSW)
GO TO 750
CALL AREARR(I1,J2,K1,I2,J1,K2,KSW)
GO TO 750
CALL AREARR(I1,J2,K1,J1,I2,K2,KSW)
GO TO 750
CALL AREARR(I1,K1,J2,I2,J1,K2,KSW)
SAVMAX = 0
KSW = 5
GO TO 100
CONTINUE
IF (LISTNO.GT.0) GO TO 100
RETURN
END

SUBROUTINE AREARR(I1,I2,J1,J2,K1,K2,KSW)

C ************** AREARR **************

C THIS ROUTINE REARRANGES THE SEQUENCE OF NODES IN IROUTE. IT
C IS DESIGNED FOR USE WITH THE ARCOPT ROUTINE ONLY.

C

IMPLICIT INTEGER*4 (A-Z)

C

*** INSERTION COMMON BLOCK ***

C

INTEGER*4 MAXPTS, BIGINT
PARAMETER (MAXPTS = 101, BIGINT = 999999)
COMMON /POINTS/ X(MAXPTS), Y(MAXPTS), NUMPTS, MATSIZ
COMMON /DISTS/ D(MAXPTS,MAXPTS), DM(MAXPTS,MAXPTS)
COMMON /LISTS/ IROUTE(MAXPTS), ILIST(MAXPTS,3)
COMMON /OUTCOM/ FUNIT
COMMON/PRNSW/ KPRINT(10)
INTEGER*4 X, Y, NUMPTS, MATSIZ
INTEGER*4 D, DM
INTEGER*4 IROUTE, ILIST
INTEGER*4 FUNIT

C
C *** END OF BLOCK ***
C
C STORAGE FOR ARC LISTS FOR USE BY "THREE"
C
COMMON /LISTTT/ LIST(MAXPTS,2), LISTNO, ISAVD(MAXPTS), NEW
INTEGER*4 LIST, LISTNO, ISAVD, NEW
C
C ARC LIST VARIABLES
C
COMMON /ARCCOM/ MAP(MAXPTS), IBEG(MAXPTS),
+           ADJ(75*MAXPTS), C(75*MAXPTS),
+           COST(4,3)
INTEGER*4 MAP, IBEG, ADJ, C, COST
C
C *** BEGIN ***
C
ILIST(I1,2) = I2
ILIST(I1,3) = COST(KSW,1)
IF((KSW.EQ.2).OR.(KSW.EQ.3)) GO TO 200
C
C REORDER ROUTE FROM I2 TO J1
C
I = I2
ILSAV = ILIST(I,1)
CSTSAV = ILIST(ILSAV,3)
ILIST(I,1) = I1
GO TO 160
150 ILSAV = ILIST(I,1)
CSTSAV = ILIST(ILSAV,3)
ILIST(I,1) = ILIST(I,2)
160 ILIST(I,2) = ILSAV
ILIST(I,3) = CSTSAV
IF(I.EQ.J1) GO TO 200
I = ILSAV
GO TO 150
200 ILIST(J1,2) = J2
ILIST(J1,3) = COST(KSW,2)
ILIST(I2,1) = I1
IF((KSW.EQ.2).OR.(KSW.EQ.4)) GO TO 400

C
C REORDER ROUTE FROM J2 TO K1
C
J = J2
JLSAV = ILIST(J,1)
CSTSAV = ILIST(JLSAV,3)
ILIST(J,1) = J1
GO TO 360

350 JLSAV = ILIST(J,1)
CSTSAV = ILIST(JLSAV,3)
ILIST(J,1) = ILIST(J,2)

360 ILIST(J,2) = JLSAV
ILIST(J,3) = CSTSAV
IF(J.EQ.K1) GO TO 400
J = JLSAV
GO TO 350

400 ILIST(K1,2) = K2
ILIST(K1,3) = COST(KSW,3)
ILIST(J2,1) = J1
ILIST(K2,1) = K1
I = IROUTE(1)
MAP(I) = 1
DO 800 L = 2,NUMPTS
    IROUTE(L) = ILIST(I,2)
    I = IROUTE(L)
    MAP(I) = L

800 CONTINUE
IF(LISTNO.EQ.-1) RETURN

C
C ADD THE NEW ARCS --(I1,I2),(J1,J2),(K1,K2)-- TO THE
C LIST OF ARCS THAT MUST BE EXAMINED
C
LISTNO = LISTNO + 1
LIST(LISTNO,1) = K1
LIST(LISTNO,2) = K2
LISTNO = LISTNO + 1
LIST(LISTNO,1) = J1
LIST(LISTNO,2) = J2
LISTNO = LISTNO + 1
LIST(LISTNO,1) = I1

110
LIST(LISTNO, 2) = I2
RETURN
END

SUBROUTINE OROPT
C
C ************************************************************ OROPT ************************************************************
C
C THIS ROUTINE BY OR IS A MODIFICATION OF LIN'S LAMDA-OPT
C IT IS A SIMPLIFICATION WHICH IS RESTRICTED TO ONLY
C CERTAIN SWAPS
C
C INCR IS THE NUMBER OF NODES THAT WILL BE SWAPPED AT ANY ITERATION
C FIRST 3 THEN 2 THEN 1
C
C *** INSERTION COMMON BLOCK ***
C
INTEGER*4 MAXPTS, BIGINT
PARAMETER (MAXPTS = 101, BIGINT = 999999)
COMMON /POINTS/ X(MAXPTS), Y(MAXPTS), NUMPTS, MATSIZ
COMMON /DISTS/ D(MAXPTS, MAXPTS), DM(MAXPTS, MAXPTS)
COMMON /LISTS/ IROUTE(MAXPTS), ILIST(MAXPTS, 3)
COMMON /OUTCOM/ FUNIT
COMMON/PRNSW/ KPRINT(IO)
INTEGER*4 X, Y, NUMPTS, MATSIZ
INTEGER*4 D, DM
INTEGER*4 IROUTE, ILIST
INTEGER*4 FUNIT
C
C *** END OF BLOCK ***
C
COMMON /_LISTTT/ LIST(MAXPTS, 2), LISTNO, ISAVD(MAXPTS), NEW
INTEGER*4 LIST, LISTNO, ISAVD, NEW
C
INTEGER*4 KSW, INCR, NM, I, I1, I2, J, J1, J2, JP, K1, K2,
+ IDIST1, IDIST2, ID
C
C *** BEGIN ***
C
IF(KPRINT(2) .EQ. 1) WRITE(1,*), ' OROPT'
NEW = NUMPTS
LISTNO = -1
80 INCR = 3
90 NM = NUMPTS - INCR - 1
KSW = 3
DO 200 I = 1,NMPTS
  I1 = IROUTE(I)
  I2 = ILIST(I1,2)
  J = I + INCR
  IF (J.GT.NMPTS) J = J - NMPTS
  J1 = IROUTE(J)
  J2 = ILIST(J1,2)
  JP = J + 1
  IDIST1 = DM(I1,I2) + DM(J1,J2) - DM(I1,J2)
  K1 = J1
  DO 100 K = 1,NM
    K1 = ILIST(K1,2)
    K2 = ILIST(K1,2)
    IDIST2 = DM(K1,K2) - DM(K1,J1) - DM(I2,K2)
    IDIST3 = DM(K1,K2) - DM(K1,I2) - DM(J1,K2)
    ID2 = IDIST1 + IDIST2
    ID3 = IDIST1 + IDIST3
    IF(ID2 .LE. 0 .AND. ID3 .LE. 0) GO TO 100
    IF(ID3 .GT. ID2) GO TO 400
  GO TO 300
100 CONTINUE
200 CONTINUE
  GO TO 500
C
C MAKE THE SWAP AND UPDATE THE TOUR
C
300 CALL REARR(I1,J2,K1,J1,I2,K2,KSW)
  GO TO 80
400 KSW = 2
  CALL REARR(I1,J2,K1,I2,J1,K2,KSW)
  GO TO 80
500 INCR = INCR - 1
  IF (INCR.GT.0) GO TO 90
  RETURN
END
SUBROUTINE THREE

C**************************************************** THREEO ****************************************************

C THIS SUBROUTINE PERFORMS A DYNAMIC 3-OPT AS DESCRIBED BY STEIGLITZ AND WEINER
AN INITIAL SOLUTION MUST BE GIVEN IN 'IROUTE' ARRAY

C

C VARIABLES:
IROUTE(K) -- POSITION ON ROUTE OF NODE K
D(I,J) -- DISTANCE FROM NODE I TO NODE J
ILIST(K,1) -- PREDECESSOR OF NODE K
(K,2) -- SUCCESSOR OF NODE K
NUMPTS -- NUMBER OF NODES
NEW -- NUMBER OF THE NODE BEING ADDED

C *** INSERTION COMMON BLOCK ***

INTEGER*4 MAXPTS, BIGINT
PARAMETER (MAXPTS = 101, BIGINT = 999999)
COMMON /POINTS/ X(MAXPTS), Y(MAXPTS), NUMPTS, MATSIZ
COMMON /DISTS/ D(MAXPTS,MAXPTS), DM(MAXPTS,MAXPTS)
COMMON /LISTS/ IROUTE(MAXPTS), ILIST(MAXPTS,3)
COMMON /OUTCOM/ FUNIT
COMMON/PRNSW/ KPRINT(10)
INTEGER*4 X, Y, NUMPTS, MATSIZ
INTEGER*4 D, DM
INTEGER*4 IROUTE, ILIST
INTEGER*4 FUNIT

C *** END OF BLOCK ***

C STORAGE FOR ARC LISTS FOR USE BY "THREE"

COMMON /LISTTT/ LIST(MAXPTS,2), LISTNO, ISAVD(MAXPTS), NEW
INTEGER*4 LIST, LISTNO, ISAVD, NEW

C STORAGE OF LATEST SST ARCS

COMMON /FRSST/ ISIN(MAXPTS), XXX(MAXPTS), YYY(MAXPTS)
INTEGER*4 XXX, YYY
LOGICAL ISIN

113
c
C DISTANCE VARIABLES (KSW IS TYPE OF SWAP).
C
   INTEGER*4 SAVMAX, DIST, DIST3, DIFF, KSW
C
C *** BEGIN ***
C
c  IF(KPRINT(2) .EQ. 1) WRITE(1,*) '    THREE'
   LISTNO = 0
   KSW = 5
   SAVMAX = 0
C
C MAKE A 3 ARC ROUTE USING THE FIRST THREE "IROUTE" POINTS
C
   I     = IROUTE(3)
   DO 90 L = 1,3
      J     = IROUTE(L)
      ILIST(J,1) = I
      ILIST(I,2) = J
      I     = J
90 CONTINUE
C
C SUCCESSIVELY ADD ONE MORE NODE TO THE ROUTE
C
   DO 2000 NEW = 4,NUMPTS
      NEWM = NEW-1
C
C THIS SUBROUTINE CHOSSES THE LEAST EXPENSIVE PLACE TO INSERT NODE
C IROUTE(NEW) IN THE EXISTING ROUTE OF NEW-1 NODES
C
   CALL CPINST
C
C WE NEED TO EXAMINE EACH ARC IN THE LIST OF
C ARCS THAT HAVE BEEN CHANGED
C INITIALLY THIS LIST 'LIST(LISTNO,-)' WILL BE
C COMPOSED OF THE TWO ARCS ADDED BY CPINST.
C HOWEVER, IF ANY ARCS HAVE BEEN CHANGED BY
C THE 3-OPT, THESE ARCS HAVE BEEN ADDED TO THIS
C LIST IN SUBROUTINE REARR.
C
   100 DO 200 L = 1,NEW

114
IF (IROUTE(L) .NE. LIST(LISTNO,1)) GO TO 150
I1 = LIST(LISTNO,1)
IF (ILIST(I1,2).NE.LIST(LISTNO,2)) GO TO 150
I2 = LIST(LISTNO,2)
LISTNO = LISTNO - 1
GO TO 300
150 IF (IROUTE(L) .NE. LIST(LISTNO,2)) GO TO 200
I1 = LIST(LISTNO,2)
IF (ILIST(I1,2).NE.LIST(LISTNO,1)) GO TO 200
I2 = LIST(LISTNO,1)
LISTNO = LISTNO - 1
GO TO 300
200 CONTINUE

C IF THE PROGRAM CAN'T GET PAST HERE, THERE ARE SERIOUS ERRORS IN
C THE "LIST", "IROUTE", AND/OR "ILIST" ARRAYS.
C
WRITE(FUNIT,55)
WRITE(FUNIT,56) L, IROUTE(L), ILIST(I1,2)
WRITE(*,56) L, IROUTE(L), ILIST(I1,2)
FORMAT(3X,' *** ERRORS FOUND IN LISTS')
FORMAT(' ',56) L, I3,' IROUTE(L) ',I3,' NEXT NODE ',I3)
STOP

C NO SERIOUS ERRORS -- CONTINUE. GET THREE ARCS NAMED
C "I1-I2", "J1-J2", AND "K1-K2".
C
300 J1 = I1
DO 1000 J = 2,NEWM
JP = J + 1
J1 = ILIST(J1,2)
J2 = ILIST(J1,2)
DIST = D(I1,I2) + D(J1,J2)
K1 = J1
DO 800 K = JP,NEW
KP = K + 1
K1 = ILIST(K1,2)
K2 = ILIST(K1,2)
DIST3 = DIST + D(K1,K2)

C TEST THE ARRANGEMENT "I1-J1" "I2-K1" "J2-K2"

C
DIFF = DIST3 - (D(I1,J1)+D(I2,K1)+D(J2,K2))
IF (DIFF.LE.SAVMAX) GO TO 400
SAVMAX = DIFF
KSW = 1

C TEST THE ARRANGEMENT "I1-J2" "K1-I2" "J1-K2"

C
400 DIFF = DIST3 - (D(I1,J2)+D(K1,I2)+D(J1,K2))
IF (DIFF.LE.SAVMAX) GO TO 500
SAVMAX = DIFF
KSW = 2

C TEST THE ARRANGEMENT "I1-J2" "K1-J1" "I2-K2"

C
500 DIFF = DIST3 - (D(I1,J2)+D(K1,J1)+D(I2,K2))
IF (DIFF.LE.SAVMAX) GO TO 600
SAVMAX = DIFF
KSW = 3

C TEST THE ARRANGEMENT "I1-K1" "J2-I2" "J1-K2"

C
600 DIFF = DIST3 - (D(I1,K1)+D(J2,I2)+D(J1,K2))
IF (DIFF.LE.SAVMAX) GO TO 700
SAVMAX = DIFF
KSW = 4

C REMOVE ARCS --(I1,I2),(J1,J2),(K1,K2)-- FROM LIST

C
700 IF (KSW.EQ.5) GO TO 709
IF (LISTNO.EQ.0) GO TO 709
NSW = 1
N1 = I1
N2 = I2

701 DO 704 L = 1, LISTNO
LP = L - LADD
IF (LIST(L,1).EQ.N1.AND.List(L,2).EQ.N2) GO TO 702
IF (LIST(L,2).EQ.N1.AND.List(L,1).EQ.N2) GO TO 702
LIST(LP,1) = LIST(L,1)
LIST(LP,2) = LIST(L,2)
GO TO 704
702 CONTINUE
LADD = 1
704 CONTINUE
LISTNO = LISTNO - LADD
GO TO (706,707,709), NSW
706 NSW = 2
N1 = J1
N2 = J2
GO TO 701
707 NSW = 3
N1 = K1
N2 = K2
GO TO 701
709 GO TO (710,720,730,740,800), KSW
C
C SUBROUTINE REARR MAKES THE ARC SWAPS
C
710 CALL REARR(I1,J1,I2,K1,J2,K2,KSW)
GO TO 750
720 CALL REARR(I1,J2,K1,I2,J1,K2,KSW)
GO TO 750
730 CALL REARR(I1,J2,K1,J1,I2,K2,KSW)
GO TO 750
740 CALL REARR(I1,K1,J2,I2,J1,K2,KSW)
750 SAVMAX = 0
KSW = 5
GO TO 100
800 CONTINUE
1000 CONTINUE
IF (LISTNO.GT.0) THEN
C
C IF THE "LIST" ARRAY IS NOT EMPTY, REDO THIS ITERATION
C
GO TO 100
ENDIF
2000 CONTINUE
C
1 FORMAT(10I4)
C
RETURN
SUBROUTINE THROPT

C **************************** THROPT ************************ ****
C
C *** INSERTION COMMON BLOCK ***
C
INTEGER+4 MAXPTS, BIGINT
PARAMETER (MAXPTS = 101, BIGINT = 999999)
COMMON /POINTS/ X(MAXPTS), Y(MAXPTS), NUMPTS, MATSIZ
COMMON /DISTS / D(MAXPTS,MAXPTS), DM(MAXPTS,MAXPTS)
COMMON /LISTS / IROUTE(MAXPTS), ILIST(MAXPTS,3)
COMMON /OUTCOM/ FUNIT
COMMON/PRNSW/ KPRINT(IO)
INTEGER+4 X, Y, NUMPTS, MATSIZ
INTEGER+4 D, DM
INTEGER+4 IROUTE, ILIST
INTEGER+4 FUNIT

C
C *** END OF BLOCK ***
C
C STORAGE FOR ARC LISTS FOR USE BY "THREE"
C
COMMON /LISTTT/ LIST(MAXPTS,2), LISTNO, ISAVD(MAXPTS), NEW
INTEGER+4 LIST, LISTNO, ISAVD, NEW

C
INTEGER+4 SAVMAX, DIST, DIST3, DIFF

C
C *** BEGIN ***
C
IF(KPRINT(2) .EQ. 1) WRITE(1,*), ' THROPT'
NM = NUMPTS - 1
KSW = 5
SAVMAX = 0
DO 90 I = 1, NUMPTS
   II = IROUTE(I)
   JJ = ILIST(II,2)
   LIST(I,1) = II
   LIST(I,2) = JJ
90 CONTINUE
LISTNO = NUMPTS
NEW   = NUMPTS

100 DO 200 L = 1,NUMPTS
   IF (IROUTE(L).NE.LIST(LISTNO,1)) GO TO 150
      I1   = LIST(LISTNO,1)
   IF (ILIST(I1,2).NE.LIST(LISTNO,2)) GO TO 150
      I2   = LIST(LISTNO,2)
      LISTNO = LISTNO - 1
      GO TO 300
150 IF (IROUTE(L).NE.LIST(LISTNO,2)) GO TO 200
      I1   = LIST(LISTNO,2)
   IF (ILIST(I1,2).NE.LIST(LISTNO,1)) GO TO 200
      I2   = LIST(LISTNO,1)
      LISTNO = LISTNO - 1
      GO TO 300
200 CONTINUE

