Approximating Shortest Paths in Large Networks

David Randolph Lorek

A Thesis Submitted to the
University of North Carolina Wilmington in Partial Fulfillment
Of the Requirements for the Degree of
Master of Science

Department of Mathematics and Statistics

University of North Carolina Wilmington

2005

Approved by

Advisory Committee

_____ _____

_____

Chair

Accepted by

_____

Dean, Graduate School

This thesis has been prepared in the style and format

Consistent with the journal

American Mathematical Monthly.

TABLE OF CONTENTS

ABSTRACT

In the classroom students are introduced to shortest route calculation using small datasets (those that can be hand-drawn.) For demonstrating the application of an algorithm a small dataset is typically sufficient. However, real-world applications of shortest path calculations seem to be useful only when applied to large datasets. This paper presents research on a computer based implementation of a modified Dijkstra algorithm as applied to large datasets including tens of thousands of arcs. In an attempt to improve the performance of calculating paths two heuristics are also examined. The intuition behind the heuristics is to remove the arcs that will likely not be traversed by the optimal path from the set of arcs that can possibly be traversed by the optimal path. By reducing this number less labeling is required, resulting in fewer CPU cycles being used to generate a route. This paper compares the results of the optimal against those of the two heuristics.

ACKNOWLEDGMENTS

I would like to thank the faculty of UNCW's Mathematics and Statistics department that assisted in my research and studies. I would also like to thank Dr. Gene A. Tagliarini of UNCW's Computer Science department for his instruction. A special thank you goes to my thesis defense committee members: Dr. John K. Karlof, Dr. David A. Rolls, and especially to my advisor Dr. Yaw O. Chang, whose instruction inspired, and whose guidance enabled, the creation of this thesis.

The data used by this study was compiled by the United States Census Bureau and downloaded from http://www.ESRI.com.

DEDICATION

This paper would not have been possible without the encouragement and support of my wife Kirsten.

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

Mathematical programming focuses on problems where an objective function is to be optimized relative to one or more constraints. [2, p.1] One subset of these problems known as linear programming problems can be represented by the following standard form:

$$\begin{aligned} &\text{Minimize} && c^T x = z \\ &\text{subject to} && Ax = b, \qquad A : m \times n, \\ & && x \geq 0. \end{aligned}$$

The vector x represents what are known as the decision variables of the objective function. The vector c represents the coefficients associated with the decision variables. The matrix A represents a system of coefficients applied to the x that form a set of constraints for the system.

When one of the constraints on a linear program is that the decision variables assume discrete or non-fractional values the problem is classified as an integer programming problem. Network flow problems represent an important class of integer programming problems. This subset of problems can be used to address many real world issues involving transportation, resource allocation, IP (Internet Protocol) routing and more. [4, p.1] In this paper, we are interested in a specific network flow problem known as the shortest path or shortest route problem. We will introduce the formulation of the problem and discuss the difficulties associated with its solution using the famous Dijkstra algorithm with real world data.

Definition Let N be a finite, nonempty set of nodes and let A be a set of unordered pairs (or arcs) of distinct nodes in N. A graph $G = (N, A)$ is a convention used to represent the node set N associated with the arcs that connect its nodes to each other. [4, p.73]

Definition Let $G = (N, A)$ be a graph and let $n, m \in N$. A path exists between n and m if a sequence of arcs in A can be found that connects n to m either directly or through a number of intermediate arcs in N and no node is repeated. [4, p.74]

Example Figure 1 depicts a graph with four nodes: 1, 2, 3, and 4. The graph contains three arcs: (1,2), (2,3), and (3,4). While there are obviously no arcs connecting node 1 directly to node 4, the sequence of these three arcs does generate a path from node 1 to node 4.

Figure 1: Four node network with three arcs.

The shortest route problem is concerned with finding the best, or minimum-cost, path between two nodes in a graph. Cost can describe the distance or time required to traverse a path. We are concerned with finding the path between two nodes that traverses the minimum distance possible. Additional constraints are required to ensure success in determining a solution. One such constraint is that the graph be connected.

Definition A graph $G = (N, A)$ is connected if for any two nodes $n, m \in N$ at least one path can be found from n to m using the arcs in A. [4, p.75]

To determine the best path between two nodes we need a way of representing what it means for one path to be better than another. The idea of a network provides this ability.

Definition A network is a graph associated with one or more functions that map the arcs or nodes in the graph to some quantifiable values. These function mappings enable the defining of constraints on the system. [4, p.75]

In the case of the shortest route problem, a function exists that associates each arc with the cost associated with traversing that arc. If i and j are nodes connected by an arc (i, j), the cost of traversing the arc might be represented as $c(i, j)$. In the path example above, the cost of the path from node 1 to node 4 is equal to the sum of the costs of its component arcs, or

$$c(path) = c(1, 2) + c(2, 3) + c(3, 4).$$

With the ability to associate costs to arcs we can define the optimal path as follows.

Definition Let P_i be a path between two nodes a and b in a network and suppose n distinct paths can be found. P_o is the optimal path if $c(P_o) \leq c(P_i)$, $i = 1, \ldots, n$.

In some network applications, upper and lower bounds are associated with arcs to restrict the flows through the arcs. For instance, in a network flow problem related to city streets, the maximum flow over a road segment is proportional to the speed limit associated with the segment as well as the number of distinct lanes contained in the segment. It is easy to imagine how adding lanes can increase the maximum flow through a segment.

For the shortest route problem, we consider a network in which the flows associated with each arc have lower bounds of 0 and upper bounds of 1. If a particular arc is being traversed by the solution its flow is 1 and if the arc is not being traversed then its flow is 0. Because of the special structure of the shortest route problem, the optimal solutions produced by the simplex method will automatically satisfy this integrality requirement.

However, some difficulties are associated with the simplex method when solving the shortest route problem. One difficulty, common to most integer programming problems, is a high degree of degeneracy.

Definition A solution to an objective function is said to be degenerate when at least one of its decision variables is at one of its bounds.

The Simplex method can enter a phenomenon known as cycling when solutions become degenerate.

Definition An algorithm is cycling when it repeats the same sequence of iterations indefinitely.

Cycling can occur in these situations because a pivot involving a degenerate variable (one at one of its bounds) tends to not improve the solution. When a pivot operation does not improve the solution it is possible that the reverse operation will seem desirable to the algorithm, re-introducing the degenerate variable into the solution. At this point the operations may repeat or cycle.

Fortunately, the difficulties associated with the shortest route problem have already been solved by Dijkstra, who introduced his solution in a paper entitled *A Note on Two Problems in Connexion with Graphs* in 1959[1, p.269-271]. Dijkstra's algorithm is considered to be a greedy algorithm[5].

Definition An algorithm that always takes the best immediate, or local, solution while finding an answer is called a greedy algorithm[6].

Algorithms of this class are typically characterized as efficient but faulty. This is detailed in the following example demonstrating a typical greedy algorithm.

Example Let us consider the data in Table 1. Our objective is to maximize the values that we record for each observation, where we can only record one value per observation and each variable must be recorded once and only once. Consider the following possibilities for greedy algorithms.

Option 1 - Iterate through the observations

- For Observation1, record the value of Variable1.
- For Observation2, record the value of Variable2, since Variable1 has already been used.
- For Observation3, record the value of Variable3.

The objective value would be $9 + 2 + 3 = 14$.

Table 1: Greedy algorithm sample data

	Variable1	Variable2	Variable3
Observation1	9	7	2
Observation2	8	2	1
Observation3	1	3	3

Option 2 - Iterate through the variables

- For Variable1, record the value of Observation 1.
- For Variable2, record the value of Observation 3, since Observation1 has already been used.
- For Variable3, record the value of Observation 2, since Observation1 and Observation3 have already been used.

