ABSTRACT

A real-world application of GIS and graph theory for analysis of an optimization problem is presented in this paper. The problem entailed determining the optimal route for distribution of mail on our University's campus. We focused our work only on the processes utilized to deliver mail externally from building to building. Information was collected on the procedure and routes utilized by the mailing department for the daily delivery of mail. To determine the optimal closed path, geographic information systems (GIS) techniques were utilized for analysis of the delivery routes and measurement of distances from building to building. The problem domain was modeled as a complete graph and approached as being a variant of the traditional Traveling Salesman Problem. An implementation of the Brute Force method developed in C++ code was utilized to solve the problem.

Keywords: GIS, Graph Theory, Optimization, Traveling Salesman Problem, Brute Force method, C++.
1.0 INTRODUCTION
A problem was presented to optimize the delivery routes for our University's mail distribution system. The purpose in solving the problem was to determine the optimal path (i.e. route) for the delivery of mail from building to building as opposed to determining the optimal methods for distribution of mail internally within each building. Therefore, we did not care about the internal methods utilized at each particular building when the mail arrived. Rather, we only cared about the processes utilized to deliver mail externally from building to building. Essentially, this breaks down into determining the minimum distance closed path, which is a classic example of the Traveling Salesman Problem.

Information was collected on the procedure and routes utilized by the mailing department for the daily delivery of mail. The current system consists of three routes: a morning walking route, a morning driving route, and an afternoon driving route. For sake of discussion and simplicity, each building name's acronym will be utilized. Each route must start and end at the UC, where the mailing department resides. The morning walking route starts at the UC and transgresses to five other buildings before returning back, and the afternoon driving route starts at the UC and transgresses to five buildings before returning.

Through observation and reasoning, the above routes were determined to be optimal. Furthermore, due to the relatively small distances and number of buildings covered for the above two routes, our algorithm, presented later, can easily be extended to cover the above instances if need be. Therefore, we decided to focus on the morning driving route. The morning driving route consists of ten buildings including the UC and a generic representation for the current route is presented in figure (1) below. The following conditions exist; the route must start and end at the UC and the Library must be the first building on the path from the UC.

To determine the optimal closed path, geographic information systems (GIS) techniques were utilized for analysis of the delivery routes and measurement of distances from building to building. The problem domain was modeled as a complete graph and translated into a Traveling Salesman Problem. An implementation of the Brute Force method developed in C++ code was utilized to solve the problem.

2.0 GRAPH CONSTRUCTION AND ANALYSIS WITH GIS
The buildings on the morning driving route were represented as vertices, and each edge connecting two vertices represented the minimal path (weight) from one building to another. Because there was the preexisting condition that the first stop on the mail delivery route must be the Library, the UC and the Library were combined into one
vertex. From this, a complete graph of size K9 was created where each vertex represented a building and an edge connecting two vertices represented the minimum path between them. The K9 complete graph is displayed in figure (2) below.

![K9 Complete Graph](image)

**Figure (2): Visualize Problem as a K9 Complete Graph**

The next step in the process was to determine the weights, the minimal distance from node to node (building to building), for each edge. This was accomplished by utilizing Intergraph's GeoMedia Professional, version 4.0, geographic information software. A USGS digital-ortho-photo-quad (DOQ) image of the campus and surrounding area was obtained from the Texas Natural Resources Information System (TNRIS) geospatial data clearinghouse [4]. The DOQ is a high-altitude near-infrared digital image with 1-meter spatial resolution, and this particular image was captured in 1995. During the georeferencing process, the image was projected into the Universal Transverse Mercator, zone 14 coordinate system and the datum used was the North America 1983. Since the DOQ image was captured in 1995, there are a few new buildings on the campus that are not represented in the image. Therefore, an up-to-date map of our campus was ascertained and digitally scanned into a "tiff" image, georeferenced, and overlaid on top of the DOQ within GeoMedia. The root mean squared error for the registration process was 1.84 meters, which is relatively low considering the tendency for warping during the registration process.

The scanned image of the campus was now represented in geographic coordinates allowing distances to be measured. To measure distances we utilized GeoMedia’s measuring tool. Distances were measured in meters, and the minimal path between two buildings was narrowed down by reasoning and observation. Constraints for the paths were that they must be a paved walkway, sidewalk, or road that the mail delivery vehicle, a golf cart, could traverse. Most of the shortest paths between two buildings were easily observed; however, if there was more than one path that could not be readily distinguished between two buildings as the shortest distance, measurements were conducted for each possibility and the one with the minimum distance was selected. Once
all paths were determined, each edge in the complete graph was assigned its corresponding weight representing the minimum distance. After the weights for the graph had been selected, an approach to solving the Traveling Salesman Problem needed to be determined.

**3.0 TRAVELING SALESMAN PROBLEM**

The conventional Postman solution did not apply in our case because we did not care about traversing every edge as in a Eulerian tour. Our objective was to find a minimal closed path, a Hamiltonian circuit, based upon distance. In essence, we needed to find the minimal closed path which starts at the UC, goes to the Library, and then traverses the other eight buildings exactly once before returning to the UC. This is conventionally known as the Traveling Salesman Problem (TSP). A more formal description of the problem is as follows:

*Modeling the TSP as a complete graph with n vertices [refer to section 2.0 above], we can say that the salesperson desires to make a tour, or Hamiltonian cycle, visiting each city (vertex) exactly once and finishing at the city he or she starts from. There is a cost \( c(i,j) \) to travel from city \( i \) to city \( j \), and the salesperson wishes to make the tour whose total cost is minimum, where the total cost is the sum of the individual costs along the edges of the tour [1].*

As mentioned previously, the objective of the TSP is to find the minimal (optimal) Hamiltonian circuit in a weighted graph. First though, one must prove that the graph in question has a Hamiltonian circuit. Thus far, there is no quick way for determining if a graph is Hamiltonian (i.e. it has a Hamiltonian circuit). There are some rules that can be applied to a graph to tell if it is not Hamiltonian, but the problem of "is \( G \) Hamiltonian?" is an NP-complete problem [2]. In our case, since the graph is complete, it is evident that our graph is Hamiltonian.