C IF THE PROGRAM CAN'T GET PAST HERE, THERE ARE SERIOUS ERRORS IN
C THE "LIST", "IROUTE", AND/OR "ILIST" ARRAYS.
C
WRITE(FUNIT,55)
WRITE(FUNIT,56) L, IROUTE(L), ILIST(I1,2)
WRITE( * ,55)
WRITE( * ,56) L, IROUTE(L), ILIST(I1,2)
55 FORMAT(3X,' *** ERRORS FOUND IN LISTS')
56 FORMAT( 'L ','I3,' IROUTE(L) 'I3,' NEXT NODE 'I3)
STOP

C NO SERIOUS ERRORS -- CONTINUE. GET THREE ARCS NAMED
C "I1-I2", "J1-J2", AND "K1-K2".
C
300 J1   = I1
   DO 1000 J = 2,NM
      JP   = J + 1
      J1   = ILIST(J1,2)
      J2   = ILIST(J1,2)
      DIST = DM(I1,I2) + DM(J1,J2)
      K1   = J1
   DO 800 K = JP,NUMPTS
      KP   = K + 1
      K1   = ILIST(K1,2)
   800 CONTINUE
      K1   = ILIST(K1,2)
K2 = ILIST(K1,2)
DIST3 = DIST + DM(K1,K2)
DIFF = DIST3-(DM(I1,J1)+DM(I2,K1)+DM(J2,K2))
IF (DIFF.LE.SAVMAX) GO TO 400
SAVMAX = DIFF
KSW = 1

400 DIFF = DIST3-(DM(I1,J2)+DM(K1,I2)+DM(J1,K2))
IF (DIFF.LE.SAVMAX) GO TO 500
SAVMAX = DIFF
KSW = 2

500 DIFF = DIST3-(DM(I1,J2)+DM(K1,J1)+DM(I2,K2))
IF (DIFF.LE.SAVMAX) GO TO 600
SAVMAX = DIFF
KSW = 3

600 DIFF = DIST3-(DM(I1,K1)+DM(J2,I2)+DM(J1,K2))
IF (DIFF.LE.SAVMAX) GO TO 700
SAVMAX = DIFF
KSW = 4

C C REMOVE ARCS --(I1,I2),(J1,J2),(K1,K2)-- FROM LIST
C

700 IF (KSW.EQ.5) GO TO 709
IF (LISTNO.EQ.0) GO TO 709
NSW = 1
N1 = I1
N2 = I2

701 LADD = 0
DO 704 L = 1,LISTNO
   LP = L - LADD
   IF (LIST(L,1).EQ.N1.AND.LIST(L,2).EQ.N2) GO TO 702
   IF (LIST(L,2).EQ.N1.AND.LIST(L,1).EQ.N2) GO TO 702
   LIST(LP,1) = LIST(L,1)
   LIST(LP,2) = LIST(L,2)
   GO TO 704

702 LADD = 1
704 CONTINUE
LISTNO = LISTNO - LADD
GO TO (706,707,709), NSW

706 NSW = 2
N1 = J1
N2 = J2
GO TO 701
707 NSW = 3
N1 = K1
N2 = K2
GO TO 701
709 GO TO (710,720,730,740,800),KSW
C
C SUBROUTINE REARR MAKES THE ARC SWAPS
C
710 CALL REARR(I1,J1,I2,K1,J2,K2,KSW)
GO TO 750
720 CALL REARR(I1,J2,K1,I2,J1,K2,KSW)
GO TO 750
730 CALL REARR(I1,J2,K1,J1,I2,K2,KSW)
GO TO 750
740 CALL REARR(I1,K1,J2,I2,J1,K2,KSW)
750 SAVMAX = 0
KSW = 5
GO TO 100
800 CONTINUE
1000 CONTINUE
IF(LISTNO.GT.0) GO TO 100
RETURN
END

SUBROUTINE CPINST
C*****************************************************************************************
C*****************************************************************************************
C
C THIS ROUTINE CHOOSES WHERE TO INSERT THE NEWTH NODE IN THE EXISTING ROUTE. IT REORDERS THE ROUTE WITH THIS NODE IN POSITION 2
C
C *** INSERTION COMMON BLOCK ***
C
C INTEGER*4 MAXPTS, BIGINT
PARAMETER (MAXPTS = 101, BIGINT = 999999)
COMMON /POINTS/ X(MAXPTS), Y(MAXPTS), NUMPTS, MATSIZ
COMMON /DISTS/ D(MAXPTS,MAXPTS), DM(MAXPTS,MAXPTS)
COMMON /LISTS/ IROUTE(MAXPTS), ILIST(MAXPTS,3)
COMMON /OUTCOM/ FUNIT
COMMON/PRNSW/ KPRINT(10)
INTEGER*4 X, Y, NUMPTS, MATSIZ
INTEGER*4 D, DM
INTEGER*4 IROUTE, ILIST
INTEGER*4 FUNIT

C
C *** END OF BLOCK ***
C
C STORAGE FOR LIST ARRAYS FROM "THREEO"
C
COMMON /LISTTT/ LIST(MAXPTS,2), LISTNO, ISAVD(MAXPTS), NEW
INTEGER*4 LIST, LISTNO, ISAVD, NEW
C
C DISTANCE AND ENDPOINT VARIABLES
C
INTEGER*4 K, ID, IDIST
C
C *** BEGIN ***
C
C FIND CHEAPEST PLACE TO INSERT NODE K IN THE EXISTING ROUTE
C
K = IROUTE(NEW)
IDIST = BIGINT * 2
NEWM = NEW - 1
I = IROUTE(NEWM)
DO 100 L = 1, NEWM
   ISAVD(L) = IROUTE(L)
   J = IROUTE(L)
   ID = (DM(I,K) + DM(J,K)) - DM(I,J)
C
C IF THE DISTANCE DIFFERENCE IS NOT BETTER THAN THE PREVIOUS BEST,
C SKIP AND START ANOTHER TEST.
C
   IF (ID.GE.IDIST) GO TO 90
LSAV = L
   ILIST(K,1) = I
   ILIST(K,2) = J
   IDIST = ID
90   I = J
100 CONTINUE
C
C INSERT K AFTER IROUTE(LSAV-1)=I AND MAKE NODE I THE FIRST NODE ON THE ROUTE.

C

I = ILIST(K,1)
J = ILIST(K,2)
ILIST(I,2) = K
ILIST(J,1) = K
IRoute(1) = I
IRoute(2) = K
DO 200 L = 3,NEWM
   LP = LSAV + L - 3
   IROUTE(L) = ISAVD(LP)
   IF (LP.EQ.NEWM) LSAV = LSAV-NEWM
200 CONTINUE
LIST(1,1) = I
LIST(1,2) = K
LIST(2,1) = K
LIST(2,2) = J
LISTNO = 2
RETURN
END

SUBROUTINE REARR(I1, I2, J1, J2, K1, K2, KSW)
C****************************** REARR ************************** ****
C
C THIS ROUTINE REARRANGES THE SEQUENCE OF NODES IN IROUTE
C
C *** INSERTION COMMON BLOCK ***
C
INTEGER*4 MAXPTS, BIGINT
PARAMETER (MAXPTS = 101, BIGINT = 999999)
COMMON /POINTS/ X(MAXPTS), Y(MAXPTS), NUMPTS, MATSIZ
COMMON /DIST2/ D(MAXPTS,MAXPTS), DM(MAXPTS,MAXPTS)
COMMON /LISTS / IROUTE(MAXPTS), ILIST(MAXPTS,3)
COMMON /OUTCOM/ FUNIT
COMMON/PRNSW/ KPRINT(10)
INTEGER*4 X, Y, NUMPTS, MATSIZ
INTEGER*4 D, DM
INTEGER*4 IROUTE, ILIST
INTEGER*4 FUNIT
C
C *** END OF BLOCK ***
C
C STORAGE FOR LIST ARRAYS FROM "THREEO"
C
COMMON /LISTTT/ LIST(MAXPTS,2), LISTNO, ISAVD(MAXPTS), NEW
    INTEGER*4 LIST, LISTNO, ISAVD, NEW
C
C *** BEGIN ***
C
ILIST(I1,2) = I2
IF ((KSW.EQ.2).OR.(KSW.EQ.3)) GO TO 200
C
C REORDER ROUTE FROM I2 TO J1
C
   I       = I2
   ILSAV   = ILIST(I,1)
   ILIST(I,1) = I1
   GO TO 160
150  ILSAV   = ILIST(I,1)
   ILIST(I,1) = ILIST(I,2)
160  ILIST(I,2) = ILSAV
   IF (I.EQ.J1) GO TO 200
   I       = ILSAV
   GO TO 150
200  ILIST(J1,2) = J2
   ILIST(J2,1) = J1
   IF ((KSW.EQ.2).OR.(KSW.EQ.4)) GO TO 400
C
C REORDER ROUTE FROM J2 TO K1
C
   J       = J2
   JLSAV   = ILIST(J,1)
   ILIST(J,1) = J1
   GO TO 360
350  JLSAV   = ILIST(J,1)
   ILIST(J,1) = ILIST(J,2)
360  ILIST(J,2) = JLSAV
   IF (J.EQ.K1) GO TO 400
   J       = JLSAV
   GO TO 350

124
400  ILIST(K1,2) = K2
     ILIST(J2,1) = J1
     ILIST(K2,1) = K1
     I     = IROUTE(1)
     DO 800 L = 2,NEW
           IROUTE(L) = ILIST(I,2)
           I     = IROUTE(L)
800  CONTINUE
     IF (LISTNO.EQ.-1) RETURN

C
C ADD THE NEW ARCS --(I1,I2), (J1,J2), (K1,K2)-- TO THE
C LIST OF ARCS THAT MUST BE EXAMINED
C
     LISTNO       = LISTNO + 1
     LIST(LISTNO,1) = K1
     LIST(LISTNO,2) = K2
     LISTNO       = LISTNO + 1
     LIST(LISTNO,1) = J1
     LIST(LISTNO,2) = J2
     LISTNO       = LISTNO + 1
     LIST(LISTNO,1) = I1
     LIST(LISTNO,2) = I2
     RETURN
END
Appendix B - Best Parallel Code
Program His.f (Divide and Prune - ordered min to max)

Ordered min to max

This is the Master.

The Message Types are as follows: 1) send us each a route
5) I am exiting

include 'fpvm3.h'
IMPLICIT NONE
Integer NPROC
INTEGER MAXPTS, Maxrws
Parameter(NPROC = 4)
PARAMETER(MAXPTS = 51)
Parameter(Maxrws = 92200)
COMMON/MATRIX/DIST,NUMVTX,LOWERB
INTEGER DIST(MAXPTS,MAXPTS),NUMVTX,CITIES,LOWERB
INTEGER DISTANCE(MAXPTS,MAXPTS)
INTEGER X(MAXPTS),Y(MAXPTS),NV,STOPTIME,TIMELIMIT
INTEGER N,MINW,MAXW,T,I,J,K,W(MAXPTS),L
INTEGER COUNT,CARDINALITY,ORIG_LABEL(MAXPTS),0
REAL DUAL(MAXPTS),DUALT,Z,GAP

Real tocost
Integer totalspawned, iter, r100000, i100, sechalf
Integer split, newrtnum
Integer info, numstarted, ijk, m
Integer mytid, child
Integer tids(NPROC), DistParam
integer indual(MAXPTS),indualt
integer totalcount, tempcount, prevtotal
integer Power, temparray(Maxpts + 2)

C

integer BigMatrix(Maxrws, Maxpts + 2)
integer PvmDataRaw, oldval, rtid, rtag, rlen
C
integer PvmRoute, PvmRouteDirect
real temp, Childtimes(NPROC), MaxChildtime

integer time1, time2, mclock
external function mclock
real totseconds
Power = INT((log(NPROC *1.0)/log(2.0)) + 0.1)

COUNT=0
CARDINALITY=0

$OPEN(15,FILE='/u/user10/wolf/VRP/HisWithPrune/ENUM_29-3',
$     STATUS = 'UNKNOWN')
$READ(15,*) N,MAXW,NV,STOPTIME,TIMELIMIT,GAP
T=0
C   print *,'starting to run enumerate > enumerate.out'
Cities = N-1
C   Maxlength = Cities + 2
DistParam = Maxpts * Maxpts
C   Read in City Coordinates, Weight Requirements, dual vars
DO I=1,N
   ORIG_LABEL(I)=I
   READ(15,*) X(i),Y(i), W(I)
   PRINT *,X(i),Y(i),W(I)
   T=T+W(I)
ENDDO
DO I=1,N
   read(15,*)dual(i)
   dual(i)=dual(i)*1000.
   indual(i)=int(dual(i))
endo
N=N-1
C   Compute Minimum Weight (in attempt to evenly distribute
C     the deliveries by truck)
MINW=T-(NV-1)*MAXW
MINW=MAX(0,MINW)
c   read(15,*)minw
PRINT *,MINW,MAXW
C   Dual total = Dual of the depot
C   indual total = indual of the depot
DUALT = DUAL(N+1)
indualt = indual(n+1)

C Reorder cities by weight requirement (min to max)

L = 1
DO WHILE (L .NE. N)
   IF (W(L) .GT. W(L+1)) THEN
      O = ORIG_LABEL(L)
      I = W(L)
      J = X(L)
      K = Y(L)
      Z = DUAL(L)
      ijk = indual(L)
      ORIG_LABEL(L) = ORIG_LABEL(L+1)
      W(L) = W(L+1)
      X(L) = X(L+1)
      Y(L) = Y(L+1)
      DUAL(L) = DUAL(L+1)
      indual(L) = indual(L+1)
      ORIG_LABEL(L+1) = 0
      W(L+1) = I
      X(L+1) = J
      Y(L+1) = K
      DUAL(L+1) = Z
      indual(L+1) = ijk
   ELSE
      L = L + 1
   END IF
END DO

PRINT *, (W(I), I = 1, N)
PRINT *, (dual(i), i = 1, n)

C Set up distance array and print out distances

DO I = 1, N+1
   distance(i, i) = 999999999
END DO
dist(i,i)=99999999
do j=i+1,n+1
  distance(i,j)=int(1000*(((x(i)-x(j))**2+(y(i)-y(j))**2)**.5))
  dist(i,j)=distance(i,j)
  distance(j,i)=distance(i,j)
  dist(j,i)=distance(i,j)
  print *, 'distance of coordinates ',i,j, ' is ',dist(i,j)
enddo
enddo
numvtx=n+1

Time1 = mclock()

C Find initial lower bounds

  call rreduce
  print *, 'MAM = ', lowerb
  call onetree
  print *, '1-TREE = ', lowerb
  call HEURISTIC_4(numvtx,dist,lowerb)
  print *, 'TSP = ', lowerb

  Call pvmfmytid(mytid)

  Call pvmfsetopt(PvmRoute, PvmRouteDirect, oldval)

  Call pvmfcatchout(1, info)

C Spawn the Worker Tasks
C TotalSpawned = 0
C Do while (TotalSpawned.LT.NPROC)
C   Call pvmfspawn('his2',0, '*', NPROC-TotalSpawned,
C &   tids(TotalSpawned +1), numstarted)
C   Call pvmfspawn('his2',0, '*', NPROC, tids, numstarted)
C   TotalSpawned = TotalSpawned + numstarted
   Print *, 'I have spawned ', numstarted, ' workers'
C   Print *, ' The tids are ',(tids(i), i = TotalSpawned+1,
C &   TotalSpawned+numstarted)
C ENDDO

Call pvmfinitsend(PVMDatagram, info)
Call pvmfpack(INTEGER4, Cities, 1, 1, info)
Call pvmfpack(INTEGER4, maxW, 1, 1, info)
Call pvmfpack(INTEGER4, NV, 1, 1, info)
Call pvmfpack(INTEGER4, StopTime, 1, 1, info)
Call pvmfpack(INTEGER4, TimeLimit, 1, 1, info)
Call pvmfpack(INTEGER4, Gap, 1, 1, info)
Call pvmfpack(INTEGER4, minW, 1, 1, info)
Call pvmfpack(REAL4, dualt, 1, 1, info)
Call pvmfpack(INTEGER4, indualt, 1, 1, info)
Call pvmfpack(INTEGER4, lowerb, 1, 1, info)
Call pvmfpack(INTEGER4, Power, 1, 1, info)
Call pvmfmcast(numstarted, tids, 8, info)

Do i = 1, NPROC
   Call pvmfinitsend(PVMDataRaw, info)
   Call pvmfpsend(tids(i), 9, X, Cities, INTEGER4, info)
ENDDO

Do i = 1, NPROC
   Call pvmfinitsend(PVMDataRaw, info)
   Call pvmfpsend(tids(i), 19, Y, Cities, INTEGER4, info)
ENDDO

Do i = 1, NPROC
   Call pvmfinitsend(PVMDataRaw, info)
   Call pvmfpsend(tids(i), 29, W, Cities, INTEGER4, info)
ENDDO

Do i = 1, NPROC
   Call pvmfinitsend(PVMDataRaw, info)
   Call pvmfpsend(tids(i), 39, dual, Cities, REAL4, info)
ENDDO

Do i = 1, NPROC
   Call pvmfinitsend(PVMDataRaw, info)
   Call pvmfpsend(tids(i), 49, indual, Cities, INTEGER4, info)
ENDDO

Call pvmfinitsend(PVMDataRaw, info)
Do i = 1, NPROC
    child = tids(i)
    Do j = 1, Cities + 1
        Call pvmfinitsend(PVMDaraRaw, info)
        Call pvmfpsend(child, 111, dist(1,j),
                        Cities + 1, REAL4, info)
    ENDDO
ENDDO

Do i = 1, NPROC
    child = tids(i)
    Do j = 1, Cities + 1
        Call pvmfinitsend(PVMDaraRaw, info)
        Call pvmfpsend(child, 111, distance(1,j),
                        Cities + 1, REAL4, info)
    ENDDO
ENDDO

Do j = 1, NPROC

    C   Find the New Rt number to be sent, and send it
    C   for each of the 1 to NPROC tasks

        Call pvmfinitsend(PVMDaraRaw, info)
        Call pvmfpsend(tids(j), 2, j-1, 1, INTEGER4, info)
        Print *, 'Tag number ', j-1, ' to ', tids(j)
    ENDQ

    C   Wait for output messages from the children,
    C   in order.