The objective value would be $9 + 3 + 1 = 13$.

Since these two greedy algorithms offer two different objective values, it is obvious that greedy algorithms can provide faulty results. The best solution is the following combination:

$$\text{Observation1.Variable2} + \text{Observation2.Variable1} + \text{Observation3.Variable3}$$
$$= 7 + 8 + 3 = 18.$$

The source of the error identified here is that the typical scope of a greedy algorithm is limited to the current iteration and to prior iterations. They usually have no knowledge or concern as to how a decision will affect future decisions. How then can Dijkstra's algorithm, a greedy algorithm, reliably produce the desired objective? The proof of Dijkstra's algorithm is beyond the scope of this paper. However, the process that ensures optimality, as well as the intuition behind the process, will both be discussed.

In the following sections we will implement the Dijkstra algorithm and evaluate its performance against two heuristics. The large datasets that are typical in the real world are the motivation behind applying heuristics to reduce the period required to generate a path. We will first implement lessons learned from computer science to make the algorithm more efficient while not affecting its optimality condition. Next, we will apply heuristics to limit the number of nodes and arcs that are considered by the algorithm. The heuristics are expected to generate a nearly optimal path in a reduced amount of time.

2 IMPLEMENTING DIJKSTRA

Before the algorithm can be applied to find a shortest-path we need several things. First, we need a network including a connected graph $G = (N, A)$ and a function mapping costs to the arcs in A. Next, we need to identify a starting node and an ending node. If the desire is to know the best paths from a starting node to every other node in the network then the ending node does not need to be specified (we are not concerned with this case.) Finally, an additional constraint must exist, that the arcs in A all have non-negative costs.

Let us discuss how Dijkstra's algorithm works. We begin the algorithm by defining two sets of nodes: let I represent the set of nodes that have been included in the solution and let S represent the set of nodes that are still available for inclusion. Initially, $I = \{\}$ and $S = N$. The steps of the algorithm are as follows[3, 4]:

Step 1: Remove the starting node a from S and place it into I.

Step 2: Generate a list of candidates for entry into I. $\forall$ arcs $(i, j) \in A$, if $i \in I$ and $j \in S$ then j is a candidate node.

Step 3: Choose the best candidate n_{best} and move it out of S and into I. Choose $n_{best} \mid c(a, n_{best}) = \text{argmin}(c(a, j)) \ \ \forall\, j \in S$.

Step 4: If n_{best} is the ending node or $S = \{\}$ exit the algorithm.

Step 5: If n_{best} is not the ending node then repeat from step 2.

When the algorithm completes, a tree exists that spans a sub-network of the original network. If no ending node was specified then the sub-network contains all the nodes in the original network. Tracing the tree from the starting node to any node n in the tree reveals the shortest path from the starting node to n. In practice, the trace from the starting node to the ending node is traced in reverse. The nodes

passed during this tracing are known as the ending node's predecessors.

Step 2 is the critical step that ensures optimality. However, evaluating the quality of the candidates on each iteration of the algorithm can be very costly. The idea of candidates is future-thinking, proposing choices to move into the solution but realizing that future steps may find better alternatives. When a node is included in I it can introduce new candidate nodes. Additionally, it can introduce new paths to previously identified candidates. The costs of these new paths are evaluated against the costs of the previously identified paths and the candidate system is updated if its quality can be improved by the new paths. Consider the following simple example.

Example Figure 2 depicts a network with four nodes ($N = \{1, 2, 3, 4\}$) and four arcs ($A = \{(1,2), (1,3), (2,4), (3,4)\}$). Our objective is to find the best way to get from node 1 to node 4.

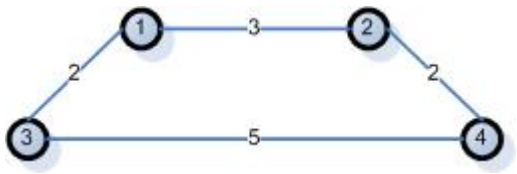

Figure 2: Network with four nodes and four arcs.

Executing the algorithm we have the following states:

I. $I = \{\}$ and $S = \{1, 2, 3, 4\}$(Figure 3)

Figure 3: No arcs included in solution.

II. Move node 1 into $I \rightarrow I$=$\{1\}$ and S=$\{2,3,4\}$(Figure 4)

a. Node 2 is a candidate with cost 3

b. Node 3 is a candidate with cost 2

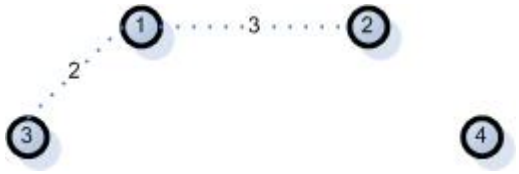

Figure 4: Arcs (1,2) and (1,3) are candidates.

III. Move node 3 into $I \rightarrow I$={1,3} and $S = \{2,4\}$(Figure 5)

a. Node 2 is a candidate with cost 3

b. Node 4 is a candidate with cost $5 + 2 = 7$

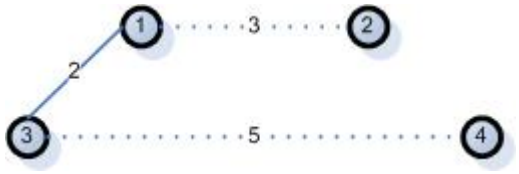

Figure 5: Arcs (1,2) and (3,4) are candidates.

IV. Move node 2 into $I \rightarrow I$={1,2,3} and $S = \{4\}$(Figure 6)

a. Node 4's candidate status is updated to reflect the lower cost ($3 + 2 = 5$) made available by the inclusion of node 2

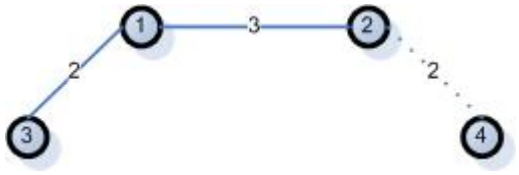

Figure 6: Arc (2,4) is a candidate.

V. Move node 4 into $I \rightarrow I$={1,2,3,4}(Figure 7)

VI. Trace the path from the ending node back to the starting node(Figure 8)

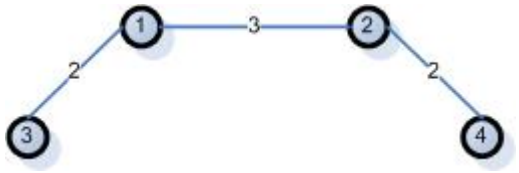

Figure 7: Solution tree.

Figure 8: Path identified.

We conclude that the best path from node 1 to node 4 contains two arcs (1,2), (2,4). The cost of the path is equal to the sum of the costs of its arcs, which is equal to 5. Note that even though arc (3,4) was considered for inclusion early, it was later refuted with a better possibility. This step of reviewing and excluding older candidates ensures the optimality condition is satisfied.

2.1 Modeling the Real World

We will generate shortest paths in a system of roads, which we claim is analogous to a connected network. Our goal is to be able to dynamically generate shortest paths in very large networks in very little time. There are three important aspects about this last statement:

- Dynamic Generation,
- Large Networks,
- Very Little Time.

An explanation of each of these notions follows.

Several factors can affect the availability of road segments. For example, a traffic accident could be congesting a roadway causing several nearby road segments to

be virtually impassable. Construction or repair work on roads can also make then impassable. These factors impose the need to dynamically generate the routes in real time (as opposed to storing and retrieving previously generated routes.)