The next challenge in the TSP is to actually find the minimal Hamiltonian circuit. Presently, there exists no fast algorithm that can be used and guaranteed to produce the optimal solution. In fact, theorems show that a fast algorithm solution to the TSP is unlikely to exist. It is an NP-hard problem [1]. There are certain heuristics that can be applied based upon graph characteristics. One heuristic is known as the Nearest Neighbor. It is a greedy algorithm similar to a prioritized depth-first traversal, but it has no performance guarantee. It can sometimes produce extremely non-optimal solutions. Two other heuristics exist for graphs that satisfy the triangle-inequality. These two heuristics are known as Double the Tree and Christofide's Algorithm. Both algorithms find a minimum spanning tree, create a Eulerian tour of an associated graph, and then extract a Hamiltonian cycle from the Eulerian tour by taking shortcuts [2]. Christofide's Algorithm uses strategies similar to the Chinese Postman. For all instances of the TSP that obey the triangle inequality, the solution obtained by Double the Tree is never worse than twice the optimal, and the solution obtained by Christofide's Algorithm is never worse than 1.5 times the optimal solution. Christofide's Algorithm achieves the best-known performance guarantee of any approximate algorithm, but whichever heuristic is chosen, there exists no guarantee that the solution provided is the most optimal [2]. Furthermore, our graph does not satisfy the triangle-inequality since our
edges represent paths, which contain curves and avoidances of obstacles; our edges are not straight-line distances.

The only method guaranteed to produce an optimal solution is the Brute Force algorithm, which lists all possible Hamiltonian circuits, finds the weight of each circuit, and then selects the minimum circuit. The problem with this method is that it is computationally expensive and only feasible for graphs with a relatively small number of vertices. For a complete graph, this becomes the number of permutations for the vertices in the graph, \(n!\). From this, it is evident that as values for \(n\) increase, the Brute Force method can quickly become impractical to implement, even for today's latest super-computing machines [3]. Fortunately for our case, the graph consisted of only nine vertices so the Brute Force method could easily be applied to provide a quick, optimal solution without requiring any special computational capabilities.

4.0 BRUTE FORCE ALGORITHM

Due to the preexisting condition that the mailing route must start and end at the UC, we needed to compute all possible Hamiltonian cycles originating and ending from that node in our graph. This simply entailed computing all possible permutations for the other \((n-1)\) vertices of the graph, \((n-1)!\). The deduction is as follows:

Let's select any node \(A\) among the \(n\) vertices in a complete graph \(G\) as the starting point. Then by definition of a Hamiltonian cycle for \(A\), every cycle must start at \(A\), go through all other nodes in \(G\) once without repetition of any node, and return back to \(A\). Starting from \(A\), there are \((n-1)\) nodes left for which one can traverse to. Setting up a string with \((n-1)\) positions. The first position in the string indicates the number of possibilities \(A\) can be linked with any other node. As there are \((n-1)\) nodes reachable from \(A\), the first position contains \(n-1\) possibilities. Similarly, for every possibility from position 1, there are \((n-2)\) possibilities. For every possibility from position 2 in the string, there are \((n-3)\) possibilities and so on. In total there are \((n-1)!\) possible Hamiltonian cycles starting and ending at an arbitrary vertex \(A\) in \(G\).

```cpp
void ComputePermutation(int num_nodes, int paths[][9], int StepArray[], int &i, int &j)
{
    for(int a=num_nodes; a>=2; a--){
        bool B = true;
        for(int m=0; m<j; m++)
            B = (B && (a!=StepArray[m]));
        if(B) {
            StepArray[j]=a;
            if(j!=7) {
                j++;
                ComputePermutation(num_nodes, paths, StepArray, i, j);
            }
            else if(j==7) {
                i++;
                for(int x=0;x=j;x++)
                    paths[i][x]=StepArray[x];
                i++;
                j=0;
            }
        }
    }
    if(a==2) j--;
}
```

Figure (3): Recursive C++ Function to Generate the Permutations
Therefore, since \( n = 9 \), we needed to compute the \((n-1)! = (8)! = 40,320\) possible permutations for the graph. Each permutation string represented a cycle, and a potential minimum route. To accomplish this task, we created a program in C++ that implemented the Brute Force method by computing the number of possible paths (i.e. permutations), calculating the distances for each, and determining the minimum. Figure (3) below displays the recursive function used to generate the permutations.

5. RESULTS

The determined optimal route, figure (4) below, had a minimum distance of 2000 meters. Only one route was found with this optimal distance. The current morning driving route used by the University had a distance of 2733 meters. The calculated optimal route is 733 meters shorter. Including the optimal route, there are 1,708 closed paths that have shorter total distances than the morning driving route currently in use by the University's mailing department.

6.0 CONCLUSION

This discussion presented a real-world application of GIS and graph theory for analysis of an optimization problem. The problem entailed determining the optimal route for distribution of mail on our University's campus. The problem domain was modeled as a complete graph and approached as being a variant of the traditional Traveling Salesman Problem. GIS tools were utilized for visualization, analysis, and measurement. The Brute Force algorithm was implemented to solve the problem. By applying GIS and graph theory methods, the problem was represented in a practical and manageable form, thus enabling an optimal solution to be found.

7.0 REFERENCES