        Totalcount = 0
        Print *, 'Total count is ', Totalcount

    DO j = 1, NPROC
        Call pvmfinitsend(PvmDataRaw, info)
        Call pvmfpsend(tids(j), 22, j, 1, INTEGER4, info)
        Print *, 'I am waiting for messages from task ', j
        Call pvmfrecv(tids(j), 7, tempcount, 1, INTEGER4,
                        & rtid, rtag, rlen, info)
        Print *, 'I received a tempcount of ', tempcount, 'from ', tids(j)
    ENDDO
If (tempcount.GT.0) Then
C Print *, 'Count from task ', j, ' is ', tempcount
Prevtotal = totalcount
C Print *, 'Previous total is ', tempcount
Totalcount = totalcount + tempcount
C Print *, 'New total is ', tempcount
Do k = Prevtotal + 1, Totalcount
    Print *, 'k is ', k
    Call pvmfrecv(tids(j), 6, info)
    Do m = 1, Cities + 2
        Call pvmfunpack(INTEGER4, temparray(m), 1, 1, info)
    ENDDO
C Print *, K, ',(temparray(m), m=1,Cities +2)
ENDDO
ENDIF
Call pvmfinitsend(PvmDataRaw, info)
Call pvmfpsend(tids(j), 4, j, 1, INTEGER4, info)
C Print *, 'I have sent end message to task ', j
Call pvmfrecv(tids(j), 3, temp, 1, Real4, &
    rtid, rtag, rlen, info)
Childtimes(j) = temp
C Print *, 'I am sending end message to task ', j
Call pvmfinitsend(PvmDataRaw, info)
    Call pvmfpsend(tids(j), 44, j, 1, INTEGER4, info)
C Print *, 'I have sent end message to task ', j
ENDDO
Print *, 'Total count is ', Totalcount
Time2 = mclock()
Maxchildtime = Childtimes(1)
Do i = 2, NPROC
    If (Childtimes(i).GT.Maxchildtime) Maxchildtime =
&    Childtimes(i)
ENDDO
C OPEN(16, FILE='H_EOUT', STATUS = 'UNKNOWN')
C OPEN(17, FILE='H_cols', STATUS = 'UNKNOWN')
C
C write(17,71)
C71 format('NAME',10x,'SETPARLP')
C write(17,72)
C72 format('ROWS')
C write(17,73)
C73 format(1x,'N',2x,'COST')
C do i=1,Cities
C    il00=i+100
C    write(17,74)il00
C74 format(1x,'E',2x,'NOD',i3)
C enddo
C write(17,755)
C755 format(1x,'E',2x,'VEHI')
C write(17,77)
C77 format('COLUMNS')
C
C iter = 1
C Do K = totalcount, 1, -1
C NewRtNum = 0
C If (Cities.LE.30) Then
C    Split = Cities
C Else
C    Split = Cities / 2
C ENDIF
C Do i = 1, Split - 1
C    NewRtNum = 2 * (NewRtNum + Bigmatrix(K, i+2))
C ENDDO
C NewRtNum = NewRtNum + Bigmatrix(K, split+2)
C SecHalf = 0
C If (Split.NE.Cities) Then
C    Do i = Split + 1, Cities -1
C    NewRtNum = 2 * (NewRtNum + Bigmatrix(K, i+2))
C ENDDO
C NewRtNum = NewRtNum + Bigmatrix(K, cities+2)
C ENDIF
C WRITE(16,*)Iter,BigMatrix(K,1),BigMatrix(K,2),
C & NewRtNum, '-', SecHalf
C
tocost = (BigMatrix(K,2))/1000.0
C r100000=Iter+100000
C write(17,119) r100000,tocost
C119 format(4x,'R',i6,3x,'COST',6x,f8.2)
C do p=3,Cities+2
C if(BigMatrix(K,p).eq.1) then
C i100=orig_label(p-2)+100
C write(17,44)rl00000,i100
C44 format(4x,'R',i6,3x,'NOD',i3,4x,'1.0')
C end if
C enddo
C write(17,157)rl00000
C157 format(4x,'R',i6,3x,'VEHI',6x,'1.0')
C iter = iter + 1
C ENDDO
C
C WRITE(16,*)TotalCOUNT
C write(17,101)
C101 format('RHS')
C do i=1,Cities
C i100=i+100
C write(17,102)i100
C102 format(4x,'RHS',7x,'NOD',i3,4x,'1.0')
C enddo
C write(17,115)float(nv)
C115 format('BOUNDS')
C do i=1,Totalcount
C write(17,107)100000+i
C107 format(1x,'BV',1x,'BOUNDS',4x,'R',i6)
C enddo
C write(17,104)
C104 format('ENDATA')
C
C Print *, 'I have finished printing routes.'

DO j = 1, NPROC
Print *, 'Waiting for exit message ', j
Call pvmfrecv(-1, 5, info)
ENDDO

C Then, exit

Print *, 'I am Exiting'
Call pvmfexit(info)
totseconds = real(time2 - time1)/100.0
print *, totseconds, ' Was the time for the master with ',
& Cities,' cities and ',nv, ' vehicles using the Pruning Code.'
print *, maxchildtime, ' Was the max time for a child with ',
& Cities,' cities and ',nv, ' vehicles using the Pruning Code.'
STOP
END

C***********************************************************************
C
C THIS IS A PROTOTYPE CODE (OCT.1989) WRITTEN AND DESIGNED BY
C CHARLES E. NOON (UNIV OF TENNESSEE) AND THOMAS CHAN (S.M.U.) FOR
C PRODUCING LOWER BOUNDS ON THE LP RELAXATION OF THE PERFECT
C 2-MATCHING PROBLEM. THE APPROACH IS BASED ON "A MULTIPLIER
C ADJUSTMENT APPROACH FOR THE SET PARTITIONING PROBLEM" BY
C THOMAS CHAN AND CANDACE YANO (TECH. PAPER, DEPT. OF CSE, S.M.U.,
C DECEMBER, 1988). NO PART OF THIS CODE MAY BE COPIED OR
C DISTRIBUTED WITHOUT WRITTEN CONSENT OF THE AUTHORS.
C
C THE PROBLEM TO BE BOUNDED IS GIVEN AS,
C
C MINIMIZE CX
C
C      AX = 2
C
C      0 <= X <= 1
C
C WHERE "A" IS THE NODE EDGE INCIDENCE MATRIX FOR A GRAPH. THE
C EDGE COSTS "C" ARE TO BE INPUT AS A COMPLETE N BY N DISTANCE
C MATRIX.
C
C NOTE: THE DIMENSIONS ARE SET FOR 100 NODES.
C
C
C***********************************************************************
C
SUBROUTINE RREDUCE
C
THE FOLLOWING FOUR LINES SHOULD BE COPIED AND PLACED IN
THE PART OF YOUR CODE WHERE AN INTEGER DISTANCE MATRIX IS STORED.
THE COMPLETE SYMMETRIC DISTANCE MATRIX (ALL N SQUARED VALUES)
SHOULD BE WRITTEN INTO DIST(I,J) AND THE NUMBER OF VERTICES
SHOULD BE ASSIGNED TO NUMVTX.

INTEGER MAXPTS
PARAMETER(MAXPTS = 51)
COMMON/MATRIX/DIST,NUMVTX,LOWERB
INTEGER DIST(MAXPTS,MAXPTS),NUMVTX,LOWERB

C
C
C***********************************************************************C
C AFTER THE DISTANCE MATRIX AND NUMVTX IS LOADED INTO THE C
C COMMON BLOCK /MATRIX/, THE SUBROUTINE CAN BE CALLED BY C
C A SIMPLE "CALL RREDUCE" COMMAND. THE LOWER BOUND WILL BE GIVEN C
C AS THE "LOWERB" VALUE AND THE DUAL FEASIBLE REDUCED COSTS WILL C
C BE WRITTEN BACK INTO THE DIST(I,J) MATRIX. THEREFORE, YOU MAY C
C READ THE REDUCED COSTS OUT OF THE /MATRIX/ COMMON BLOCK AND WORK C
C ON THEM. THE INTERNAL ARRAY BESTU( ) IS THE FINAL VECTOR OF DUAL C
C VARIABLE VALUES CORRESPONDING TO THE AX=2 CONSTRAINTS. C
C***********************************************************************C

IMPLICIT NONE

 Integer NUMROW, NUMCOL

 PARAMETER(NUMROW = 900, NUMCOL = 450000)
 INTEGER C,R,CADD1,P1,PADD1,SUM,CUTOFF,NCOUNT,NROW
 INTEGER LB,NSOL,NCOL,TOTW,ITER,TOTAL,TEMP,TEMPT
 INTEGER ROWST(NUMROW),ROWEND(NUMROW),AROW(NUMCOL+NUMCOL)
 INTEGER ACOL(NUMCOL+NUMCOL),U(NUMROW),BESTU(NUMROW)
 INTEGER COST(NUMCOL),ACOST(NUMCOL),COUNT(NUMROW)
 INTEGER NCOVER(NUMROW),SOL(NUMCOL),TABUR1(NUMROW)
 LOGICAL FOUND,FOUNDA,SFLAG
 INTEGER MXITER, index, nless1,i1,i2,i1and1,m,j,minim,min
 INTEGER iadd1, iadd2, iadd, l, i, ntotu, ntotsi, nreduc
 MXITER = 10
 NROW = NUMVTX
 NCOL=NROW*(NROW-1)/2
 DO R = 1, NROW
   COUNT(R) = 0
 ENDDO

C
C THE FOLLOWING LOOP IS WHERE THE UPPER TRIANGLE OF C
C THE DISTANCE MATRIX IS READ. C

C=0
INDEX=1
NLESS1=NROW-1
DO I1=1,NLESS1
   I1AND1=I1+1

DO I2=I1AND1,NROW
    C=C+1
    COST(C)=DIST(I1,I2)
    ACOL(INDEX) = I1
    ACOL(INDEX+1) = I2
    COUNT(I1) = COUNT(I1) + 1
    COUNT(I2) = COUNT(I2) + 1
    ACOST(C) = COST(C)
    INDEX = INDEX + 2
ENDDO
ENDDO

C FURTHER INITIALIZATION.

TOTAL = 0
DO R = 1,NROW
    ROWST(R) = TOTAL + 1
    TOTAL = TOTAL + COUNT(R)
    ROWEND(R) = TOTAL
    COUNT(R) = ROWST(R) - 1
    NCOVER(R) = 0
    U(R) = 0
    TABUR1(R) = 0
ENDDO
INDEX = 0
DO C = 1,NCOL
    P1 = 2 * C
    I1 = ACOL(P1-1)
    I2 = ACOL(P1)
    COUNT(I1) = COUNT(I1) + 1
    AROW(COUNT(I1)) = C
    COUNT(I2) = COUNT(I2) + 1
    AROW(COUNT(I2)) = C
ENDDO
LB = 0
NSOL = 0
TOTAL = 0
ITER = 0

C END OF INITIALIZATION.
C BEGIN ITERATIONS.
DO WHILE (ITER.LT.MXITER)
  ITER = ITER + 1
  IF(ITER .NE. 1) THEN
    CUTOFF = ITER - 1
    M = NSOL
    FOUND = .FALSE.
    DO J = M,1,-1
      C = SOL(J)
      IF(ACOST(C) .LE. 0) THEN
        P1 = 2 * C
        I1 = ACOL(P1-1)
        I2 = ACOL(P1)
        IF(NCOVER(I2) .LT. NCOVER(I1)) THEN
          I = I1
          I1 = I2
          I2 = I
        END IF
        IF(NCOVER(I1) .EQ. 1) THEN
          FOUND = .TRUE.
          MINIM = 10**9
          R=I1
          DO L = ROWST(R),ROWEND(R)
            TEMP = ACOST(AROW(L))
            IF((TEMP .LT. MINIM) .AND.(TEMP .GT. 0)) THEN
              MINIM = TEMP
              CADD1 = AROW(L)
            END IF
          ENDDO
          U(R) = U(R) + MINIM
          LB = LB + MINIM
          DO L = ROWST(R),ROWEND(R)
            CADD1 = AROW(L)
            ACOST(CADD1) = ACOST(CADD1) - MINIM
            IF(ACOST(CADD1) .EQ. 0) THEN
              PADD1 = 2 * CADD1
              IADD1 = ACOL(PADD1-1)
              IADD2 = ACOL(PADD1)
              NCOVER(IADD1) = NCOVER(IADD1) + 1
              NCOVER(IADD2) = NCOVER(IADD2) + 1
            END IF
          SFLAG = .FALSE.
        END IF
      END IF
    ENDDO
  END IF
ENDDO
IADD = 1

DO While (IADD.LE.NSOL.AND..NOT.SFLAG)
     IF(SOL(IADD) .EQ. CADD1) SFLAG = .TRUE.
     IADD = IADD + 1
ENDDO

IF (.NOT.SFLAG) THEN
     NSOL = NSOL + 1
     SOL(NSOL) = CADD1
END IF
ENDIF

ENDDO

TABUR1(I1) = ITER

IF(((NCOVER(I2).GT.2).AND.(TABUR1(I2).LT.CUTOFF))THEN
     MIN = ACOST(C)
     R = I2
     SUM = MIN
     U(R) = U(R) + SUM
     LB = LB + SUM
     DO L = ROWST(R),ROWEND(R)
         CADD1 = AROW(L)
         IF(ACOST(CADD1) .LE. 0) THEN
             FOUNDA = .TRUE.
         ELSE
             FOUNDA = .FALSE.
         END IF
     ACOST(CADD1) = ACOST(CADD1) - SUM
     IF((ACOST(CADD1) .GT. 0) .AND.(FOUND)) THEN
         PADD1 = 2 * CADD1
         IADD1 = ACOL(PADD1-1)
         IADD2 = ACOL(PADD1)
         NCOVER(IADD1) = NCOVER(IADD1) - 1
         NCOVER(IADD2) = NCOVER(IADD2) - 1
     END IF
     ENDDO
     END IF
ENDIF

ENDIF

ENDDO

IF(.NOT.FOUND) Go to 300

NCOUNT = 0

TOTW = 0
DO I = 1,NSOL
  C = SOL(I)
  IF(ACOST(C) .LE. 0) THEN
    NCOUNT = NCOUNT + 1
    SOL(NCOUNT) = C
    TOTW = TOTW + ACOST(C)
  END IF
ENDDO
NSOL = NCOUNT
ELSE
  TOTW=0
  FOUND = .TRUE.
ENDIF
DO R = 1,NROW
  IF(NCOVER(R) .EQ. 0) THEN
    MINIM = 10**9
    DO L = ROWST(R),ROWEND(R)
      TEMP = ACOST(AROW(L))
      IF(TEMP .LT. MINIM) THEN
        MINIM = TEMP
        CADD1 = AROW(L)
      END IF
    ENDDO
    U(R) = U(R) + MINIM
    LB = LB + MINIM
    DO L = ROWST(R),ROWEND(R)
      CADD1 = AROW(L)
      ACOST(CADD1) = ACOST(CADD1) - MINIM
    END IF
  ELSE
    SFLAG=.FALSE.
    IADD = 1
    DO While (IADD.LE.NSOL.AND..NOT.SFLAG)
      IF(SOL(IADD) .EQ. CADD1) SFLAG = .TRUE.
    IADD = IADD + 1
    ENDDO
  ENDIF
IF (.NOT.SFLAG) THEN
NSOL = NSOL + 1
SOL(NSOL) = CADD1
ENDIF
END IF
ENDDO
ENDDO
IF(.NOT. FOUND) GO TO 300
TEMPT = 2 * LB + TOTW
IF(TEMPT .GT. TOTAL) THEN
  TOTAL = TEMPT
  DO R=1,NROW
    BESTU(R) = U(R)
  ENDDO
END IF
ENDDO

C
C ITERATIONS COMPLETED.
C BEGIN CALCULATION OF REDUCED COSTS AND TERMINATION.
C
300 NTOTU=0
  DO I=1,NROW
    NTOTU=NTOTU+BESTU(I)
  ENDDO
  NTOTSI=0
  DO C=1,NCOL
    P1 = 2 * C
    I1 = ACOL(P1-1)
    I2 = ACOL(P1)
    NREDUC=COST(C)-BESTU(I1)-BESTU(I2)
    IF(NREDUC.LT.0) THEN
      NTOTSI=NTOTSI+NREDUC
      NREDUC=0
    END IF
    DIST(I1,I2) = NREDUC
    DIST(I2,I1) = NREDUC
  ENDDO
  LOWERB=2*NTOTU+NTOTSI
  PRINT *, 'FINAL LOWERBOUND = ',LOWERB,' = 2 * ',ntotu,' + ',ntotsi
RETURN
END
SUBROUTINE ONETREE

IMPLICIT NONE
INTEGER MAXPTS
PARAMETER(MAXPTS = 51)
COMMON/MATRIX/DIST,NUMVTX,LOWERB
INTEGER DIST(MAXPTS,MAXPTS),NUMVTX,LOWERB

INTEGER FF(MAXPTS),UU(MAXPTS),QQ(MAXPTS)
INTEGER DEGREE(MAXPTS)
COMMON/DEG/DEGREE

Integer j,i,k,min1,min2,id

Print *, 'I am in one-tree'
J = 1
MIN1 = DIST(1,NUMVTX)
DO I = 2,NUMVTX - 1
   IF(MIN1 .GT. DIST(I,NUMVTX)) THEN
      MIN1 = DIST(I,NUMVTX)
      J = I
   END IF
END DO

IF(J .NE. 1) THEN
   K = 1
ELSE
   K = 2
END IF

MIN2 = DIST(K,NUMVTX)
DO I = K+1,NUMVTX - 1
   IF(MIN2 .GT. DIST(I,NUMVTX) .AND. I .NE. J) THEN
      MIN2 = DIST(I,NUMVTX)
      K = I
   END IF
END DO
NUMVTX = NUMVTX - 1