One of the datasets used to test our implementation is the system of roads in Cook County, IL (including Chicago and its suburbs.) This dataset includes over one hundred fifty thousand distinct road segments. Although this may not be a typical size for a dataset, our goal is to provide a system that can accommodate any dataset.

Emergency responders use the term *Golden Hour* when referring to the hour immediately following a traffic accident [7]. The chance for a victim's survival dramatically decreases as time progresses. The goal for these emergency care providers is to deliver medical attention to victims in as little time as possible.

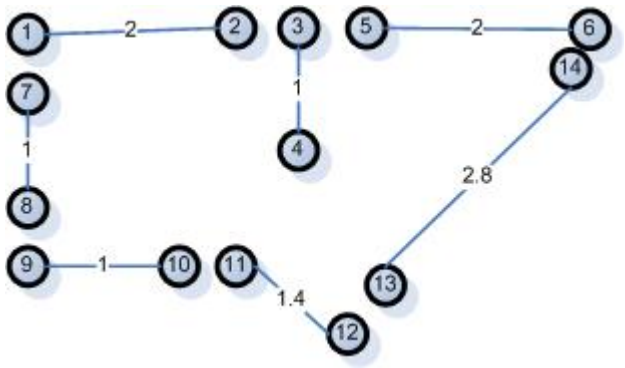

Figure 9: Dataset representation of road segments.

A typical dataset representing a system of roads contains information about road segments only, where each segment has two endpoints (Figure 9.) In this figure the node groupings {1,7}, {2,3,5}, {6,14}, {8,9}, {10,11}, and {12,13} each represent a single geographic point. Human intuition allows us to know that the end of one segment is the beginning of another, or possibly many others. However, a computer can only know what it has been told. To implement Dijkstra in this type of environment code must be written, not only to perform the algorithm, but also

to adapt the map datasets so that they more closely mimic the node and arc format of the theoretical system (Figure 10.) A reliable dataset is of utmost importance to the accuracy of the results produced by the algorithm.

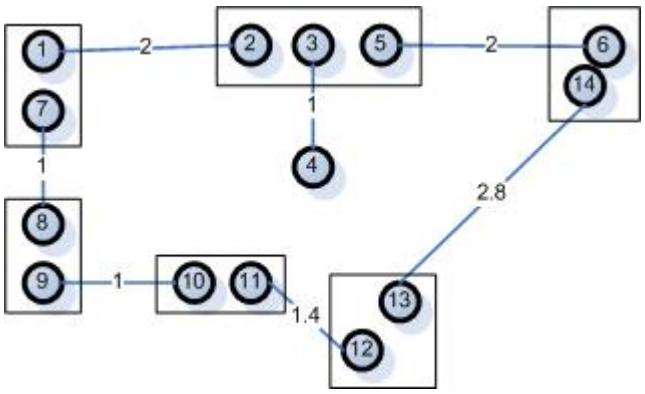

Figure 10: Merged dataset that mimics the mathematical model.

The merging of the nodes in the dataset is made possible by the two dimensional nature of the data. When loading a road segment into the required data structure the coordinates of its endpoints are compared to endpoints that have already been recorded. If the coordinates are identical to a previously recorded node then a reference is created between the new segment and the old node. This process creates the node and arc structure in memory that is necessary for the proper execution of the algorithm.

2.2 Improving the Base Algorithm

Having the dataset adapted to the in-memory data structure, it is now possible to develop the code for the algorithm. The pseudocode described earlier (page 7) indicates five distinct steps. Step 2, which seems simple enough, is the source for most of the time consumed by the algorithm. Before we apply heuristics, we should attempt to identify coding practices that could aid in the algorithm's performance while maintaining optimality.

In earlier comments about step 2 the updating of a candidate system is proposed.

In code, the candidate system is an intermediate storage list for the set of nodes that are candidates for inclusion in the solution. This list is not purged between iterations of the algorithms. Instead, it is updated to reflect new candidate inclusions as well as modifications to any appropriate existing candidates. This collection of candidates is maintained as an ordered list, reducing the time required to insert new candidates.

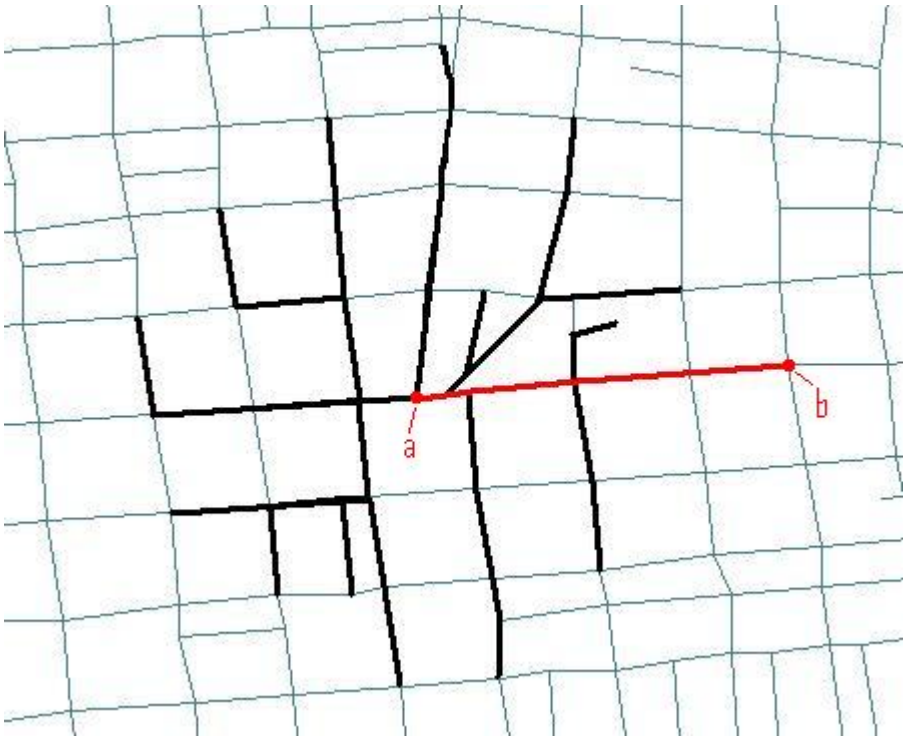

Figure 11: Clipped region from a real world dataset.

Maintaining this list between iterations also reduces the number of nodes that must be considered for candidate status. As a worst case scenario, it is possible to imagine that every node in S (the available nodes) would have to be evaluated as to whether or not it is adjacent to one of the nodes in I (nodes in the solution.) Additionally, any node in S adjacent to more than one node in I would have to be evaluated for each arc from that node into I to calculate its quality. In our system, only the nodes adjacent to the entering node need to be evaluated. The number of nodes evaluated in this manner is typically fewer than 5 due to the natural structure

of road systems. A natural consequence of maintaining and updating the ordered list is that the first node in the list after the updates is the best candidate for the next iteration of the algorithm.

With the algorithm implemented and the real world modeled, it is possible to calculate paths. Figure 11 depicts a sample spanning tree generated by the algorithm starting at node a and ending at node b. The thicker lines define the spanning tree while the nearly straight line from a to b identifies the shortest path from a to b. Notice how the tree expands radially from a until b is encountered and the algorithm exits.

The heuristics presented in the following sections rely on the natural topography of the dataset. The two-dimensional nature of the data causes a very specific pattern of progression by Dijkstra's algorithm. Our claim is that the time required to calculate a path can be abbreviated if this radial expansion pattern can be restrained to a smaller region.

3 DUAL-BRANCH APPROACH

3.1 Overview

For our first heuristic, let us consider a property of the shortest path between two nodes in a network. All road segments in our network are considered to be two-way roads. That is, a vehicle can traverse an arc (a, b) by starting at a and ending at b or by starting at b and ending at a. Our paths are intended for emergency vehicles in emergency situations. These vehicles have the ability to traverse the "wrong-way" on one-way roads in times of crisis. To apply this research to a wider array of drivers, one-way roads should be considered as such in a directed graph. However, assuming that all road segments can be traversed either way we know that for a given path P_{ab}, the distance(cost) of the path $d(P_{ab})$ is equal to the distance of the reverse of the path $d(P_{ba})$. This means that calculating the shortest path from one node a in a network to another node b would generate the same result as if the path were generated from b to a.

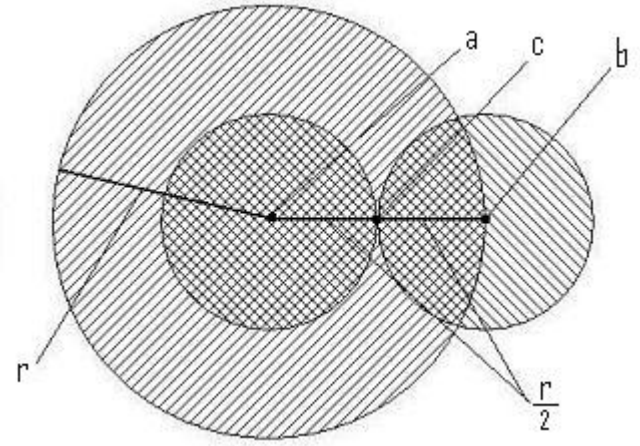

Figure 12: Approximation of coverage difference.

Figure 12 depicts three partially overlapping circular regions. The large circle has radius r while the two smaller circles have radius $\frac{r}{2}$. We hope to generate two small spanning trees, represented by the two small circles, that intersect close to the

midpoint of the optimal path that would connect their centers.

It is important to realize that the intersections in a system of roads are not evenly distributed over the entire system. Typically, urban areas have many more intersections per square mile than would be found in rural areas. We use the symbol ρ to represent the number of nodes per unit of area in a network and refer to ρ as the network's node density.

Proposition Let P_o be an optimal path generated between two nodes a and b in a network where ρ is uniform over the entire network. Next, let c be a node on P_o that bisects P_o such that $d(c,a) = d(c,b)$. If two spanning trees about a and b are generated until they intersect each other, they will intersect at c. The number of nodes encountered by the two small spanning trees will be half the number of nodes encountered by the spanning tree generated when calculating P_o.

Proof Assume the straight-line distance from a to b is r and that the straight-line distance from a to c and from b to c is $\frac{r}{2}$. Given a constant node density ρ we know that the number of nodes encountered when generating P_o is

$$N_o \approx \rho\pi r^2.$$

We also know that the number of nodes encountered when generating the smaller trees can be represented as

$$\begin{aligned} N_{h1} &= N_{h2} \approx \rho\pi(\tfrac{r}{2})^2 \\ &\text{or} \\ N_h &= N_{h1} + N_{h2} \\ &\approx 2 * \rho\pi(\tfrac{r}{2})^2 \\ &\approx 2 * \rho\pi(\tfrac{r^2}{4}) \\ &\approx \rho\pi(\tfrac{r^2}{2}) \\ &= \tfrac{1}{2}N_o \end{aligned}$$ □

In practice it is common that the node c does not exactly bisect the path P_o. Therefore, it is more appropriate to state that $N_h \approx \frac{1}{2}N_o$.

3.2 Expectation

Let P_o be the optimal path generated between two nodes in a network. Let P_h be a different path between the same two nodes in the network. As a consequence of the definition of the optimal path we can expect the following regarding the distances of the paths:

$$d(P_o) \leq d(P_h).$$

In real-world datasets it is rare that ρ is uniform. The use of a function $\rho(x, y)$, where x and y represent coordinates for a geographical location in the map, is more appropriate. When $\rho(x, y)$ varies greatly over the range of the dataset the heuristic can encounter problems.

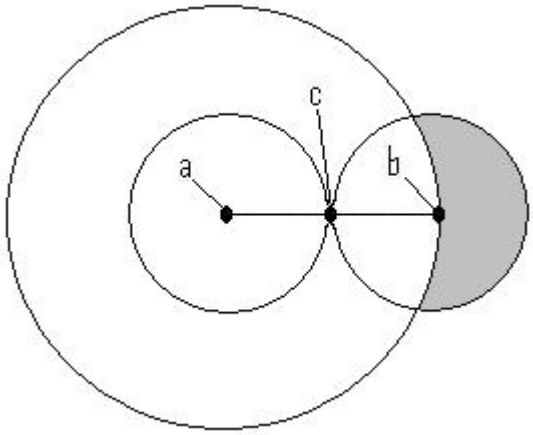

Figure 13: Node density difference illustration.

In figure 13, the shaded area represents a region of the network that exhibits a higher node density than the rest of the network. If both the optimal and dual-branch algorithms are applied to find a path from a to b arcs in the shaded region would be considered by the heuristic and not the optimal. This scenario would cause

the heuristic to label even more nodes than the optimal algorithm. This case will be illustrated when the simulation results are presented in section 3.4.

3.3 Dual-Branch Algorithm

As with Dijkstra, we begin the algorithm by defining two sets of nodes: I to represent the set of nodes that has been included in the solution and S to represent the set of nodes that are still available for inclusion. Initially, $I = \{\}$ and $S = N$. We refer to the spanning tree rooted at a as T_a and the spanning tree rooted at b as T_b. The steps of the dual-branch algorithm are as follows:

Step 1: Remove the starting node a and the ending node b from S and place them into I.

Step 2: Generate a list of candidates for entry into I.

$\forall$ arcs $(i, j) \in A$, if $i \in I$ and $j \in S$ then j is a candidate node.

Step 3: Choose the best candidate n_{best}, the one with the minimum total cost, and remove it from S and place it into I.

Step 3.1: Choose $k \mid c(a, k) = \text{argmin}(c(a, j)) \; \forall j \in S$.

Step 3.2: Choose $l \mid c(b, l) = \text{argmin}(c(b, j)) \; \forall j \in S$.

Step 3.3: If $c(a, k) < c(b, l)$ then $n_{best} = k$ else $n_{best} = l$.

Step 4: Determine if the T_a intersects the T_b.

Step 4.1: If yes then go to Step 5.

Step 4.2: If no then go to Step 2.

Step 5: Combine the sub-paths $a \rightarrow c$ and $b \rightarrow c$. This is the approximated path from $a \rightarrow b$.

3.4 Simulation Results

To compare the Dijkstra algorithm with the dual-branch heuristic we will generate paths with both methods. The dataset being considered contains over 12,000 arcs

with over 10,000 distinct nodes. To begin, 154 node pairs representing the starting and ending nodes of a path were randomly selected from the dataset. Next, both methods were used to calculate paths for each node pair and the number of nodes labeled, as well as the total distance traversed by the calculated path, were recorded.