! extracting degree
! of each node

TIME0 = CTIME()
CALL PRIM(qq,UU,FF)
DO I = 1,NUMVTX
   DEGREE(I) = 0
END DO
DO I = 2,NUMVTX
   DEGREE(FF(I)) = DEGREE(FF(I)) + 1
   DEGREE(I) = DEGREE(I) + 1
END DO
DEGREE(J) = DEGREE(J) + 1
DEGREE(K) = DEGREE(K) + 1
DEGREE(NUMVTX+1) = 2
TIME1 = CTIME() - TIME0
WRITE(20,11) NUMVTX + 1
FORMAT(IX,'NUMBER OF NODES = ',13)
WRITE(20,12) TIME1
FORMAT(IX,'CPU TIME = ',F5.2)
WRITE(20,+)
WRITE(20,13)
FORMAT(IX,' F(I) : PREDECESSOR OF NODE I :')
WRITE(20,'(10I6)')( FF(I),I = 1,NUMVTX)
WRITE(20,'(2I6)')J,K
WRITE(20,+)
WRITE(20,14)
FORMAT(IX,' W(I,F(I)) : WEIGHT OF EDGES :')
WRITE(20,'(10I6)')( UU(I),I = 1,NUMVTX)
WRITE(20,'(2I6)')MIN1,MIN2

J = 0
DO I = 1,NUMVTX
   J = J + UU(I)
END DO
WRITE(20,*)
WRITE(*,15) J+MIN1+MIN2
WRITE(*,15) J+MIN1+MIN2
FORMAT(IX,' TOTAL WEIGHT = ',I9)
lowerb = j+min1+min2
NUMVTX = NUMVTX+1
ID = 0
DO I = 1,NUMVTX
  PRINT*, ' DEGREE OF ', I, ' = ', DEGREE(I)
  ID = ID + DEGREE(I)
END DO
PRINT*, ' TOTAL DEGREE = ', ID
END ! END OF ONETREE

C ******************** SUBROUTINE PRIM ********************
C *** (SHORTEST SPANNING TREE PROBLEM) ***
C *** *** *** ***
C *** THE PROGRAM IS BASED ON THE PAPER ***
C *** P.M. CAMERINI, G.M. GALBIATI, F. MAFFIOLI ***
C *** "ALGORITHMS FOR FINDING OPTIMUM TREES: ***
C *** DESCRIPTIONS, USE AND EVALUATION", ***
C *** ANNALS OF OPERATIONS RESEARCH 7, 1988. ***
C *** *** ***
C ********************

SUBROUTINE PRIM (Q,U,F)

C SUBROUTINE PRIM COMPUTES A SPANNING TREE OF MINIMUM TOTAL
C WEIGHT BY PRIM'S METHOD
C THE GRAPH MUST BE SIMPLE AND COMPLETE
C
C WARNINGS
C
C MATRIX OF ARC WEIGHTS MUST BE COMPLETE AND SYMMETRIC
C SUBROUTINE PRIM DOES NOT CHECK COMPLETENESS NOR SYMMETRY
C
C OUTPUT ARGUMENTS
C
C F(I)  PREDECESSOR OF NODE I IN THE OUTWARDS ORIENTED TREE WITH ROO
C IF N IS LESS THAN OR EQUAL TO 1, PRIM RETURNS WITH F(1)=0
C (DIM. NRD)
C
C WORKING ARGUMENTS
C
C Q(.)  VECTOR OF ISOLATED NODES (DIM. NRD)

145
VECTOR OF CURRENT NODE WEIGHTS (DIM. NRD)

LOCAL VARIABLES

K ISOLATED NODE CHOSEN
I DO-LOOP INDEX
J DO-LOOP INDEX
JJ CONTAINER OF Q(J)
QO CARDINALITY OF Q

CALLED FUNCTION

PMIN RETURNS A NODE OF MINIMUM WEIGHT

TYPE AND DIMENSION STATEMENTS

IMPLICIT NONE
INTEGER MAXPTS
PARAMETER(MAXPTS = 51)
COMMON/MATRIX/DIST,NUMVTX,LOWERB
INTEGER DIST(MAXPTS,MAXPTS),NUMVTX,LOWERB
INTEGER F(MAXPTS),Q(MAXPTS),QO,PMIN,U(MAXPTS)

Integer i,j,k,jj

DATA STRUCTURES INITIALIZATION

F(1) = 0
QO = NUMVTX - 1
IF ( NUMVTX . GT . 1 ) THEN
  DO I=2,NUMVTX
    Q(I-1) = I
    F(I) = 1
    U(I) = DIST(1,I)
  ENDDO

MAIN ITERATION: DETERMINATION OF AN ARC OF MINIMUM WEIGHT
CONNECTING A NODE IN THE TREE TO AN EXTERNAL NODE. THIS
ARC IS THEN ADDED TO THE TREE.

DO WHILE( QO . GT . 1 )
   K = PMIN(U,Q,QO)

UPDATING OF LABELS = CLEANUP OPERATION

DO J=1,QO
   JJ = Q(J)
   IF ( U(JJ) . GT . DIST(K,JJ) ) THEN
      U(JJ) = DIST(K,JJ)
      F(JJ) = K
   ENDIF
ENDDO
ENDDO
ENDIF
RETURN
END

INTEGER FUNCTION PMIN (U,Q,QO)

FUNCTION PMIN EXTRACTS FROM Q AND RETURNS AN ISOLATED NODE K
OF MINIMUM WEIGHT U(K) IF QO IS GREATER THAN 0, ELSE RETURNS 0.

INPUT ARGUMENTS

U(.) VECTOR OF NODE WEIGHTS

INPUT/OUTPUT ARGUMENTS

Q(.) VECTOR OF ISOLATED NODES
QO CARDINALITY OF VECTOR Q

LOCAL VARIABLES

I DO-LOOP INDEX
II CURRENT NAME FOR Q(I)
P POSITION OF CURRENT OPTIMAL NODE

TYPE AND DIMENSION STATEMENTS
INTEGER MAXPTS
PARAMETER(MAXPTS = 51)
COMMON/MATRIX/DIST,NUMVTX,LOWEB
INTEGER DIST(MAXPTS,MAXPTS),NUMVTX,LOWEB

INTEGER Q(MAXPTS),QO,P,U(MAXPTS)

IF ( QO .LE. 0 ) THEN
  PMIN = 0
  RETURN
ENDIF
P = 1
PMIN = Q(1)
DO I=1,QO
  II = Q(I)
  IF ( U(II) .LT. U(PMIN) ) THEN
    P = I
    PMIN = II
  END IF
1000 ENDDO
Q(P) = Q(QO)
QO = QO - 1
RETURN
END

****************************************************************
****************************************************************

SUBROUTINE HEURISTIC_4(nnn,ddd,min_tsp)

IMPLICIT NONE
INTEGER*4 MAXPTS, BIGINT,min_tsp
PARAMETER (MAXPTS = 51, BIGINT = 999999)
INTEGER*4 MAXOPT
PARAMETER (MAXOPT = 30)
COMMON /POINTS/ X(MAXPTS), Y(MAXPTS), NUMPTS, MATSIZ
COMMON /DISTS / D(MAXPTS,MAXPTS), DM(MAXPTS,MAXPTS)
COMMON /LISTS / IROUTE(MAXPTS), ILIST(MAXPTS,3)
COMMON /OUTCOM/ FUNIT
COMMON/PRN$W/ KPRINT(10)
INTEGER*4 X, Y, NUMPTS, MATSIZ, NNN
INTEGER*4 D, DM, ddd(maxpts, maxpts)
INTEGER*4 IROUTE, ILIST
INTEGER*4 FUNIT, Kprint

common /keep/ dk(maxpts, maxpts)
integer*4 dk

C
C *** END OF BLOCK ***
C

DIMENSION TIMARR(20)
INTEGER*4 I, J, LTREE, UTREE
INTEGER*4 SEED, THRTOT, LIMTOT, DIFF
REAL SECOND(MAXOPT)
INTEGER*4 TOTALS(MAXOPT), RNUMB(MAXOPT)
INTEGER*4 KSWIT(10)
CHARACTER*8 RNAME(MAXOPT)
CHARACTER*50 OFILE, SFILE
Integer iand1, numptsless1

C
C *** BEGIN ***
C

C GET PROGRAM PARAMETERS AND GENERATING DISTANCE MATRIX
C

min_tsp=999999999

kprint(1)=1
kprint(2)=1
numpts=nnn

DO I = 1, NUMPTS
   d(i,i) = ddd(i,i)
   dk(i,i) = d(i,i)
dm(i,i) = d(i,i)
ENDDO

numptsless1=numpts-1
DO I = 1,NUMPTSless1
   iand1=i+1
   DO J = iand1,NUMPTS
      d(i,j) = ddd(i,j)
      d(j,i) = d(i,j)
      dk(i,j) = d(i,j)
      dk(j,i) = d(i,j)
      dm(i,j) = d(i,j)
      dm(j,i) = d(i,j)
   ENDDO
ENDDO

LTREE = 12
UTREE = 12

c print *, 'going into first heuristic'

C
C GET AND MEASURE THE "THREE" TOUR.
C
CALL SIMPTR
RNAME(1) = ' THREE '
c print *, 'back from simptr'
RNUMB(1) = 0
CALL THREE
c print *, 'back from three'
CALL TOURLN(TOTALS(1))

min_tsp=TOTALS(1)
return
end

SUBROUTINE TOURLN(TOTAL)
C ********************************************************************** TOURLN **********************************************************************
C

C FIND THE LENGTH OF THE SOLUTION IN "IROUTE".
C
C *** INSERTION COMMON BLOCK ***
C
IMPLICIT NONE
INTEGER*4 MAXPTS, BIGINT
PARAMETER (MAXPTS = 51, BIGINT = 999999)
COMMON /POINTS/ X(MAXPTS), Y(MAXPTS), NUMPTS, MATSIZ
COMMON /DIST/ D(MAXPTS,MAXPTS), DM(MAXPTS,MAXPTS)
COMMON /LISTS/ IROUTE(MAXPTS), ILIST(MAXPTS,3)
COMMON /OUTCOM/ FUNIT
COMMON/PRNSW/ KPRINT(10)
INTEGER*4 X, Y, NUMPTS, MATSIZ
INTEGER*4 D, DM
INTEGER*4 IROUTE, ILIST
INTEGER*4 FUNIT, Kprint

C *** END OF BLOCK ***
C
INTEGER*4 TOTAL, I
C
*** BEGIN ***
C
WRITE(1,*) ' TOURLN'
WRITE(1, '(1016)') (IROUTE(I), I=1,NUMPTS)
TOTAL = 0
DO I = 1, NUMPTS
   TOTAL = TOTAL + DM(I, ILIST(I,2))
ENDDO
RETURN
END

SUBROUTINE SIMPTR
C ****************************************************** SIMPTR ******************************************************
C
C GENERATE A TOUR FROM WHICH THE ROUTINES CAN WORK AND STORE THEM IN THE "IROUTE" AND "ILIST" ARRAYS.
C
C THE TOUR WILL BE 1-2-3-4-...-NUMPTS.
C
C
C *** INSERTION COMMON BLOCK ***
C
IMPLICIT NONE
INTEGER+4 MAXPTS, BIGINT
PARAMETER (MAXPTS = 51, BIGINT = 999999)
COMMON /POINTS/ X(MAXPTS), Y(MAXPTS), NUMPTS, MATSIZ
COMMON /DISTS/ D(MAXPTS, MAXPTS), DM(MAXPTS, MAXPTS)
COMMON /LISTS/ IROUTE(MAXPTS), ILIST(MAXPTS, 3)
COMMON /OUTCOM/ FUNIT
COMMON/PRNSW/ KPRINT(10)
INTEGER+4 X, Y, NUMPTS, MATSIZ
INTEGER+4 D, DM
INTEGER+4 IROUTE, ILIST
INTEGER+4 FUNIT, KPrint
C
C *** END OF BLOCK ***
C
INTEGER+4 I, J
C
C *** BEGIN ***
C
C IF(KPRINT(2) .EQ. 1) WRITE(1,*) ' SIMPTR'
J = NUMPTS
DO I = 1, NUMPTS
IROUTE(I) = I
ILIST(J, 2) = I
ILIST(I, 1) = J
J = I
ENDDO
RETURN
END

SUBROUTINE THREE
C********************************************************** THREEO **************************************************
C
THIS SUBROUTINE PERFORMS A DYNAMIC 3-OPT AS DESCRIBED
BY STEIGLITZ AND WEINER
AN INITIAL SOLUTION MUST BE GIVEN IN 'IROUTE' ARRAY

152
VARIABLES:
IRDUTE(K) -- POSITION ON ROUTE OF NODE K
D(I,J) -- DISTANCE FROM NODE I TO NODE J
ILIST(K,1) -- PREDECESSOR OF NODE K
(K,2) -- SUCCESSOR OF NODE K
NUMPTS -- NUMBER OF NODES
NEW -- NUMBER OF THE NODE BEING ADDED

*** INSERTION COMMON BLOCK ***

IMPLICIT NONE
INTEGER*4 MAXPTS, BIGINT
PARAMETER (MAXPTS = 51, BIGINT = 999999)
COMMON /POINTS/ X(MAXPTS), Y(MAXPTS), NUMPTS, MATSIZ
COMMON /DISTS / D(MAXPTS,MAXPTS), DM(MAXPTS,MAXPTS)
COMMON /LISTS / IROUTE(MAXPTS), ILIST(MAXPTS,3)
COMMON /OUTCOM/ FUNIT
COMMON/PRNSW/ KPRINT(10)
INTEGER*4 X, Y, NUMPTS, MATSIZ
INTEGER*4 D, DM
INTEGER*4 IROUTE, ILIST
INTEGER*4 FUNIT, KPrint

*** END OF BLOCK ***

C STORAGE FOR ARC LISTS FOR USE BY "THREE"

COMMON /LISTTT/ LIST(MAXPTS,2), LISTNO, ISAVD(MAXPTS), NEW
INTEGER*4 LIST, LISTNO, ISAVD, NEW

C STORAGE OF LATEST SST ARCS

COMMON /FRSST / ISIN(MAXPTS), XXX(MAXPTS), YYY(MAXPTS)
INTEGER*4 XXX, YYY
LOGICAL ISIN
Integer i,j,k,l,newm,i1,i2,j1,j2,k1,k2,jp,n1,n2,nsw,ladd
Integer kp, lp

C DISTANCE VARIABLES (KSW IS TYPE OF SWAP).
C
INTEGER*4 SAVMAX, DIST, DIST3, DIFF, KSW
Logical errflag

*** BEGIN ***

IF(KPRINT(2) .EQ. 1) WRITE(1,*,'(THREE')
LISTNO = 0
KSW = 5
SAVMAX = 0

MAKE A 3 ARC ROUTE USING THE FIRST THREE "IROUTE" POINTS

I = IROUTE(3)
DO L = 1,3
   J = IROUTE(L)
   ILIST(J,1) = I
   ILIST(I,2) = J
   I = J
ENDDO

SUCCESSIVELY ADD ONE MORE NODE TO THE ROUTE

DO NEW = 4,NUMPTS
   NEWM = NEW-1
   THIS SUBROUTINE CHOOSES THE LEAST EXPENSIVE PLACE TO INSERT NODE
   IROUTE(NEW) IN THE EXISTING ROUTE OF NEW-1 NODES
   CALL CPINST
   WE NEED TO EXAMINE EACH ARC IN THE LIST OF
   ARCS THAT HAVE BEEN CHANGED
   INITIALLY THIS LIST 'LIST(LISTNO,-)' WILL BE
   COMPOSED OF THE TWO ARCS ADDED BY CPINST.
   HOWEVER, IF ANY ARCS HAVE BEEN CHANGED BY
   THE 3-OPT, THESE ARCS HAVE BEEN ADDED TO THIS
   LIST IN SUBROUTINE REARR.

100 Errflag = .TRUE.
   L= 1
   DO While ((Errflag).AND.(L.LE.NEW))
      IF (IROUTE(L) .EQ. LIST(LISTNO,1)) THEN
I1 = LIST(LISTNO,1)
IF (ILIST(I1,2).EQ.LIST(LISTNO,2)) THEN
   I2 = LIST(LISTNO,2)
   LISTNO = LISTNO - 1
Errflag = .FALSE.
   GO TO 200
ENDIF
ENDIF
IF (IROUTE(L) .EQ. LIST(LISTNO,2)) THEN
   I1 = LIST(LISTNO,2)
   IF (ILIST(I1,2).EQ.LIST(LISTNO,1)) THEN
      I2 = LIST(LISTNO,1)
      LISTNO = LISTNO - 1
   ENDIF
   Errflag = .FALSE.
ENDIF
ENDIF
L = L + 1
200 ENDDO
IF (Errflag) GO TO 888
C
C NO SERIOUS ERRORS -- CONTINUE. GET THREE ARCS NAMED
C "I1-J1", "J1-J2", AND "K1-K2".
C
J1 = I1
DO J = 2,NEWM
   JP = J + 1
   J1 = ILIST(J1,2)
   J2 = ILIST(J1,2)
   DIST = D(I1,I2) + D(J1,J2)
   K1 = J1
   DO K = JP,NEW
      KP = K + 1
      K1 = ILIST(K1,2)
      K2 = ILIST(K1,2)
      DIST3 = DIST + D(K1,K2)
   C
   C TEST THE ARRANGEMENT "I1-J1" "I2-K1" "J2-K2"
   C
      DIFF = DIST3 - (D(I1,J1)+D(I2,K1)+D(J2,K2))
      IF (DIFF.GT.SAVMAX) THEN
         SAVMAX = DIFF
   C
   C
KSW = 1
ENDIF
C
C TEST THE ARRANGEMENT "I1-J2" "K1-I2" "J1-K2"
C
DIFF = DIST3 - (D(I1,J2)+D(K1,I2)+D(J1,K2))
IF (DIFF.GT.SAVMAX) THEN
SAVMAX = DIFF
KSW = 2
ENDIF
C
C TEST THE ARRANGEMENT "I1-J2" "K1-J1" "I2-K2"
C
DIFF = DIST3 - (D(I1,J2)+D(K1,J1)+D(I2,K2))
IF (DIFF.GT.SAVMAX) THEN
SAVMAX = DIFF
KSW = 3
ENDIF
C
C TEST THE ARRANGEMENT "I1-K1" "J2-I2" "J1-K2"
C
DIFF = DIST3 - (D(I1,K1)+D(J2,I2)+D(J1,K2))
IF (DIFF.GT.SAVMAX) THEN
SAVMAX = DIFF
KSW = 4
ENDIF
C
C REMOVE ARCS --(I1,I2),(J1,J2),(K1,K2)-- FROM LIST
C
IF (KSW.NE.5) THEN
  IF (LISTNO.NE.0) THEN
    DO NSW = 1, 3
      IF (NSW.EQ.1) THEN
        N1 = I1
        N2 = I2
      ELSE
        IF (NSW.EQ.2) THEN
          N1 = J1
          N2 = J2
        ELSE
          N1 = K1
      ENDIF
    ENDDO
  ENDIF
ENDIF
N2 = K2
ENDIF
ENDIF