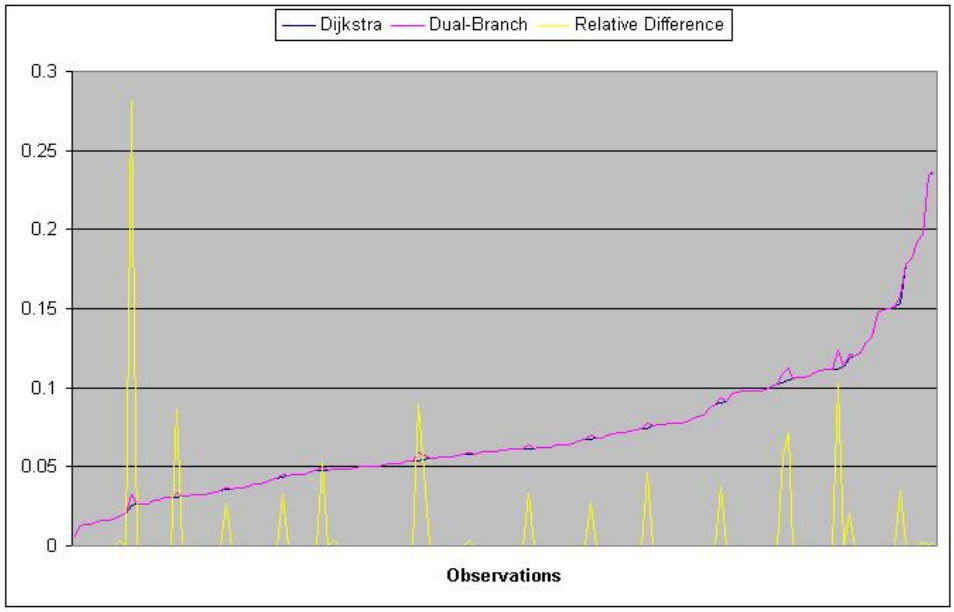

Figure 14: Distance comparison of Dijkstra and dual-tree heuristic trials.

Consider the graph in figure 14. The similarity of the Dijkstra and dual-branch curves indicate a high degree of similarity between the paths calculated by the Dijkstra and dual branch algorithms, which was the prediction. The high occurrence rate of heuristic paths equal in distance to the optimal path was unexpected. Most trials generated a relative difference less than 10%. In fact, out of the 154 trials performed, 132 of the paths generated by the dual-branch algorithm produced a path with a distance equal to that of the optimal. The percentage of average difference in distance for all 154 trials is 0.0062. The average distance calculated by the optimal algorithm is approximately 0.07 decimal degrees, which is approximately 3.5 miles in New Hanover County - the locale represented by the dataset. This suggests that there is a small relative difference between the dual-branch heuristic paths and the

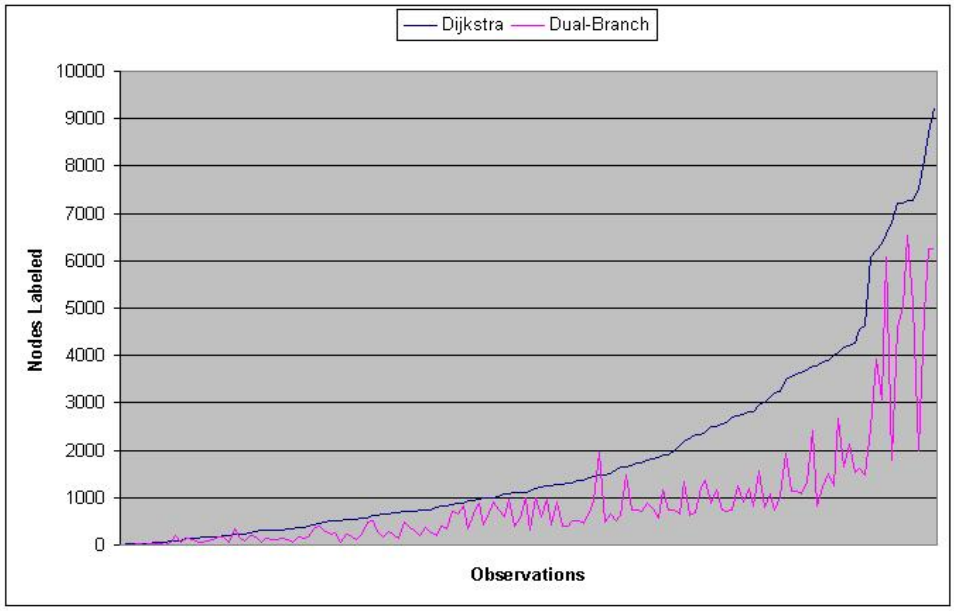

Figure 15: Labeling comparison of Dijkstra and dual tree heuristic trials.

Dijkstra paths.

Considering the chart in figure 15, it is easy to see that on several of the trials, the dual branch algorithm was able to find a path by labeling far fewer nodes than the optimal. In fact, the average number of nodes labeled for all of the trials by the optimal algorithm is approximately 1892. The average number of nodes labeled by the dual branch algorithm is approximately 937. This indicates the percentage of nodes labeled by the dual branch algorithm compared to the optimal algorithm is 0.495, which is very close to the $\frac{1}{2}$ value claimed earlier.

In section 3.1 the idea of node density was introduced. Three of the samples in figure 15 spike above the optimal curve. This phenomenon, where the heuristic labels more nodes than the Dijkstra algorithm, was predicted. It is probable that the ending node exists in a region of high node density like the area depicted by the shaded region in figure 13.

4 COVERING (WEIGHT-ADJUSTMENT)

4.1 Overview

The first heuristic presented identified a limitation of the system whereby all road segments were considered to be traversable in both directions. This constraint was required because we needed the path generated from a to b to be equal in distance to a path generated from b to a. The heuristic presented in this section does not require this property. In fact, depending on the parameters specified, this heuristic can be made to act just like the Dijkstra algorithm.

Consider the drawing in figure 16. Let P be a path from a node a and another node b in the network. Let d represent the straight-line distance between a and b. We will adjust the weight of the arcs that lie outside this region to influence the path propagation to remain inside the region.

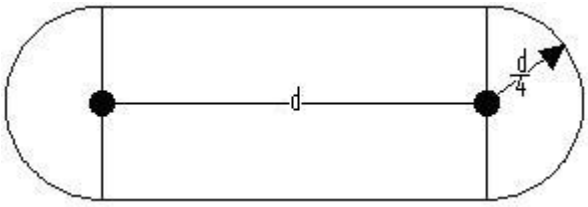

Figure 16: Illustration of a coverage region.

4.2 Weight-Adjustment Algorithm

This algorithm is essentially the same as Dijkstra except for some preprocessing of the data before the execution of the algorithm. We begin with a graph $G = (N, A)$ of nodes and arcs.

Step 1 - Choose a starting node a and an ending node b.

Step 2 - Calculate R as the region in space within αd units of the straight line connecting a and b.

Step 3 - Define a set of arcs $X = \{(i, j)|i \in R \, or \, j \in R\}$.

Step 4 - Increase the cost of the arcs in A but not in X by a factor of β.

Step 5 - Perform the standard Dijkstra algorithm.

This process only approximates the best path because of the possibility that an arc on the would-be optimal path is excluded from the solution due to its proximity.

In the algorithm we introduce the following parameters:

α - the coefficient used in determining the region R.

β - the cost multiplier applied to the arcs not in X.

For the paths generated in our trials, α was assigned a value of $\frac{1}{4}$ and β was assigned a value of 4. These factors can be adjusted to affect the execution of the heuristic. By decreasing α or increasing β it is likely that the number of nodes labeled by the algorithm would be reduced, which is the goal of the heuristic. However, these changes could also increase the relative difference of the paths generated by the heuristic algorithm and Dijkstra's algorithm.