LADD = 0
DO L = 1, LISTNO
   LP = L - LADD
   IF ((LIST(L,1).EQ.N1.AND.LIST(L,2).EQ.N2).OR.
      (LIST(L,2).EQ.N1.AND.LIST(L,1).EQ.N2)) THEN
      LADD = 1
   ELSE
      LIST(LP,1) = LIST(L,1)
      LIST(LP,2) = LIST(L,2)
   ENDIF
ENDDO
LISTNO = LISTNO - LADD
ENDDO
ENDIF

C
C SUBROUTINE REARR MAKES THE ARC SWAPS
C
If (KSW.EQ.1) THEN
   CALL REARR(I1,J1,I2,K1,J2,K2,KSW)
ELSE IF (KSW.EQ.2) THEN
   CALL REARR(I1,J2,K1,I2,J1,K2,KSW)
ELSE IF (KSW.EQ.3) THEN
   CALL REARR(I1,J2,K1,J1,I2,K2,KSW)
ELSE IF (KSW.EQ.4) THEN
   CALL REARR(I1,K1,J2,I2,J1,K2,KSW)
ENDIF
SAVMAX = 0
KSW = 5
GO TO 100
ENDDO

800 ENDDO
ENDDO
IF (LISTNO.GT.0) Go to 100
C
C IF THE "LIST" ARRAY IS NOT EMPTY, REDO THIS ITERATION
C
ENDDO

C
1 FORMAT(10I4)

C
888 If (.Not.Errflag) Then
RETURN
ELSE

C
C IF THE PROGRAM CAN'T GET PAST HERE, THERE ARE SERIOUS ERRORS IN
THE "LIST", "IROUTE", AND/OR "ILIST" ARRAYS.
C
C
WRITE(FUNIT,55)
WRITE(FUNIT,56) L, IROUTE(L), ILIST(I1,2)
WRITE(' *** ERRORS FOUND IN LISTS')
WRITE(' L ',13,' IROUTE(L) ',13,' NEXT NODE ',13)

Print *, 'Numpts ', Numpts
Print *, 'IROUTE ', (I, IROUTE(I) , 1 = 1, New)
Print *, 'Listno is ', Listno
C Print *, 'List ', (J, K, LIST(J,K), J=l, New, K=1, New)
STOP
ENDIF

END

SUBROUTINE CPINST
C***************************************************************************
C THIS ROUTINE CHOOSES WHERE TO INSERT THE NEWTH NODE IN
THE EXISTING ROUTE. IT REORDERS THE ROUTE WITH
THIS NODE IN POSITION 2
C
C *** INSERTION COMMON BLOCK ***
C
IMPLICIT NONE
INTEGER*4 MAXPTS, BIGINT
PARAMETER (MAXPTS = 51, BIGINT = 999999)
COMMON /POINTS/ X(MAXPTS), Y(MAXPTS), NUMPTS, MATSIZ
COMMON /DISTS / D(MAXPTS.MAXPTS), DM(MAXPTS.MAXPTS)
COMMON /LISTS / IROUTE(MAXPTS), ILIST(MAXPTS,3)
COMMON /OUTCOM/ FUNIT
COMMON/PRNSW/ KPRINT(10)
INTEGER*4 X, Y, NUMPTS, MATSIZ
INTEGER*4 D, DM
INTEGER*4 IROUTE, ILIST
INTEGER*4 FUNIT, KPrint

C
C *** END OF BLOCK ***
C
C STORAGE FOR LIST ARRAYS FROM "THREEQ"
C
COMMON /LISTTT/ LIST(MAXPTS,2), LISTNO, ISAVD(MAXPTS), NEW
INTEGER*4 LIST, LISTNO, ISAVD, NEW

C
C DISTANCE AND ENDPOINT VARIABLES

INTEGER*4 K, ID, IDIST
Integer i,j,l,lp,lsav,newm

C
C *** BEGIN ***
C
C FIND CHEAPEST PLACE TO INSERT NODE K IN THE EXISTING ROUTE
C
K = IROUTE(NEW)
IDIST = BIGINT * 2
NEWM = NEW - 1
I = IROUTE(NEWM)
DO L = 1,NEWM
ISAVD(L) = IROUTE(L)
J = IROUTE(L)
ID = (DM(I,K) + DM(J,K)) - DM(I,J)
C
C IF THE DISTANCE DIFFERENCE IS NOT BETTER THAN THE PREVIOUS BEST,
C SKIP AND START ANOTHER TEST.
C
IF (ID.LT.IDIST) THEN
LSAV = L
ILIST(K,1) = I
ILIST(K,2) = J
IDIST = ID
ENDIF
I J
ENDDO
C
C INSERT K AFTER IROUTE(LSAV-1)=I AND MAKE NODE I THE FIRST NODE
C ON THE ROUTE.
C
I = ILIST(K,1)
J = ILIST(K,2)
ILIST(I,2) = K
ILIST(J,1) = K
IROUTE(1) = I
IROUTE(2) = K
DO L = 3,NEW
    LP = LSAV + L - 3
    IROUTE(L) = ISAVD(LP)
    IF (LP.EQ.NEWM) LSAV = LSAV-NEWM
ENDDO
LIST(1,1) = I
LIST(1,2) = K
LIST(2,1) = K
LIST(2,2) = J
LISTNO = 2
RETURN
END

SUBROUTINE REARR(I1,I2,J1,J2,K1,K2,KSW)
C******************************************************************************
C
C THIS ROUTINE REARRANGES THE SEQUENCE OF NODES IN IROUTE
C
C *** INSERTION COMMON BLOCK ***
C
IMPLICIT NONE
INTEGER*4 MAXPTS, BIGINT
PARAMETER (MAXPTS = 51, BIGINT = 999999)
COMMON /POINTS/ X(MAXPTS), Y(MAXPTS), NUMPTS, MATSIZ
COMMON /DISTS / D(MAXPTS,MAXPTS), DM(MAXPTS,MAXPTS)
COMMON /LISTS / IROUTE(MAXPTS), ILIST(MAXPTS,3)
COMMON /OUTCOM/ FUNIT
COMMON/PRNSW/ KPRINT(10)
C******************************************************************************

160
INTEGER*4 X, Y, NUMPTS, MATSIZ
INTEGER*4 D, DM
INTEGER*4 IROUTE, ILIST
INTEGER*4 FUNIT, KPrint

C
C *** END OF BLOCK ***
C
C STORAGE FOR LIST ARRAYS FROM "THREEO"
C
COMMON /LISTTT/ LIST(MAXPTS,2), LISTNO, ISAVD(MAXPTS), NEW
INTEGER*4 LIST, LISTNO, ISAVD, NEW

C
C *** BEGIN ***
C
ILIST(I1,2) = I2
IF ((KSW.NE.2).AND.(KSW.NE.3)) THEN
C
C REORDER ROUTE FROM I2 TO J1
C
I = I2
ILSAV = ILIST(I,1)
ILIST(I,1) = I1
ILIST(I,2) = ILSAV
DO WHILE (I.NE.J1)
I = ILSAV
ILSAV = ILIST(I,1)
ILIST(I,1) = ILIST(I,2)
ILIST(I,2) = ILSAV
ENDDO
ENDIF
ILIST(J1,2) = J2
ILIST(I2,1) = I1
IF ((KSW.NE.2).AND.(KSW.NE.4)) THEN
C
C REORDER ROUTE FROM J2 TO K1
C
J = J2
JLSAV = ILIST(J,1)
ILIST(J,1) = J1
ILIST(J,2) = JLSAV
DO WHILE (J.NE.K1)
  J = JLSAV
  JLSAV = ILIST(J,1)
  ILIST(J,1) = ILIST(J,2)
  ILIST(J,2) = JLSAV
ENDDO
ENDIF
ILIST(K1,2) = K2
ILIST(J2,1) = J1
ILIST(K2,1) = K1
I = IROUTE(1)
DO L = 2,NEW
  IROUTE(L) = ILIST(I,2)
  I = IROUTE(L)
ENDDO
IF (LISTNO.EQ.-1) RETURN

C
C ADD THE NEW ARCS --(I1,I2),(J1,J2),(K1,K2)-- TO THE
C LIST OF ARCS THAT MUST BE EXAMINED
C
LISTNO = LISTNO + 1
LIST(LISTNO,1) = K1
LIST(LISTNO,2) = K2
LISTNO = LISTNO + 1
LIST(LISTNO,1) = J1
LIST(LISTNO,2) = J2
LISTNO = LISTNO + 1
LIST(LISTNO,1) = I1
LIST(LISTNO,2) = I2
RETURN
END

C Program His2.f
C This is a Worker Program
C The Message Types are as follows: 1) send me a route
C 5) I am exiting
C
include 'fpvm3.h'
IMPLICIT NONE
Integer MaxCit
INTEGER MAXPTS
PARAMETER(MAXPTS = 51)
Parameter (MaxCit = MAXPTS - 1)
COMMON/MATRIX/DIST,NUMVTX,LOWEB
INTEGER DIST(MAXPTS,MAXPTS),NUMVTX,CITIES,LOWEB
INTEGER DISTANCE(MAXPTS,MAXPTS),NodeOf(Maxpts)
INTEGER X(MAXPTS),Y(MAXPTS),NV,STOPTIME,TIMELIMIT
INTEGER MINW,MAXW,I,W(MAXPTS),J
INTEGER CARDINALITY,TotW
Integer info, mytid, pid, Init_lowerb
Integer error, power, PvmDataRaw, oldval
Integer RtArray(MaxCit), Orig_Indualt, LastCit
integer indual(MAXPTS),indualt, nz, DistParam
integer tagnum, lastrt, diff, whereindex, k
integer rtid, rtag, rlen
integer mymatrix(92200, Maxpts + 1), pos, mycount
integer PvmRoute, PvmRouteDirect
logical stop, set, maxed
REAL DUALT,GAP
Real Dual(MAXPTS)

integer timel, time2, mclock
external function mclock
real mysecs

DistParam = Maxpts * Maxpts

timel = mclock()

Call pvmfmytid(mytid)
Print *, 'Mytid is ',Mytid
Call pvmfsetopt(PvmRoute, PvmRouteDirect, oldval)
Call pvmfparent(pid)

Call pvmfrecv(pid, 8, info)
Call pvmfunpack(INTEGER4, Cities, 1, 1, info)
Call pvmfunpack(INTEGER4, MaxW, 1, 1, info)
Call pvmfunpack(INTEGER4, NV, 1, 1, info)
Call pvmfunpack(INTEGER4, StopTime, 1, 1, info)
Call pvmfunpack(INTEGER4, TimeLimit, 1, 1, info)
Call pvmfunpack(INTEGER4, Gap, 1, 1, info)
Call pvmfunpack(INTEGER4, minW, 1, 1, info)
Call pvmfunpack(REAL4, dualt, 1, 1, info)
Call pvmfunpack(INTEGER4, indualt, 1, 1, info)
Orig_Indualt = indualt
Call pvmfunpack(INTEGER4, lowerb, 1, 1, info)
Call pvmfunpack(INTEGER4, power, 1, 1, info)
Init_lowerb = lowerb

Call pvmfprecvCpid,9,X,Cities,INTEGER4,rtid,rtag,rlen,info)
Call pvmfprecvCpid,19,Y,Cities,INTEGER4,rtid,rtag,rlen,info)
Call pvmfprecvCpid,29,W,Cities,INTEGER4,rtid,rtag,rlen,info)
Call pvmfprecvCpid,39,dual,Cities,REAL4,rtid,rtag,rlen,info)
Call pvmfprecvCpid,49,indual,Cities,INTEGER4,rtid,rtag,rlen,info)

Do j = 1, Cities + 1
   Call pvmfprecvCpid,ll,distCl,j),Cities + 1,INTEGER4,
      &       rtid,rtag,rlen,info)
ENDDO

Do j = 1, Cities + 1
   Call pvmfprecvCpid, 111, distanceCl,j),
      &       Cities + 1, INTEGER4,rtid,rtag,rlen, info)
ENDDO

C Wait for initial tag to be sent and receive mine

Call pvmfprecvCpid,2,TagNum,1,INTEGER4,rtid,rtag,rlen,info)
Print *, 'My tag num: ',Tagnum

C While there are still routes to check,
C transform the given route number into a binary array
C check the route (if it is feasible, send message to I/O)
C The number of Rts I have to check is 2 ** of spaces
C left in the RtArray past my tag number

lowerb = Init_lowerb
Call binary(Tagnum, Cities, Power, RtArray,Diff)

LastRt = 2**Diff
Stop = .false.
k = 1
Whereindex = 1

Do while (k.LE.LastRt and .not. stop)
   C   Print *(, 'This Rt: ', (RtArray(i), i=1, Cities)

   Cardinity = 0
   TotW = 0
   Indualt = Orig_Indualt

   LastCit = 0
   maxed = .false.

   Do I = 1, Cities
      If (RtArray(I).EQ.1) then
         TotW = TotW + W(I)
         indualt = indualt + indual(I)
         Cardinity = Cardinity + 1
         Nodeof(Cardinity) = I
         DIST(1, CARDINALITY+1) = DISTANCE(Cities+1, I)
         DIST(CARDINALITY+1, I) = DISTANCE(I, Cities+1)
         DO J=1, CARDINALITY-1
            DIST(J+1, CARDINALITY+1) = DISTANCE(NODEOF(J), I)
            DIST(Cardinality+1, J+1) = DISTANCE(I, NODEOF(J))
         ENDDO
         LastCit = I
      ENDIF
   ENDDO
   If (TotW .GE. minW) THEN
      If (TotW .LE. MaxW) then
         DIST(Numvtx, Numvtx) = 9999999999
         Numvtx = Cardinity + 1
         C Find total weight and Cardinality of route, set up new dist matrix!
         C If weight is between max and min, then do the following:

         If (Cardinality.GT.1) Then
call onetree
IF(LOWERB+1000*CARDINALITY*STOPTIME.LE.1000*TIMELIMIT) then
  lowerb = Init_lowerb
call Heuristic_4(Numvtx, dist, lowerb)
  IF(LOWERB+1000*CARDINALITY*STOPTIME.LE.1000*TIMELIMIT.
    &
    AND.lowerb-indualt.lt.int(gap)) Then
    nz = lowerb-indualt
  mymatrix(Whereindex,1) = nz
  mymatrix(Whereindex,2) = lowerb
do j = 1, Cities
    mymatrix(Whereindex,j+2) = RtArray(j)
  enddo
  Whereindex = Whereindex + 1
  C Print *,'Route was printable'
  ENDIF
ENDIF
ELSE
  If (Cardinality.GT.1) Then
    lowerb = dist(1,2) + dist(2,1)
    IF(LOWERB+1000*CARDINALITY*STOPTIME.LE.1000*TIMELIMIT.
      &
      AND.lowerb-indualt.lt.int(gap)) Then
      nz = lowerb-indualt
    mymatrix(Whereindex,1) = nz
    mymatrix(Whereindex,2) = lowerb
do j = 3, Cities + 2
    mymatrix(Whereindex,j) = RtArray(j-2)
  enddo
  Whereindex = Whereindex + 1
  C Print *,'Route was printable'
  ENDIF
ENDIF
ELSE
  Maxed = .true.
  C Print *, 'Rt was TOO heavy'
  ENDIF
ELSE
C Print *, 'Rt was TOO light! '

ENDIF

C Set the next route to be the next binary value
C through the bit setting and resetting below

k = k + 1

Set = .FALSE.

If (maxed) Then
   Pos = LastCit
   maxed = .false.
Else
   Pos = Cities
ENDIF
Do while (.NOT.Set)
   If (RtArray(Pos).EQ.0) Then
      RtArray(Pos) = 1
      Set = .TRUE.
   Else
      RtArray(Pos) = 0
      If (Pos.GT.(Power+1)) Then
         Pos = Pos - 1
      ELSE
         Set = .true.
         Stop = .true.
      ENDIF
   End if
ENDDO

ENDDO

time2 = mclock()
mysecs = real(time2-time1)/100.0

MyCount = Whereindex - 1
C Send master process mycount and mymatrix
Call pvmfrecv(pid, 22, info)
Call pvmfinitsend(PvmDataRaw, info)
Call pvmfsend(pid, 7, mycount, 1, INTEGER4, info)
Print *, 'Total number of routes from ', mytid, ' is ', mycount
If (mycount.GT.0) Then
    Do j = 1, mycount
        Call pvmfinitsend(PvmDataRaw, info)
        Do k = 1, Cities + 2
            Call pvmfpack(INTEGER4, mymatrix(j,k), 1, 1, info)
        ENDDO
    Call pvmfsend(pid, 6, info)
    ENDDO
    Print *, 'I have sent my routes'
Endif

Call pvmfrecv(pid, 4, info)
Print *, 'I have received messsage back from master'

Call pvmfinitsend(PvmDataRaw, info)
Call pvmfsend(pid, 3, mysecs, 1, REAL4, info)
Print *, 'I have sent my seconds to master'

Call pvmfrecv(pid, 44, info)
Print *, 'I have received YouAreFinished messsage from master'

C Tell master process I am exiting

Call pvmfinitsend(PvmDataRaw, info)
Call pvmfsend(pid, 5, mytid, 1, INTEGER4, info)
Print *, 'Tell Master, ', pid, ', that ', mytid, ' is exiting'

C Then, exit

Print *, 'I am exiting'
Call pvmfexit(error)

Stop
END
Subroutine Binary takes a route number, Number, and
the number of cities, C, which DOES NOT include the depot,
and returns the binary route array, RtArray, and Diff

Subroutine binary(Number, C, Pow, RtArray, Diff)
IMPLICIT NONE

Integer Number, Count, Newnum
Integer C, Diff, i, Pow
Integer RtArray(C), Temp(21)

Count = 0
Do while(Number.NE.0)
  NewNum = Number/2
  If ((NewNum * 2).EQ.Number) Then
    Temp(C - Count) = 0
  Else
    Temp(C - Count) = 1
  ENDIF
  Count = Count + 1
  Number = NewNum
ENDDO

Do i = 1, C
RtArray(i) = 0
ENDDO

Diff = C - Pow
Do i = 1, C-Diff
  RtArray(i) = Temp(i + Diff)
ENDDO

END

SUBROUTINE ONETREE
IMPLICIT NONE
INTEGER MAXPTS
PARAMETER(MAXPTS = 51)
COMMON/MATRIX/DIST,NUMVTX,LOWERB
INTEGER DIST(MAXPTS,MAXPTS),NUMVTX,LOWER

INTEGER FF(MAXPTS),UU(MAXPTS),QQ(MAXPTS)
INTEGER DEGREE(MAXPTS)
COMMON/DEG/DEGREE

Integer  j, i, k, min1, min2,id

J = 1
MIN1 = DIST(1,NUMVTX)
DO I = 2,NUMVTX - 1
   IF(MIN1 .GT. DIST(I,NUMVTX)) THEN
      MIN1 = DIST(I,NUMVTX)
      J = I
   END IF
END DO

IF(J .NE. 1) THEN
   K = 1
ELSE
   K = 2
END IF

MIN2 = DIST(K,NUMVTX)
DO I = K+1,NUMVTX - 1
   IF(MIN2 .GT. DIST(I,NUMVTX) .AND. I .NE. J) THEN
      MIN2 = DIST(I,NUMVTX)
      K = I
   END IF
END DO

NUMVTX = NUMVTX - 1

TIME0 = CTIME()
CALL PRIM(QQ,UU,FF)
DO I = 1,NUMVTX
   DEGREE(I) = 0
   ! extracting degree
END DO
DO I = 2,NUMVTX
   DEGREE(FF(I)) = DEGREE(FF(I))+1
   ! of each node
DEGREE(I) = DEGREE(I)+1
END DO
DEGREE(J) = DEGREE(J)+1
DEGREE(K) = DEGREE(K)+1
DEGREE(NUMVTX+1) = 2

TIME1 = CTIME() - TIME0
WRITE(20,11) NUMVTX+1
11 FORMAT(1X,'NUMBER OF NODES = ',I3)
WRITE(20,12)TIME1
12 FORMAT(1X,'CPU TIME = ',F5.2)
WRITE(20,*)
WRITE(20,13)
13 FORMAT(1X,' F(I) : PREDECESSOR OF NODE I :')
WRITE(20,'(1016)')( FF(I),I = 1,NUMVTX)
WRITE(20,'(2I6)')J,K
WRITE(20,*)
WRITE(20,14)
14 FORMAT(1X,' W(I,F(I)) : WEIGHT OF EDGES :')
WRITE(20,'(10I6)')( UU(I),I = 1,NUMVTX)
WRITE(20,'(216)')MIN1,MIN2

J = 0
DO I = 1,NUMVTX
     J = J + UU(I)
END DO
WRITE(20,*)
WRITE(*,15) J+MIN1+MIN2
15 FORMAT(1X,' TOTAL WEIGHT = ',I9)
lowerb=j+min1+min2
NUMVTX = NUMVTX+1
ID = 0
DO I = 1,NUMVTX
     PRINT*, ' DEGREE OF',I,' = ',DEGREE(I)
     ID = ID + DEGREE(I)
END DO
PRINT*, ' TOTAL DEGREE = ',ID
END ! END OF ONETREE

171
SUBROUTINE PRIM (Q,U,F)

SUBROUTINE PRIM COMPUTES A SPANNING TREE OF MINIMUM TOTAL WEIGHT BY PRIM'S METHOD
THE GRAPH MUST BE SIMPLE AND COMPLETE

WARNINGS

MATRIX OF ARC WEIGHTS MUST BE COMPLETE AND SYMMETRIC
SUBROUTINE PRIM DOES NOT CHECK COMPLETENESS NOR SYMMETRY

OUTPUT ARGUMENTS

F(I) PREDECESSOR OF NODE I IN THE OUTWARDS ORIENTED TREE WITH ROOT
IF N IS LESS THAN OR EQUAL TO 1, PRIM RETURNS WITH F(1)=0 (DIM. NRD)

WORKING ARGUMENTS

Q(.) VECTOR OF ISOLATED NODES (DIM. NRD)
U(.) VECTOR OF CURRENT NODE WEIGHTS (DIM. NRD)

LOCAL VARIABLES

K ISOLATED NODE CHOSEN
I DO-LOOP INDEX
J DO-LOOP INDEX
JJ CONTAINER OF Q(J)
QO CARDINALITY OF Q

172
CALLED FUNCTION

PMIN RETURNS A NODE OF MINIMUM WEIGHT

TYPE AND DIMENSION STATEMENTS

IMPLICIT NONE
INTEGER MAXPTS
PARAMETER(MAXPTS = 51)
COMMON/MATRIX/DIST,NUMVTX,LOWERS
INTEGER DIST(MAXPTS,MAXPTS),NUMVTX,LOWERS
INTEGER F(MAXPTS),Q(MAXPTS),QO,PMIN,U(MAXPTS)

Integer i, j, k, jj

DATA STRUCTURES INITIALIZATION

F(1) = 0
QO = NUMVTX - 1
IF ( NUMVTX . GT . 1 ) THEN
   DO I=2,NUMVTX
      Q(I-1) = I
      F(I) = 1
      U(I) = DIST(1,I)
   ENDDO

MAIN ITERATION: DETERMINATION OF AN ARC OF MINIMUM WEIGHT
CONNECTING A NODE IN THE TREE TO AN EXTERNAL NODE. THIS
ARC IS THEN ADDED TO THE TREE.

DO WHILE( QO . GT . 1 )
   K = PMIN(U,Q,QO)

UPDATING OF LABELS = CLEANUP OPERATION

DO J=1,QO
   JJ = Q(J)
IF ( U(JJ) . GT . DIST(K,JJ) ) THEN
  U(JJ) = DIST(K,JJ)
  F(JJ) = K
ENDIF
ENDDO
ENDDO
ENDIF
RETURN
END

INTEGER FUNCTION PMIN (U,Q,QO)

C
C FUNCTION PMIN EXTRACTS FROM Q AND RETURNS AN ISOLATED NODE K
C OF MINIMUM WEIGHT U(K) IF QO IS GREATER THAN 0, ELSE RETURNS 0.
C
C INPUT ARGUMENTS
C
C U(.) VECTOR OF NODE WEIGHTS
C
C INPUT/OUTPUT ARGUMENTS
C
C Q(.) VECTOR OF ISOLATED NODES
C QO CARDINALITY OF VECTOR Q
C
C LOCAL VARIABLES
C
C I DO-LOOP INDEX
C II CURRENT NAME FOR Q(I)
C P POSITION OF CURRENT OPTIMAL NODE
C
C TYPE AND DIMENSION STATEMENTS
C
INTEGER MAXPTS
PARAMETER(MAXPTS = 51)
COMMON/MATRIX/DIST,NUMVTX,LOWERB
INTEGER DIST(MAXPTS,MAXPTS),NUMVTX,LOWERB

INTEGER Q(MAXPTS),QO,P,U(MAXPTS)

IF ( QO .LE. 0 ) THEN
  PMIN = 0
RETURN
ENDIF
P = 1
PMIN = Q(1)
DO  I=1,QO
II = Q(I)
IF ( U(II) .LT. U(PMIN) ) THEN
P = I
PMIN = II
ENDIF
1000 ENDDO
Q(P) = Q(QO)
QO = QO - 1
RETURN
END

******************************************************************************
******************************************************************************

SUBROUTINE HEURISTIC_4(nnn,ddd,min_tsp)

IMPLICIT NONE
INTEGER MAXPTS, BIGINT,min_tsp
PARAMETER (MAXPTS = 51, BIGINT = 999999)
INTEGER MAXOPT
PARAMETER (MAXOPT = 30)
COMMON /POINTS/ X(MAXPTS), Y(MAXPTS), NUMPTS, MATSIZ
COMMON /DISTS / D(MAXPTS,MAXPTS), DM(MAXPTS,MAXPTS)
COMMON /LISTS / IROUTE(MAXPTS), ILIST(MAXPTS,3)
COMMON /OUTCOM/ FUNIT
COMMON/PRNSW/ KPRINT(10)
INTEGER X, Y, NUMPTS, MATSIZ,NNN
INTEGER D, DM, ddd(maxpts, maxpts)
INTEGER IROUTE, ILIST
INTEGER FUNIT, KPRINT

common /keep/ dk(maxpts,maxpts)
integer*4 dk

175
INTEGER+4 I, J, LTREE, UTREE
INTEGER+4 NTREES, OPTION
INTEGER+4 SEED, THRTOT, LIMTOT, DIFF
REAL SECOND(MAXOPT)
INTEGER+4 TOTALS(MAXOPT)
, MINUTE(MAXOPT)
INTEGER+4 RNUMB(MAXOPT)
INTEGER+4 KSWIT(10)
CHARACTER*8 RNAME(MAXOPT)
CHARACTER*50 OFILE, SFILE

Integer numptsless1
Integer iand1

C *** BEGIN ***

C GET PROGRAM PARAMETERS AND GENERATING DISTANCE MATRIX

min_tsp=999999999
kprint(1)=1
kprint(2)=1
numpts=nnn

DO I = 1, NUMPTS
   d(i,i) = ddd(i,i)
   dk(i,i) = d(i,i)
   dm(i,i) = d(i,i)
ENDDO

numptsless1=numpts-1
DO I = 1, NUMPTSless1
   iand1=i+1
   DO J = iand1, NUMPTS
d(i, j) = ddd(i, j)
d(j, i) = d(i, j)
   dk(i, j) = d(i, j)
dk(j, i) = d(i, j)
dm(i, j) = d(i, j)
dm(j, i) = d(i, j)
ENDDO
ENDDO

LTREE = 12
UTREE = 12
  print *, 'going into first heuristic'

c GET AND MEASURE THE "THREE" TOUR.
c
   CALL SIMPTR
   RNAME(l) = ' THREE '
c   print *, 'back from simptr'
   RNUMB(l) = 0
   CALL THREE
   print *, 'back from three'
   CALL TOURLN(TOTALS(1))
   min_tsp=total(1)
   return
end

SUBROUTINE TOURLN(TOTAL)
c ********************************************************* TOURLN *********************************************************
c
C FIND THE LENGTH OF THE SOLUTION IN "IRoute".
c
C *** INSERTION COMMON BLOCK ***
c
   IMPLICIT NONE
   INTEGER*4 MAXPTS, BIGINT
   PARAMETER (MAXPTS = 51, BIGINT = 999999)
COMMON /POINTS/ X(MAXPTS), Y(MAXPTS), NUMPTS, MATSIZ
COMMON /DISTS / D(MAXPTS,MAXPTS), DM(MAXPTS,MAXPTS)
COMMON /LISTS / IROUTE(MAXPTS), ILIST(MAXPTS,3)
COMMON /OUTCOM/ FUNIT
COMMON/PRNSW/ KPRINT(10)
INTEGER*4 X, Y, NUMPTS, MATSIZ
INTEGER*4 D, DM
INTEGER*4 IROUTE, ILIST
INTEGER*4 FUNIT, KPRINT

C
C *** END OF BLOCK ***
C
C

INTEGER*4 TOTAL, I

C
C *** BEGIN ***
C
C WRITE(1,*) ' TOURLN'
C WRITE(1,'(10I6)') (IRoute(I),I=1,NUMPTS)
TOTAL = 0
DO I = 1,NUMPTS
   TOTAL = TOTAL + DM(I,ILIST(I,2))
ENDDO
RETURN
END

SUBROUTINE SIMPTR
C ************************************************** SIMPTR **************************************************
C
C Generate a tour from which the routines can work and store them in the "IRoute" and "ILIST" arrays.
C
C The tour will be 1-2-3-4-. . .-NUMPTS.
C
C *** INSERTION COMMON BLOCK ***
C
IMPLICIT NONE
INTEGER*4 MAXPTS, BIGINT
PARAMETER (MAXPTS = 51, BIGINT = 999999)
COMMON /POINTS/ X(MAXPTS), Y(MAXPTS), NUMPTS, MATSIZ
COMMON /DISTS / D(MAXPTS,MAXPTS), DM(MAXPTS,MAXPTS)
COMMON /LISTS / IROUTE(MAXPTS), ILIST(MAXPTS,3)
COMMON /OUTCOM/ FUNIT
COMMON/PRNSW/ KPRINT(10)
INTEGER*4 X, Y, NUMPTS, MATSIZ
INTEGER*4 D, DM
INTEGER*4 IROUTE, ILIST
INTEGER*4 FUNIT, KPRINT

C
C *** END OF BLOCK ***
C
INTEGER*4 I, J

C *** BEGIN ***
C
IF(KPRINT(2) .EQ. 1) WRITE(1,*) ' SIMPTR'
J = NUMPTS
DO I = 1,NUMPTS
  IROUTE(I) = I
  ILIST(J,2) = I
  ILIST(I,1) = J
  J = I
ENDDO
RETURN
END

SUBROUTINE THREE
C*************************************************************************** THREEQ
C***************************************************************************
C
THIS SUBROUTINE PERFORMS A DYNAMIC 3-OPT AS DESCRIBED
BY STEIGLITZ AND WEINER
AN INITIAL SOLUTION MUST BE GIVEN IN 'IROUTE' ARRAY

VARIABLES:
IROUTE(K) -- POSITION ON ROUTE OF NODE K
D(I,J) -- DISTANCE FROM NODE I TO NODE J
ILIST(K,1) -- PREDECESSOR OF NODE K
(k,2) -- SUCCESSOR OF NODE K
NUMPTS -- NUMBER OF NODES
NEW -- NUMBER OF THE NODE BEING ADDED

179
*** INSERTION COMMON BLOCK ***

IMPLICIT NONE
INTEGER*4 MAXPTS, BIGINT
PARAMETER (MAXPTS = 51, BIGINT = 999999)
COMMON /POINTS/ X(MAXPTS), Y(MAXPTS), NUMPTS, MATSIZ
COMMON /DISTS/ D(MAXPTS,MAXPTS), DM(MAXPTS,MAXPTS)
COMMON /LISTS/ IROUTE(MAXPTS), ILIST(MAXPTS,3)
COMMON /OUTCOM/ FUNIT
COMMON/PRNSW/ KPRINT(10)
INTEGER*4 X, Y, NUMPTS, MATSIZ
INTEGER*4 D, DM
INTEGER*4 IROUTE, ILIST
INTEGER*4 FUNIT, Kprint

*** END OF BLOCK ***

STORAGE FOR ARC LISTS FOR USE BY "THREE"

COMMON /LISTTT/ LIST(MAXPTS,2), LISTNO, ISAVD(MAXPTS), NEW
INTEGER*4 LIST, LISTNO, ISAVD, NEW

STORAGE OF LATEST SST ARCS

COMMON /FRSST/ ISIN(MAXPTS), XXX(MAXPTS), YYY(MAXPTS)
INTEGER*4 XXX, YYY
LOGICAL ISIN

DISTANCE VARIABLES (KSW IS TYPE OF SWAP).

INTEGER*4 SAVMAX, DIST, DIST3, DIFF, KSW
Integer i,j,i1,i2,j1,j2,l,newm,jp,k1,k,k2,nsw,n1,n2
Integer ladd, lp, kp
Logical errflag

*** BEGIN ***

IF(KPRINT(2) .EQ. 1) WRITE(1,*), ' THREE'
LISTNO = 0
KSW    = 5
SAVMAX = 0

C MAKE A 3 ARC ROUTE USING THE FIRST THREE "IROUTE" POINTS
C
I = IROUTE(3)
DO L = 1,3
  J = IROUTE(L)
  ILIST(J,1) = I
  ILIST(I,2) = J
  I = J
ENDDO

C SUCCESSIVELY ADD ONE MORE NODE TO THE ROUTE
C
DO NEW = 4,NUMPTS
  NEWM = NEW-1
C THIS SUBROUTINE Chooses THE LEAST EXPENSIVE PLACE TO INSERT NODE
C IROUTE(NEW) IN THE EXISTING ROUTE OF NEW-1 NODES
C CALL CPINST
C
C WE NEED TO EXAMINE EACH ARC IN THE LIST OF
C ARCS THAT HAVE BEEN CHANGED
C INITIALLY THIS LIST 'LIST(LISTNO,-)' WILL BE
C COMPOSED OF THE TWO ARCS ADDED BY CPINST.
C HOWEVER, IF ANY ARCS HAVE BEEN CHANGED BY
C THE 3-OPT, THESE ARCS HAVE BEEN ADDED TO THIS
C LIST IN SUBROUTINE REARR.
C
100 Errflag = .TRUE.
L = 1
DO While ((Errflag).AND.(L.LE.NEW))
  IF (IROUTE(L) .EQ. LIST(LISTNO,1)) THEN
    I1 = LIST(LISTNO,1)
    IF (ILIST(I1,2).EQ.LIST(LISTNO,2)) THEN
      I2 = LIST(LISTNO,2)
      LISTNO = LISTNO -1
  END IF
  ERRflag = .FALSE.
  GO TO 200
ENDIF
ENDIF

IF (ROUTE(L).EQ. LIST(LISTNO,2)) THEN
   I1 = LIST(LISTNO,2)
   IF (LIST(I1,2).EQ.LIST(LISTNO,1)) THEN
      I2 = LIST(LISTNO,1)
      LISTNO = LISTNO - 1

   Errflag = .FALSE.
   ENDIF
ENDIF

L = L + 1
200 ENDDO

IF (Errflag) GO TO 888

C NO SERIOUS ERRORS -- CONTINUE. GET THREE ARCS NAMED
C "I1-I2", "J1-J2", AND "K1-K2".