Care must be taken when considering new values for these parameters. For instance, adjusting $\beta \leq 1$ would reduce the cost of the arcs exterior to the region described in figure 16, which would be counterproductive since this would cause the spanning tree to grow more easily outside the region. Also, by assigning α a value of $\frac{3}{4}$ an area greater the area spanned by the Dijkstra algorithm would be generated. Similarly, a value of $\frac{1}{2}$ would generate an area larger than the area that would be spanned by the dual branch heuristic. Therefore, it useful to use the range $0 < \alpha < \frac{1}{2}$ when sizing this region.

4.3 Simulation Results

Consider the graph in figure 17. The similarity of the optimal and weighted curves indicate a high degree of similarity between the paths calculated by the optimal and

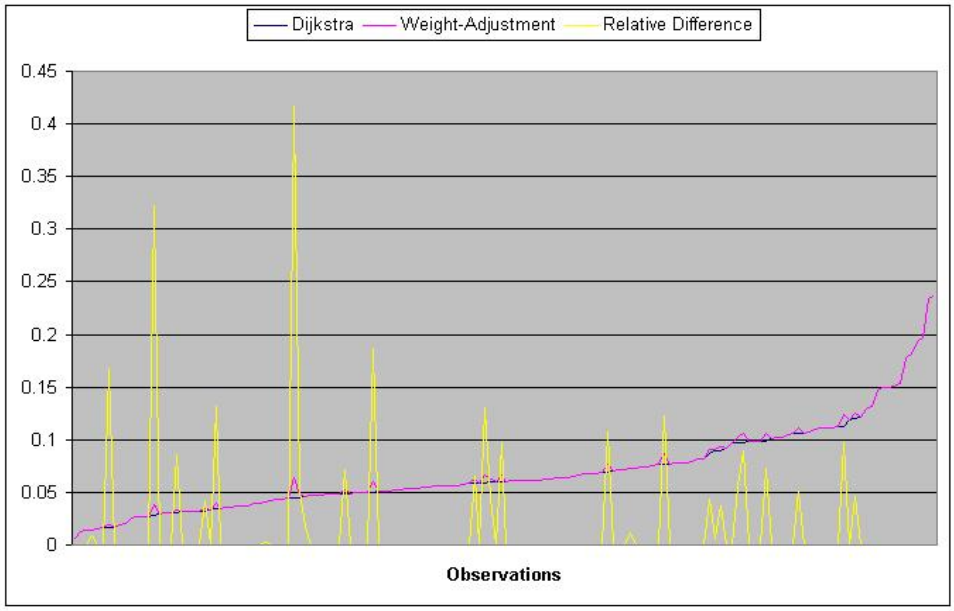

Figure 17: Distance comparison of optimal and weighting heuristic trials.

weighting algorithms.

In fact, out of the 154 trials performed, 125 of the paths generated by the weighting algorithm produced a path with a cost equal to that of the optimal. The percentage of average difference in cost for all 154 trials is 0.013. This indicates that there is a small relative difference in the paths generated by the weight-adjustment heuristic and the Dijkstra algorithm.

Considering the chart in Figure 18, it is easy to see that on several of the trials, the weighting algorithm was able to find a path by labeling far fewer nodes than the Dijkstra algorithm. In fact, the average number of nodes labeled for all of the trials by the optimal algorithm is approximately 1892. The average number of nodes labeled by the weighting algorithm is approximately 925. This indicates the percentage of nodes labeled by the weighting algorithm compared to the optimal algorithm is 0.489.

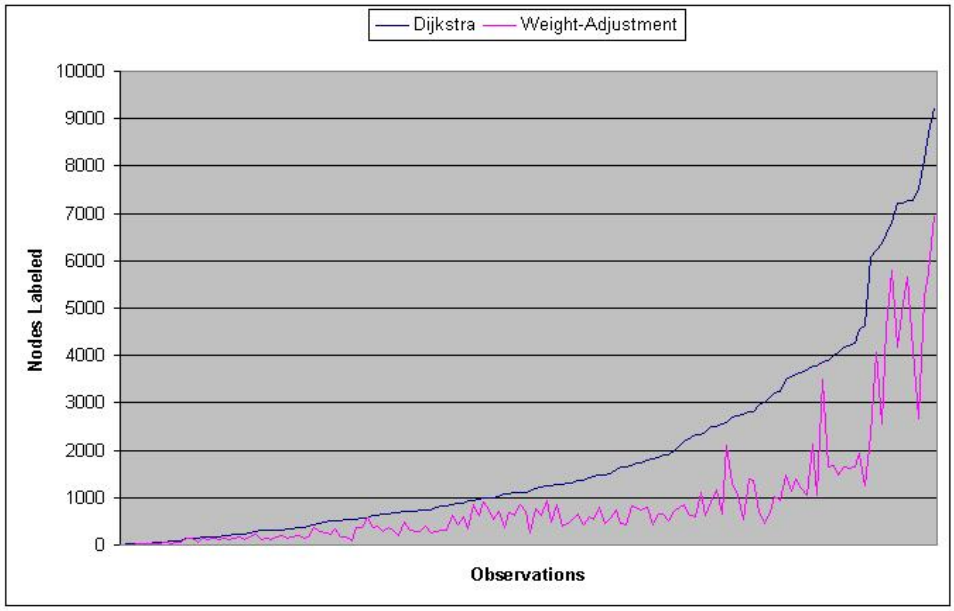

Figure 18: Labeling comparison of optimal and weighting heuristic trials.

Both of the heuristics have been shown to generate paths very close in cost to the one generated by the Dijkstra algorithm. Figure 19 shows a side by side view of the results so that the dual branch heuristic can be easily compared to the weighting heuristic.

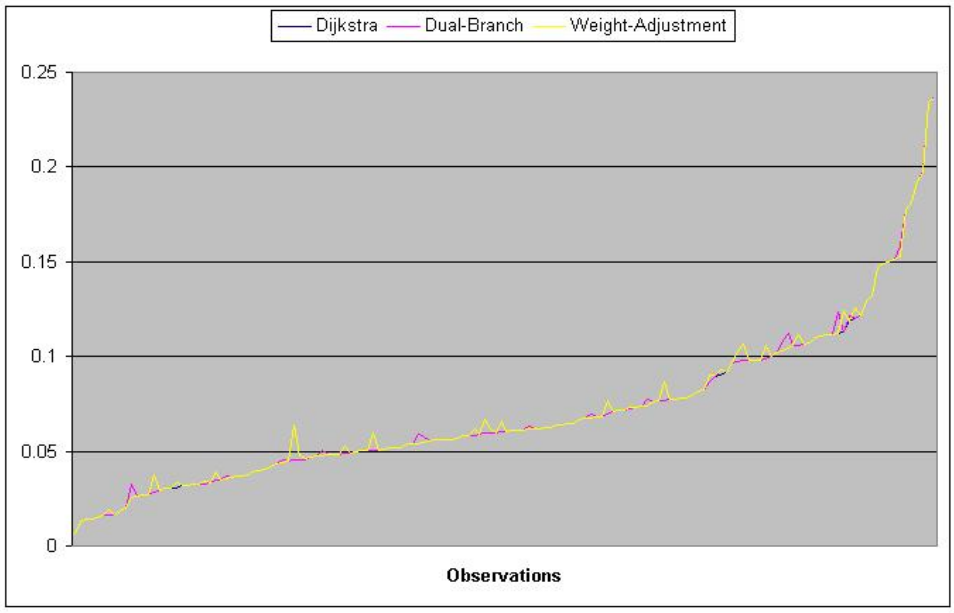

Figure 19: Distance comparison of all trials.

It has also been shown that both heuristics label approximately half as many nodes as required by the Dijkstra algorithm to generate a path. Figure 20 shows this side by side comparison. The data used to generate these graphs is provided in the appendix.

It should be noted that the dual branch method required the labeling of more nodes than the Dijkstra algorithm for three trials. This behavior was predicted in section 3.2 and is due to the difference in node densities in the regions surrounding the source and destination nodes. The weighting method never encountered this type of problem.

Interestingly, these two heuristics can be combined with very little effort. For

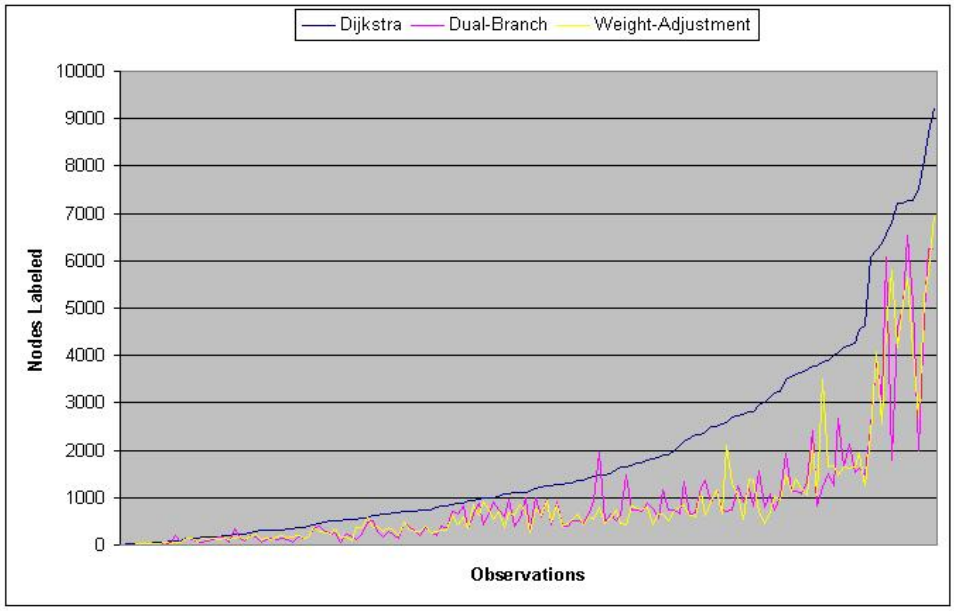

Figure 20: Labeling comparison of all trials.

instance, consider constructing a region as you would for the weighting heuristic. Next, generate a path using the dual branch heuristic with the weight-adjusted dataset. The weighting heuristic alone typically spans a region on the map that is egg shaped, with a larger radius about the source node than the destination node. Applying the dual branch and weighting heuristics together causes the solution set to span a region that is very similar to the region described in Figure 16. This further helps to reduce the number of nodes that are labeled to generate a solution.

6 FUTURE RESEARCH

Several open ended comments were made throughout this work indicating possibilities for future research. The weighing heuristic region calculation involves two parameters. It may be valuable to perform an examination of the benefits of adjusting these parameters. The two extreme cases are as follows:

1) the region depicts a line connecting the source and destination nodes or

2) the region spans the entire dataset.

It was also noted that both of these independent methods can be combined to further reduce the amount of labeling required to generate a solution. Only limited testing of this combined approach was performed. When it was performed, only the labeling was examined. It would be interesting to know if the error introduced by combining heuristics is equal to the best (least) error, equal to the worst (greatest) error, or equal to the sum of the individual errors.

The largest dataset examined during this research is Cook County, IL, which includes over 150,000 road segments. The tabular data provided in the index was generated using New Hanover County, NC road information. It would be interesting to see how well the algorithms perform on a nationwide dataset including millions of road segments.

REFERENCES

[1] E. W. Dijkstra, 1959. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik 1*, p.269-271, 1959.

[2] George B. Dantzig and Mukund N. Thapa, Linear Programming 1: Introduction, Springer-Verlag, New York, 1997.

[3] Harvey M. Salkin, Kamlesh Mathur, (Robert Hass - contributions), Foundations of Integer Programming, Elsevier Science Publishing Co., Inc., New York, 1989.

[4] Paul A. Jensen and J. Wesley Barnes, Network Flow Programming, John Wiley & Sons, New York, 1980.

[5] John Morris, Data Structures and Algorithms, http://ciips.ee.uwa.edu.au/ morris/Year2/PLDS210/dijkstra.html.

[6] Paul E. Black, Greedy Algorithm http://www.nist.gov/dads/HTML/greedyalgo.html.

[7] Lt. J.A. Lorek III, 2003. Personal Communication.

APPENDIX

The data used to generate the charts presented in this paper is provided here in tabular format. It is made available so that others may validate the statistics calculated and presented in the paper and to be used to generate other types of charts that may be more useful to the reader.

Table 2: Distance Calculations - 1

Optimal Distance	Weighted Heuristic Distance	Dual Heuristic Distance
0.006339748	0.006339748	0.006339748
0.012678073	0.012678073	0.012678073
0.013898518	0.013898518	0.013898518
0.013962653	0.014081535	0.013962653
0.015190375	0.015190375	0.015190375
0.016059801	0.016059801	0.016059801
0.016162025	0.018878289	0.016162025
0.016662519	0.016662519	0.016662519
0.018559151	0.018559151	0.0186148
0.020205504	0.020205504	0.020205504
0.025313811	0.025313811	0.032425117
0.026221216	0.026221216	0.026221216
0.026703671	0.026703671	0.026703671
0.026736075	0.026736075	0.026736075
0.028681308	0.037922422	0.028681308
0.029210181	0.029210181	0.029210181
0.030360208	0.030360208	0.030360208
0.030702856	0.030702856	0.030702856
0.030752731	0.033396474	0.033396474
0.031808852	0.031808852	0.031808852
0.031867613	0.031867613	0.031867613
0.032476326	0.032476326	0.032476326
0.032493641	0.032493641	0.032493641
0.032524115	0.033911419	0.032524115
0.033040803	0.033040803	0.033040803
0.034466761	0.039019416	0.034466761
0.034831936	0.034831936	0.034831936
0.035713429	0.035713429	0.036657479
0.036051933	0.036051933	0.036051933
0.036732725	0.036732725	0.036732725
0.036890143	0.036890143	0.036890143
0.037366565	0.037366565	0.037366565
0.03953572	0.03953572	0.03953572
0.039613654	0.039613654	0.039613654
0.040460171	0.040551086	0.040460171
0.042089172	0.042089172	0.042089172
0.043016745	0.043016745	0.043016745
0.043859758	0.043859758	0.045266134
0.044519253	0.044519253	0.044519253

Table 3: Distance Calculations - 2

Optimal Distance	Weighted Heuristic Distance	Dual Heuristic Distance
0.045253223	0.064140879	0.045253223
0.045359785	0.047651288	0.045359785
0.045450252	0.046224174	0.045450252
0.046808419	0.046808419	0.046808419
0.047756135	0.047756135	0.047756135
0.047776692	0.047776692	0.050308368
0.048035653	0.048035653	0.048035653
0.048160963	0.048160963	0.04834452
0.048517564	0.048517564	0.048517564
0.048985578	0.052511813	0.048985578
0.048988674	0.048988674	0.048988674
0.049479795	0.049479795	0.049479795
0.050239156	0.050239156	0.050239156
0.050244182	0.050244182	0.050244182
0.050450766	0.059872836	0.050450766
0.050709753	0.050709753	0.050709753
0.051201392	0.051201392	0.051201392
0.051626012	0.051626012	0.051626012
0.052097429	0.052097429	0.052097429
0.05215431	0.05215431	0.05215431
0.053673789	0.053673789	0.053673789
0.05384448	0.05384448	0.05384448
0.053937988	0.053937988	0.058764388
0.054883864	0.054883864	0.057450343
0.055565754	0.055565754	0.055565754
0.055777472	0.055777472	0.055777472
0.056212496	0.056212496	0.056212496
0.056342651	0.056342651	0.056342651
0.056448632	0.056448632	0.056448632
0.056776941	0.056776941	0.056776941
0.057901583	0.057901583	0.057901583
0.058218249	0.058218249	0.05839678
0.058304175	0.062102794	0.058304175
0.058770919	0.058770919	0.058770919
0.059344172	0.067053144	0.059344172
0.059458514	0.062086685	0.059458514
0.059618967	0.059618967	0.059618967
0.060132248	0.065937772	0.060132248
0.060219225	0.060219225	0.060219225