C

J1 = I1
DO J = 2, NEWM
   JP = J + 1
   J1 = ILIST(J1,2)
   J2 = ILIST(J1,2)
   DIST = D(I1,I2) + D(J1,J2)
   K1 = J1
   DO K = JP, NEW
      KP = K + 1
      K1 = ILIST(K1,2)
      K2 = ILIST(K1,2)
      DIST3 = DIST + D(K1,K2)

   C TEST THE ARRANGEMENT "I1-J1" "I2-K1" "J2-K2"
   C
   DIFF = DIST3 - (D(I1,J1)+D(I2,K1)+D(J2,K2))
   IF (DIFF.GT.SAVMAX) THEN
      SAVMAX = DIFF
      KSW = 1
   ENDFD

C
C TEST THE ARRANGEMENT "I1-J2" "K1-I2" "J1-K2"
C
   DIFF = DIST3 - (D(I1,J2)+D(K1,I2)+D(J1,K2))
   IF (DIFF.GT.SAVMAX) THEN
SAVMAX = DIFF
KSW = 2
ENDIF

C TEST THE ARRANGEMENT "I1-J2" "K1-J1" "I2-K2"
C
DIFF = DIST3 - (D(I1,J2)+D(K1,J1)+D(I2,K2))
IF (DIFF.GT.SAVMAX) THEN
  SAVMAX = DIFF
  KSW = 3
ENDIF

C TEST THE ARRANGEMENT "I1-K1" "J2-I2" "J1-K2"
C
DIFF = DIST3 - (D(I1,K1)+D(J2,I2)+D(J1,K2))
IF (DIFF.GT.SAVMAX) THEN
  SAVMAX = DIFF
  KSW = 4
ENDIF

C REMOVE ARCS —(I1,12),(J1,J2),(K1,K2)— FROM LIST
C
IF (KSW.NE.5) THEN
  IF (LISTNO.NE.0) THEN
    DO NSW = 1, 3
      IF (NSW.EQ.1) THEN
        N1 = I1
        N2 = I2
      ELSE
        IF (NSW.EQ.2) THEN
          N1 = J1
          N2 = J2
        ELSE
          N1 = K1
          N2 = K2
        ENDIF
      ENDIF
    ENDDO
  ELSE
    LADD = 0
    DO L = 1,LISTNO
      LP = L - LADD
    END
IF ((LIST(L,1).EQ.N1.AND.LIST(L,2).EQ.N2).OR. 
& (LIST(L,2).EQ.N1.AND.LIST(L,1).EQ.N2)) THEN 
   LADD = 1 
ELSE 
   LIST(LP,1) = LIST(L,1) 
   LIST(LP,2) = LIST(L,2) 
ENDIF 
ENDDO 
ENDDO 
LISTNO = LISTNO - LADD 
ENDDO 
ENDIF 
C 
C SUBROUTINE REARR MAKES THE ARC SWAPS 
C 
IF (KSW.EQ.1) THEN 
   CALL REARR(I1,J1,I2,K1,J2,K2,KSW) 
ELSE IF (KSW.EQ.2) THEN 
   CALL REARR(I1,J2,K1,I2,J1,K2,KSW) 
ELSE IF (KSW.EQ.3) THEN 
   CALL REARR(I1,J2,K1,J1,I2,K2,KSW) 
ELSE IF (KSW.EQ.4) THEN 
   CALL REARR(I1,K1,J2,I2,J1,K2,KSW) 
ENDIF 
SAVMAX = 0 
KSW = 5 
GO TO 100 
ENDIF 
800 ENDDO 
ENDDO 
IF (LISTNO.GT.0) Go to 100 
C 
C IF THE "LIST" ARRAY IS NOT EMPTY, REDO THIS ITERATION 
C 
ENDDO 
C 
1 FORMAT(10I4) 
C 
888 If (.Not.Errflag) Then 
RETURN 
ELSE
C IF THE PROGRAM CAN'T GET PAST HERE, THERE ARE SERIOUS ERRORS IN
C THE "LIST", "IROUTE", AND/OR "ILIST" ARRAYS.
C
C WRITE(FUNIT,55)
C WRITE(FUNIT,56) L, IROUTE(L), ILIST(I1,2)
C WRITE(*,55)
C 55 FORMAT(3X,' *** ERRORS FOUND IN LISTS')
C WRITE(*,56) L, IROUTE(L), ILIST(I1,2)
C 56 FORMAT( 2'I',',' IROUTE(L) ','I3,' NEXT NODE ','I3)
Print *, 'Numpts ', Numpts
Print *, 'IROUTE ', (I, IROUTE(I), I=1, New)
Print *, 'Listno is ', Listno
C Print *, 'List ', (J, K, LIST(J,K), J=1, New, K=1, New)
STOP
ENDIF
END

SUBROUTINE CPINST
C****************************************************************** CPINST ******************************************************************
C
C THIS ROUTINE CHOOSES WHERE TO INSERT THE NEWTH NODE IN
C THE EXISTING ROUTE. IT REORDERS THE ROUTE WITH
C THIS NODE IN POSITION 2
C
C *** INSERTION COMMON BLOCK ***
C
IMPLICIT NONE
INTEGER*4 MAXPTS, BIGINT
PARAMETER (MAXPTS = 51, BIGINT = 999999)
COMMON /POINTS/ X(MAXPTS), Y(MAXPTS), NUMPTS, MATSIZ
COMMON /DISTS / D(MAXPTS,MAXPTS), DM(MAXPTS,MAXPTS)
COMMON /LISTS / IROUTE(MAXPTS), ILIST(MAXPTS,3)
COMMON /OUTCOM/ FUNIT
COMMON/PRNSW/ KPRINT(IO)
INTEGER*4 X, Y, NUMPTS, MATSIZ
INTEGER*4 D, DM
INTEGER*4 IROUTE, ILIST
INTEGER*4 FUNIT, Kprint
C
185
C *** END OF BLOCK ***
C
C STORAGE FOR LIST ARRAYS FROM "THREEQ"
C
COMMON /LISTTT/ LIST(MAXPTS,2), LISTNO, ISAVD(MAXPTS), NEW
INTEGER*4 LIST, LISTNO, ISAVD, NEW
C
C DISTANCE AND ENDPOINT VARIABLES

INTEGER*4 K, ID, IDIST,newin,i,j,1,lsav,lp
C
C *** BEGIN ***
C
C FIND CHEAPEST PLACE TO INSERT NODE K IN THE EXISTING ROUTE
C
K = IROUTE(NEW)
IDIST = BIGINT * 2
NEWM = NEW - 1
I = IROUTE(NEWM)
DO L = 1,NEWM
   ISAVD(L) = IROUTE(L)
   J = IROUTE(L)
   ID = (DM(I,K) + DM(J,K)) - DM(I,J)
C IF THE DISTANCE DIFFERENCE IS NOT BETTER THAN THE PREVIOUS BEST,
C SKIP AND START ANOTHER TEST.
C
IF (ID.LT.IDIST) THEN
   LSAV = L
   ILIST(K,1) = I
   ILIST(K,2) = J
   IDIST = ID
ENDIF
   I = J
ENDDO
C
C INSERT K AFTER IROUTE(LSAV-1)=I AND MAKE NODE I THE FIRST NODE
C ON THE ROUTE.
C
I = ILIST(K,1)
J = ILIST(K,2)
ILIST(I,2) = K
ILIST(J,1) = K
IROUTE(1) = I
IROUTE(2) = K
DO L = 3, NEW
    LP = LSAV + L - 3
    IROUTE(L) = ISAVD(LP)
    IF (LP.EQ.NEWM) LSAV = LSAV-NEWM
ENDDO
LIST(1,1) = I
LIST(1,2) = K
LIST(2,1) = K
LIST(2,2) = J
LISTNO = 2
RETURN
END

SUBROUTINE REARR(I1, I2, J1, J2, K1, K2, KSW)
C****************************** REARR ****************************
C
THIS ROUTINE REARRANGES THE SEQUENCE OF NODES IN IROUTE
C
C *** INSERTION COMMON BLOCK ***
C
IMPLICIT NONE
INTEGER*4 MAXPTS, BIGINT
PARAMETER (MAXPTS = 51, BIGINT = 999999)
COMMON /POINTS/ X(MAXPTS), Y(MAXPTS), NUMPTS, MATSIZ
COMMON /DISTS / D(MAXPTS,MAXPTS), DM(MAXPTS,MAXPTS)
COMMON /LISTS / IROUTE(MAXPTS), ILIST(MAXPTS,3)
COMMON /OUTCOM/ FUNIT
COMMON/PRNSW/ KPRINT(10)
INTEGER*4 X, Y, NUMPTS, MATSIZ
INTEGER*4 D, DM
INTEGER*4 IROUTE, ILIST
INTEGER*4 FUNIT, Kprint
C
C *** END OF BLOCK ***
C
C STORAGE FOR LIST ARRAYS FROM "THREEO"
COMMON /LISTTT/ LIST(MAXPTS,2), LISTNO, ISAVD(MAXPTS), NEW
INTEGER*4 LIST, LISTNO, ISAVD, NEW

Integer i1,i2,j1,j2,k1,k2,ksw,i,ilsav,j,jlsav,l

C *** BEGIN ***
C
ILIST(I1,2) = I2
IF ((KSW.NE.2).AND.(KSW.NE.3)) THEN
C
C REORDER ROUTE FROM I2 TO J1
C
I = I2
ILSAV = ILIST(I,1)
ILIST(I,1) = I1
ILIST(I,2) = ILSAV
DO WHILE (I.NE.J1)
  I = ILSAV
  ILSAV = ILIST(I,1)
  ILIST(I,1) = ILIST(I,2)
  ILIST(I,2) = ILSAV
ENDDO
ENDIF
ILIST(J1,2) = J2
ILIST(I2,1) = I1
IF ((KSW.NE.2).AND.(KSW.NE.4)) THEN
C
C REORDER ROUTE FROM J2 TO K1
C
J = J2
JLSAV = ILIST(J,1)
ILIST(J,1) = J1
ILIST(J,2) = JLSAV
DO WHILE (J.NE.K1)
  J = JLSAV
  JLSAV = ILIST(J,1)
  ILIST(J,1) = ILIST(J,2)
  ILIST(J,2) = JLSAV
ENDDO
ENDIF
ILIST(K1,2) = K2
ILIST(J2,1) = J1
ILIST(K2,1) = K1
I = IROUTE(1)
DO L = 2,NEW
   IROUTE(L) = ILIST(I,2)
   I = IROUTE(L)
ENDDO
IF (LISTNO.EQ.-1) RETURN

C ADD THE NEW ARCS --(I1,I2),(J1,J2),(K1,K2)-- TO THE
C LIST OF ARCS THAT MUST BE EXAMINED
C
LISTNO = LISTNO + 1
LIST(LISTNO,1) = K1
LIST(LISTNO,2) = K2
LISTNO = LISTNO + 1
LIST(LISTNO,1) = J1
LIST(LISTNO,2) = J2
LISTNO = LISTNO + 1
LIST(LISTNO,1) = I1
LIST(LISTNO,2) = I2
RETURN
END
Appendix C - Test Cases
This is the file format for all of the test case files included within this appendix. The twelve test cases were 12 cities and 3 trucks (213 rts), 12 cities and 2 trucks (1252 rts), 15 cities and 5 trucks (88 rts), 15 cities and 3 trucks (81 rts), 19 cities and 2 trucks (8,547 rts), 20 cities and 6 trucks (25 rts), 20 cities and 4 trucks (156 rts), 25 cities and 8 trucks (91 rts), 29 cities and 3 trucks (10,740 rts), 32 cities and 4 trucks (65,657 rts), 32 cities and 3 trucks (6,337 rts), and 50 cities and 6 trucks (92,122 rts). File Format:

Cities+Depot MaxWeight NumberofVehicles StopTime TimeLimit Gap

City 1’s X-Coordinate, Y-Coordinate, WeightDemand
...

City Cities’s X-Coordinate, Y-Coordinate, WeightDemand
Depot’s X-Coordinate, Y-Coordinate, WeightDemand

Real dual value associated with City 1
...

Real dual value associated with City Cities
Real dual value associated with the depot
Case with 12 Cities and 3 Trucks

13 10000 3 20 550 900000.
65 248 1260
22 255 629
50 249 250
205 254 2267
275 34 447
269 262 1847
293 269 1437
333 212 3720
304 202 1115
286 207 273
288 191 5494
295 235 1944
250 200 0 DEPOT
271.66218566895
127.09883880615
70.481285095215
89.225608825684
286.35804748535
45.320541381836
58.351913452148
96.946693420410
38.868545532227
1.3187561035156
60.488441467285
50.324760437012
0.
Case with 12 Cities and 2 Trucks

13  15000  2  20  750  900000.
 65  248  1260
 22  255  629
 50  249  250
 205  254  2267
 275  34  447
 269  262  1847
 293  269  1437
 333  212  3720
 304  202  1115
 286  207  273
 288  191  5494
 295  235  1944
 250  200  0  DEPOT
349.26082611084
102.011286417643
32.426816304524
15.823605855306
287.58507029215
27.806889851889
45.674489339193
102.840082804362
0.45001220703136
16.403989156087
50.694875081380
25.359001159667
0.
Case with 15 Cities and 5 Trucks

16 55 5 1 1000 15000.
37 52 7 1
49 49 30 2
52 64 16 3
20 26 9 4
40 30 21 5
21 47 15 6
17 63 19 7
31 62 23 8
52 33 11 9
51 21 5 10
42 41 19 11
31 32 29 12
5 25 23 13
12 42 21 14
36 16 10 15
30 40 0 DEPOT
3.5316314697266
29.403366088867
39.338031768799
11.333984375000
18.806144714355
17.939186096191
36.673114776611
28.655319213867
18.260978698730
27.818546295166
17.142532348633
15.468154907227
44.419834136963
15.830039978027
17.067764282227
-2.8503608703613

194
Case with 15 Cities and 3 Trucks

<p>| | | | | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>90</td>
<td>3</td>
<td>1</td>
<td>1000</td>
<td>8000.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>52</td>
<td>7</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>49</td>
<td>30</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>64</td>
<td>16</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>26</td>
<td>9</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>30</td>
<td>21</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>47</td>
<td>15</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>63</td>
<td>19</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>62</td>
<td>23</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>33</td>
<td>11</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>21</td>
<td>5</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>41</td>
<td>19</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>32</td>
<td>29</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>25</td>
<td>23</td>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>42</td>
<td>21</td>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>16</td>
<td>10</td>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>40</td>
<td>0</td>
<td>DEPOT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

9.8292439778647
23.723101298014
32.731608072917
11.333976745606
13.847579956055
12.149312337240
31.106012980143
20.965606689453
17.773648579916
17.824012756348
14.321314493815
15.272453308106
41.253534952800
14.916196187337
26.042793273926
-10.655097961427
Case with 19 Cities and 2 Trucks

<p>| | | | | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>38000</td>
<td>2</td>
<td>20</td>
<td>1000</td>
<td>900000.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>65</td>
<td>248</td>
<td>1260</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>255</td>
<td>629</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>249</td>
<td>250</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>205</td>
<td>254</td>
<td>2267</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>275</td>
<td>34</td>
<td>447</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>269</td>
<td>262</td>
<td>1847</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>293</td>
<td>269</td>
<td>1437</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>333</td>
<td>212</td>
<td>3720</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>304</td>
<td>202</td>
<td>1115</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>286</td>
<td>207</td>
<td>273</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>288</td>
<td>191</td>
<td>5494</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>295</td>
<td>235</td>
<td>1944</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>467</td>
<td>67</td>
<td>713</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>484</td>
<td>179</td>
<td>1500</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>447</td>
<td>189</td>
<td>3585</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>215</td>
<td>204</td>
<td>140</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>313</td>
<td>382</td>
<td>25705</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>267</td>
<td>316</td>
<td>479</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>391</td>
<td>196</td>
<td>17456</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>250</td>
<td>200</td>
<td>0</td>
<td>DEPOT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>20.675065260667</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>113.42210740309</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>334.54757690430</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>46.829797598032</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>211.18575110802</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>56.500696035532</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>53.377099844125</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21.559220534105</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>52.264184805063</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29.424149733323</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9.3339315561150</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>57.722314687876</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>254.92920508752</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>93.010600163387</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.1255575326769</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20.932999830980</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>144.66656142015</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>45.812577467698</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Case with 20 Cities and 6 Trucks

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>58</td>
<td>6</td>
<td>1</td>
<td>1000</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>41</td>
<td>19</td>
<td>1</td>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>32</td>
<td>29</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>25</td>
<td>23</td>
<td>13</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>42</td>
<td>21</td>
<td>14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>16</td>
<td>10</td>
<td>15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>41</td>
<td>15</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>23</td>
<td>3</td>
<td>17</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>33</td>
<td>41</td>
<td>18</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>13</td>
<td>9</td>
<td>19</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>58</td>
<td>28</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>62</td>
<td>42</td>
<td>8</td>
<td>21</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>57</td>
<td>8</td>
<td>22</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>57</td>
<td>16</td>
<td>23</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>52</td>
<td>10</td>
<td>24</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>38</td>
<td>28</td>
<td>25</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>68</td>
<td>7</td>
<td>26</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>48</td>
<td>15</td>
<td>27</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>67</td>
<td>14</td>
<td>28</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>48</td>
<td>6</td>
<td>29</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>27</td>
<td>19</td>
<td>30</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>40</td>
<td>0</td>
<td>DEPOT</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

20.257507324219
24.185836791992
34.608430862427
16.814615249634
24.339595794678
16.119007110596
10.1986389160156
34.706619262696
33.890224456787
44.704353332520
17.126998901367
10.631626129150
25.256786346436
12.536298751831
40.912893295288
23.964572906494
18.034461975098
27.503013610840
13.959987640381
41.894145965576
-10.1268234252935
Case with 20 Cities and 4 Trucks

21  85  4  1  1000  9000.
  42  41  19  1
  31  32  29  2
  5   25  23  3
  12  42  21  4
  36  16  10  5
  52  41  15  6
  27  23  3   7
  17  33  41  8
  13  13  9   9
  57  58  28 10
  62  42  8   11
  42  57  8   12
  16  57  16 13
   8  52  10 14
   7  38  28 15
  27  68  7   16
  30  48  15 17
  43  67  14 18
  58  48  6   19
  58  27  19 20
  30  40  0   DEPOT

9.6658134460438
13.016071319578
20.815959930418
  6.8746604919418
18.502895355224
  9.1226119995108
  5.0367622375484
13.510864257810
23.645542144775
24.680225372312
14.838077545166
  6.9363117218010
16.454828262328
16.329492568969
15.116016387938
20.269258499144
  7.0995121002182
18.829200744628
4.8839244842525
25.922527313231
15.067432403571
Case with 25 Cities and 8 Trucks

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>26</td>
<td>58</td>
<td>8</td>
<td>1</td>
<td>1000 5000</td>
</tr>
<tr>
<td>21</td>
<td>47</td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>63</td>
<td>19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>62</td>
<td>23</td>
<td></td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>33</td>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>21</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>41</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>32</td>
<td>23</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>25</td>
<td>23</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>42</td>
<td>21</td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>16</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>41</td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>23</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>33</td>
<td>41</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>13</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>58</td>
<td>28</td>
<td></td>
<td></td>
</tr>
<tr>
<td>62</td>
<td>42</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>57</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>57</td>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>52</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>38</td>
<td>28</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>68</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>48</td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>67</td>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>48</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>27</td>
<td>19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>40</td>
<td>0</td>
<td>DEPOT</td>
<td></td>
</tr>
<tr>
<td>11.197153091431</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25.128652572634</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24.427316665654</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17.204787254345</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10.743563652028</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18.192005157471</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16.274072647104</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>34.656839370731</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16.863023757938</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26.914163589479</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14.216864585874</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.6730060577380</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29.679203033456</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28.341766357426</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>41.267613410958</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18.051892280581</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11.245053291322</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14.757444381717</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20.517787933352</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30.887218475348</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14.688415527348</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13.084611892700</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25.912337303161</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.3967103958133</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24.881469726571</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-0.14955711365650</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Case with 29 Cities and 3 Trucks

<table>
<thead>
<tr>
<th>City</th>
<th>Distance</th>
<th>Demand</th>
<th>Quantity</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>4500</td>
<td>3</td>
<td>1</td>
<td>200</td>
</tr>
<tr>
<td>55000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>218</td>
<td>382</td>
<td>300</td>
<td></td>
<td></td>
</tr>
<tr>
<td>218</td>
<td>358</td>
<td>3100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>201</td>
<td>370</td>
<td>125</td>
<td></td>
<td></td>
</tr>
<tr>
<td>214</td>
<td>371</td>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>224</td>
<td>370</td>
<td>200</td>
<td></td>
<td></td>
</tr>
<tr>
<td>210</td>
<td>382</td>
<td>150</td>
<td></td>
<td></td>
</tr>
<tr>
<td>104</td>
<td>354</td>
<td>150</td>
<td></td>
<td></td>
</tr>
<tr>
<td>126</td>
<td>338</td>
<td>450</td>
<td></td>
<td></td>
</tr>
<tr>
<td>119</td>
<td>340</td>
<td>300</td>
<td></td>
<td></td>
</tr>
<tr>
<td>129</td>
<td>349</td>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>126</td>
<td>347</td>
<td>950</td>
<td></td>
<td></td>
</tr>
<tr>
<td>125</td>
<td>346</td>
<td>125</td>
<td></td>
<td></td>
</tr>
<tr>
<td>116</td>
<td>355</td>
<td>150</td>
<td></td>
<td></td>
</tr>
<tr>
<td>126</td>
<td>335</td>
<td>150</td>
<td></td>
<td></td>
</tr>
<tr>
<td>125</td>
<td>355</td>
<td>550</td>
<td></td>
<td></td>
</tr>
<tr>
<td>119</td>
<td>357</td>
<td>150</td>
<td></td>
<td></td>
</tr>
<tr>
<td>115</td>
<td>341</td>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>153</td>
<td>351</td>
<td>150</td>
<td></td>
<td></td>
</tr>
<tr>
<td>175</td>
<td>363</td>
<td>400</td>
<td></td>
<td></td>
</tr>
<tr>
<td>180</td>
<td>360</td>
<td>300</td>
<td></td>
<td></td>
</tr>
<tr>
<td>159</td>
<td>331</td>
<td>1500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>188</td>
<td>357</td>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>152</td>
<td>349</td>
<td>300</td>
<td></td>
<td></td>
</tr>
<tr>
<td>215</td>
<td>389</td>
<td>500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>212</td>
<td>394</td>
<td>800</td>
<td></td>
<td></td>
</tr>
<tr>
<td>188</td>
<td>393</td>
<td>300</td>
<td></td>
<td></td>
</tr>
<tr>
<td>207</td>
<td>406</td>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>184</td>
<td>410</td>
<td>150</td>
<td></td>
<td></td>
</tr>
<tr>
<td>207</td>
<td>392</td>
<td>1000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>162</td>
<td>354</td>
<td>0</td>
<td>DEPOT</td>
<td></td>
</tr>
<tr>
<td>10.718106951032</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>73.011051722938</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.1064365931922</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13.518410818918</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26.561381748744</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10.2654440743581</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24.060659996549</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13.777417405207</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11.643536789974</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Case with 32 Cities and 4 Trucks

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>33</td>
<td>8000</td>
<td>4</td>
<td>10</td>
<td>340</td>
<td>23000.</td>
</tr>
<tr>
<td>298</td>
<td>427</td>
<td>700</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>309</td>
<td>445</td>
<td>400</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>307</td>
<td>464</td>
<td>400</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>336</td>
<td>475</td>
<td>1200</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>320</td>
<td>439</td>
<td>40</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>321</td>
<td>437</td>
<td>80</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>322</td>
<td>437</td>
<td>2000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>323</td>
<td>433</td>
<td>900</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>324</td>
<td>433</td>
<td>600</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>323</td>
<td>429</td>
<td>750</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>314</td>
<td>435</td>
<td>1500</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>311</td>
<td>442</td>
<td>150</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>304</td>
<td>427</td>
<td>250</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>293</td>
<td>421</td>
<td>1600</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>296</td>
<td>418</td>
<td>450</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>261</td>
<td>384</td>
<td>700</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>297</td>
<td>410</td>
<td>550</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>315</td>
<td>407</td>
<td>650</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>314</td>
<td>406</td>
<td>200</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>321</td>
<td>391</td>
<td>400</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>321</td>
<td>398</td>
<td>300</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>314</td>
<td>394</td>
<td>1300</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>313</td>
<td>378</td>
<td>700</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>304</td>
<td>382</td>
<td>750</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>295</td>
<td>402</td>
<td>1400</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>283</td>
<td>406</td>
<td>4000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>279</td>
<td>399</td>
<td>600</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>271</td>
<td>401</td>
<td>1000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>264</td>
<td>414</td>
<td>500</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>277</td>
<td>439</td>
<td>2500</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>290</td>
<td>434</td>
<td>1700</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>319</td>
<td>433</td>
<td>1100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>292</td>
<td>495</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.129382056912</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.66510009765610</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.0475921630858</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27.871253967285</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.7269592285159</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.81912231445286</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2.6786270141599
2.3633804321288
2.4240188598632
7.1433334350585
4.1998672485350
0.55508422851571
6.2570127501104
7.7733119212801
3.3621700314708
43.555377041337
8.0547145091708
8.1458093684955
8.5550686355927
16.637743650561
16.481249509937
21.173940282669
24.795726748279
28.411068268935
9.8559534114640
24.057133751194
8.9802766785992
28.715191806320
10.0226808534041
16.538634641327
8.0826075755757
2.5126800537110
124.03720855713
Case with 32 Cities and 3 Trucks

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>33</td>
<td>38000</td>
<td>3</td>
<td>20</td>
<td>1000</td>
<td>50000.</td>
<td></td>
</tr>
<tr>
<td>65</td>
<td>248</td>
<td>1260</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>255</td>
<td>629</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>249</td>
<td>250</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>205</td>
<td>254</td>
<td>2267</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>275</td>
<td>34</td>
<td>447</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>269</td>
<td>262</td>
<td>1847</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>293</td>
<td>269</td>
<td>1437</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>333</td>
<td>212</td>
<td>3720</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>304</td>
<td>202</td>
<td>1115</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>286</td>
<td>207</td>
<td>273</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>288</td>
<td>191</td>
<td>5494</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>295</td>
<td>235</td>
<td>1944</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>467</td>
<td>67</td>
<td>713</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>484</td>
<td>179</td>
<td>1500</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>447</td>
<td>189</td>
<td>3585</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>215</td>
<td>204</td>
<td>140</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>313</td>
<td>382</td>
<td>25705</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>267</td>
<td>316</td>
<td>479</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>391</td>
<td>196</td>
<td>17456</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>399</td>
<td>122</td>
<td>1143</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>363</td>
<td>187</td>
<td>1919</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>355</td>
<td>236</td>
<td>826</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>378</td>
<td>203</td>
<td>3264</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>458</td>
<td>218</td>
<td>1570</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>383</td>
<td>181</td>
<td>2215</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>240</td>
<td>326</td>
<td>1239</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>273</td>
<td>349</td>
<td>580</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>278</td>
<td>374</td>
<td>5000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>352</td>
<td>271</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>324</td>
<td>295</td>
<td>201</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>249</td>
<td>250</td>
<td>6747</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>260</td>
<td>3500</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>250</td>
<td>200</td>
<td>0</td>
<td>DEPOT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>44.841000181256</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>298.15262056601</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15.191567160867</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>35.202356666025</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>220.99118627683</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9.6061817515978</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

208
Case with 50 Cities and 6 Trucks

<p>| | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>51</td>
<td>160</td>
<td>6</td>
<td>10</td>
<td>200</td>
<td>1700</td>
<td>.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>52</td>
<td>7</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>49</td>
<td>30</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>64</td>
<td>16</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>26</td>
<td>9</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>30</td>
<td>21</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>47</td>
<td>15</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>63</td>
<td>19</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>62</td>
<td>23</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>33</td>
<td>11</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>21</td>
<td>5</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>41</td>
<td>19</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>32</td>
<td>29</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>25</td>
<td>23</td>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>42</td>
<td>21</td>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>16</td>
<td>10</td>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>41</td>
<td>15</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>23</td>
<td>3</td>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>33</td>
<td>41</td>
<td>18</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>13</td>
<td>9</td>
<td>19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>58</td>
<td>28</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>62</td>
<td>42</td>
<td>8</td>
<td>21</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>57</td>
<td>8</td>
<td>22</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>57</td>
<td>16</td>
<td>23</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>52</td>
<td>10</td>
<td>24</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>38</td>
<td>28</td>
<td>25</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>68</td>
<td>7</td>
<td>26</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>48</td>
<td>15</td>
<td>27</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>67</td>
<td>14</td>
<td>28</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>48</td>
<td>6</td>
<td>29</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>27</td>
<td>19</td>
<td>30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>69</td>
<td>11</td>
<td>31</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>46</td>
<td>12</td>
<td>32</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>10</td>
<td>23</td>
<td>33</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>61</td>
<td>33</td>
<td>26</td>
<td>34</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>62</td>
<td>63</td>
<td>17</td>
<td>35</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>63</td>
<td>69</td>
<td>6</td>
<td>36</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>22</td>
<td>9</td>
<td>37</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>35</td>
<td>15</td>
<td>38</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>59</td>
<td>15</td>
<td>14</td>
<td>39</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>40</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>17</td>
<td>27</td>
<td>41</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>10</td>
<td>13</td>
<td>42</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>64</td>
<td>11</td>
<td>43</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>15</td>
<td>16</td>
<td>44</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>10</td>
<td>10</td>
<td>45</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>39</td>
<td>5</td>
<td>46</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>32</td>
<td>25</td>
<td>47</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>55</td>
<td>17</td>
<td>48</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>28</td>
<td>18</td>
<td>49</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>37</td>
<td>10</td>
<td>50</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>40</td>
<td>0</td>
<td>DEPOT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5.9291486114212
8.8005597128838
11.643516540527
9.4896004871177
6.453941057610
6.7910370601617
6.1271693346977
7.9117957352906
9.6662435786918
8.3623018009469
4.6389753130845
4.956289617539
17.116301828098
10.735640026470
6.5618813709068
9.5970365313462
7.534096395668
8.8146807698594
8.3408495931016
12.95069677739
12.661527803319
8.6862970990701
11.260763406616
12.986684099998
13.84964852905
18.862065888714
3.9953522457085
7.5962193172830
9.0642920388665
Appendix D - Helpful PVM Hints
Helpful PVM Hints

Once again, PVM was used mainly for its portability of application. However, PVM is quite interesting to learn. Often there may be problems with printing or memory. Sometimes the error messages may be quite cryptic. A set of codes could work fine for some cases and crash and/or give strange error messages for other cases.

This researcher would suggest that a better manual be written for PVM which talks about print sinks in English rather than some engineering language. One should understand how to force information to print into the PVM sink in the event the code hangs or crashes as one is debugging. Print statements are very useful in debugging. Anyway, in case the reader is wondering, one can get into the PVM console by typing `pvm` and then type `reset` at the `pvm>` prompt. The messages which may not have been written into the `/tmp/pvml.userid` before PVM had been reset should be there after the reset. One could also flush the print buffer after every print command, or as often as necessary. Otherwise, the buffer will wait until there are many lines in the buffer to print anything. If the buffer has not been flushed and the processes have died or exited, one must reset `pvm` to flush whatever was in the buffer out to where the statements should have printed. One can get out of the `pvm` console by typing `quit` at the `pvm>` prompt.

One has to start the group server manually if the group server is needed for the codes. Otherwise, the code will crash with the first group call.
One should remember how two-dimensional arrays are stored in the programming language of C or FORTRAN and make sure that messages get sent correctly. Also, one must remember that sending the starting location and number of filled positions in a two-dimensional array MAY not effectively send the information. Assume the array is dimensioned as M X M, but only N X N spaces have been filled. (N is less than M). Sending the starting position and the number N X N will send M positions in the first row or column (depending on whether one is using C or FORTRAN), M positions in the second row or column, and so on until the number N X N has been reached. In actuality, this was not the intended message. One could either send all M X M positions in the matrix or pack and send just the information in the N X N portion of the matrix.

Remember than smaller amounts of large messages tends to be better than massive amounts of small message communications in PVM. However, the daemon must save a message in its buffer until a process can receive the message. Once the daemon has expanded to hold a message of a particular size, the daemon’s buffer remains at least that large. Therefore, if memory becomes a problem, one should try smaller messages. This can be a process of trial and error.

This researcher had problems running PVM jobs over certain machines without one PVM process dying (from not being able to allocated enough memory) or the jobs being killed (because they were running too long on certain machines). One is well advised to know the amount of real and CPU time one is allowed to use on
a machine, how often the processes are erased from machines, around what time
during the day or night such a “clean up” may occur, and what procedures are
required for reserving time on certain machines.

One must make sure that one doesn’t allow too much randomness to enter
into one’s timings. Ideally, all machines must be configured in the same manner
and be dedicated to the job being run. To maintain consistency, other researchers
should not be running code over the nodes, or processors, over which one’s code
is being timed.

One definitely should not run more than one PVM application per userid
which includes one of the same machines. This is due to conflicting writes into
the /tmp/pvml.userid on that one machine when more than one PVM application
is run.

When running PVM over a homogeneous network, use PVMDaRaw when
sending messages. Otherwise, use PvmDataDefault when sending messages.

`pvm.psend (pvmlp send)` and `pvm.precv (pvmlprecv)` are quicker forms of com-
munication than packing and sending and receiving and unpacking.

Usually, the master process should handle any input the children tasks need
as well as any output from the children which should be printed to file.

And, the most important PVM hint of all is these can be dangerous waters-
“Don’t go swimming alone!” Try to make sure that there is some friend or ac-
quaintance around who will be able to answer some questions. Also, one should
know if there is on-line help available for a particular machine and where to ask for that help.
Appendix E - Helpful SP2 Hints
SP2 Hints

One will want to know the policies and procedures regarding the SP2 at whichever location one chooses to work on the SP2. Much information is available on-line. Information on the machine's hardware, available software, programming hints, etc. all tend to be available on-line. There are also LoadLeveler templates which often prove to be extremely useful. Beginner's guides are also available on-line. Maui High Performance Computing Center has a 400 node SP2. Cornell has a 512 node SP2. These and other useful World Wide Web addresses are included in the following appendix.

One helpful hint for using PVM on the SP2 is to use PVMDataRaw when sending messages since the processors are all RS6000 processors. The data does not have to be encoded, sent, and then decoded. This saves time. Time is also saved when using PVM over the switch on the SP2. There may be template files available which allow one to use PVM over the switch with the Loadleveler. If the PVM application will not be run over batch nodes, one can get into the pvm console by typing pvm -n switchnodename. The switchnodename is the name of the switch of the node from which one is calling PVM. The naming convention for switches differs from one location to another. When one is in PVM, one should then add the switch names corresponding to the nodes over which the application should run.

One should call setopt with PvmRoute and PvmRouteDirect for an ap-
plication over 32 nodes or less. This allows the processes to communicate directly rather than through the daemon. Applications with over 32 nodes must communicate through the daemon.

If memory becomes a problem, one should print to file whenever one can do so. This might help eliminate the problem with memory.

And, of course, knowing where one can go for help before a problem arises is always quite useful.
Appendix F - World Wide Web Addresses
Good World Wide Web Site Addresses

Vehicle Routing URLs:

http://promet4.cineca.it/index.html
http://www.aiai.ed.ac.uk/~timd/vehicles/vrp.html
http://borneo.gmd.de/~andy/Benchmarks.html
http://frdsa.utc.sk/~sim/

SP2 URLs:

http://www.tc.cornell.edu
http://www.mhpcc.edu
http://www.mcs.anl.gov/Projects/sp/
http://spud-web.tc.cornell.edu/HyperNews/get/SPUserGroup.html
http://ibm.tc.cornell.edu/ibm/pps/
http://lscftp.kgn.ibm.com/pps/
http://csep1.phy.ornl.gov/SP2_guide/SP2_guide.html

PVM URLs:

http://netlib2.cs.utk.edu
http://www.epm.ornl.gov/pvm/pvm_home.html
http://www-jics.cs.utk.edu/PVM/pvm.html
http://csep1.phy.ornl.gov/pvm_guide/pvm_guide.html

Parallel Computing URLs:

http://www-jics.cs.utk.edu/
http://csep1.phy.ornl.gov/csep.html
http://nii.nist.gov
Vita

Stephanie Marie Wolf was born at St. Joseph's hospital in Murphysboro, IL. She was placed with a Catholic adoption agency in Belleville, IL. She was adopted by a very loving couple and within two years became a big sister to their miracle baby boy.

Stephanie attended grade school and high school in Freeburg, IL. where she was involved in many activities and held many honors. Stephanie graduated high school in May, 1988 with a 5.10/5.0 G.P.A., and was salutatorian of her class.

She received a James Millikin Scholarship, and attended Millikin University in Decatur, IL., where she once again was very involved in several activities and held many honors. Being in the honors program gave her the opportunity to research a topic of little known previous research at the time - handwritten character recognition (using back-propagation neural networks). She graduated Millikin magna cum laude with a double-major in mathematics and computers and a G.P.A. of 3.85/4.0. She worked for about a year and decided to go back to graduate school at UTK for her master's degree in Computer Science. She started August 1993. The master's degree was achieved in May 1996. The topic was Parallelizing code for the Set-Partitioning Method of Solving the Vehicle Routing Problem.

Stephanie has moved to Marietta, GA and is searching for a job in technical writing, computer training, or computer consultant. She also plans to become actively involved in the community.