Table 4: Distance Calculations - 3

Optimal Distance	Weighted Heuristic Distance	Dual Heuristic Distance
0.060994442	0.060994442	0.060994442
0.061061809	0.061061809	0.061061809
0.061196	0.061196	0.061196
0.061487938	0.061487938	0.06353667
0.061551786	0.061551786	0.061551786
0.061820491	0.061820491	0.061820491
0.062469902	0.062469902	0.062469902
0.06256465	0.06256465	0.06256465
0.063802497	0.063802497	0.063802497
0.064113608	0.064113608	0.064113608
0.064303213	0.064303213	0.064303213
0.064831227	0.064831227	0.064831227
0.066773151	0.066773151	0.066773151
0.067559402	0.067559402	0.067559402
0.067692742	0.067692742	0.069550475
0.068028509	0.068028509	0.068028509
0.068469639	0.068469639	0.068469639
0.069497414	0.076964711	0.069497414
0.070801945	0.070801945	0.070820064
0.071437946	0.071437946	0.071437946
0.072016095	0.072016095	0.072016095
0.072609881	0.073440146	0.072609881
0.073355593	0.073355593	0.073355593
0.073778188	0.073778188	0.073778188
0.074016029	0.074016029	0.077404136
0.075961485	0.075961485	0.075961485
0.076660663	0.076660663	0.076660663
0.077042931	0.086492979	0.077042931
0.077296575	0.077296575	0.077296575
0.077392776	0.077392776	0.077392776
0.077837564	0.077837564	0.077837564
0.07833485	0.07833485	0.07833485
0.079866175	0.079866175	0.079866175
0.08198351	0.08198351	0.08198351
0.082359328	0.082359328	0.082359328
0.086754428	0.090553046	0.086754428
0.089579324	0.090203455	0.089579324
0.090032997	0.093386681	0.093386681
0.091438909	0.091438909	0.091438909

Table 5: Distance Calculations - 4

Optimal Distance	Weighted Heuristic Distance	Dual Heuristic Distance
0.09659645	0.09659645	0.09659645
0.097368319	0.102849236	0.097368319
0.097763844	0.10645533	0.097763844
0.097902817	0.097902817	0.097902817
0.097997804	0.097997804	0.097997804
0.098153062	0.098153062	0.098153062
0.098498716	0.105669338	0.098498716
0.10049381	0.10049381	0.10049381
0.102065181	0.102065181	0.102065181
0.102905852	0.102905852	0.108759413
0.104646387	0.104646387	0.112161209
0.106093128	0.106093128	0.106093128
0.106132478	0.111613395	0.106132478
0.106279449	0.106279449	0.106279449
0.106985326	0.106985326	0.106985326
0.11002411	0.11002411	0.11002411
0.110673788	0.110673788	0.110673788
0.111343163	0.111343163	0.111343163
0.111827725	0.111827725	0.111827725
0.112001718	0.112001718	0.123558961
0.113104461	0.124101929	0.113104461
0.118776823	0.118776823	0.121183324
0.119979955	0.125563578	0.119979955
0.121756587	0.121756587	0.121756587
0.129042202	0.129042202	0.129042202
0.132020906	0.132020906	0.132020906
0.147193548	0.147193548	0.147193548
0.149272084	0.149272084	0.149272084
0.149882032	0.149882032	0.149882032
0.151046604	0.151046604	0.151046604
0.1529897	0.152992935	0.158400801
0.177725093	0.177725093	0.177725093
0.181485813	0.181485813	0.181485813
0.192744187	0.192744187	0.192744187
0.196684196	0.196684196	0.197131528
0.234356291	0.234356291	0.234356291
0.23644375	0.23644375	0.237084364

Table 6: Labeling Calculations - 1

Optimal Labels	Weighted Heuristic Labels	Dual Heuristic Labels
14	13	11
16	8	5
16	13	9
36	37	9
36	32	36
40	23	18
47	42	34
56	29	18
66	46	12
74	35	56
89	69	206
96	63	61
144	130	130
148	133	102
149	68	47
167	155	46
176	117	79
178	147	108
181	123	172
204	136	184
207	101	65
217	146	355
218	176	128
238	120	87
249	165	192
277	222	182
309	120	67
316	156	116
316	117	134
318	161	107
325	191	131
327	135	109
340	167	51
377	205	172
382	150	132
393	179	181
431	356	342
457	272	396
477	245	274

Table 7: Labeling Calculations - 2

Optimal Labels	Weighted Heuristic Labels	Dual Heuristic Labels
507	230	227
517	347	243
521	168	43
529	183	232
543	96	166
548	373	116
564	380	236
576	574	490
615	381	525
635	406	298
640	279	170
648	383	275
679	285	204
682	207	143
699	494	484
715	313	359
722	291	284
733	272	197
748	389	378
751	247	247
797	279	194
812	302	384
837	321	346
844	633	713
881	429	661
882	583	814
927	348	327
941	848	660
976	627	874
982	920	436
983	808	590
1001	544	897
1010	720	730
1074	364	583
1076	678	974
1098	639	409
1099	840	562
1120	724	987
1135	243	302

Table 8: Labeling Calculations - 3

Optimal Labels	Weighted Heuristic Labels	Dual Heuristic Labels
1188	773	981
1212	617	603
1236	944	964
1256	486	430
1266	848	909
1270	399	404
1311	456	406
1317	543	519
1358	647	506
1372	439	441
1417	587	675
1449	537	951
1472	790	1965
1475	443	491
1495	565	642
1597	731	525
1641	453	611
1651	416	1470
1671	814	737
1735	759	737
1745	735	698
1795	795	868
1813	423	774
1854	642	563
1891	650	1163
1907	499	737
1980	698	750
2101	787	650
2180	864	1346
2255	611	633
2328	607	686
2329	1094	1210
2389	636	1376
2504	907	881
2514	1154	1175
2557	660	727
2581	2090	711
2689	1316	747
2719	1062	1257

Table 9: Labeling Calculations - 4

Optimal Labels	Weighted Heuristic Labels	Dual Heuristic Labels
2746	542	896
2800	1391	1207
2821	1361	828
2961	705	1563
3012	443	808
3111	701	1070
3224	1023	752
3225	937	1013
3498	1474	1934
3549	1140	1133
3605	1387	1141
3640	1205	1072
3697	1044	1335
3771	2142	2410
3785	1056	818
3868	3486	1213
3888	1646	1518
4009	1677	1249
4050	1487	2680
4180	1659	1659
4196	1616	2139
4268	1649	1535
4552	1942	1617
4619	1263	1490
6082	2328	2480
6232	4066	3927
6370	2543	3057
6562	4566	6078
6806	5796	1776
7203	4175	4616
7223	5083	5036
7263	5666	6546
7265	4129	5087
7539	2682	2001
8120	5230	4896
8702	5764	6264
9228	6954	6248