IMPLEMENTING THE DIJSKTRA'S ALGORITHM WITH PRIORITY QUEUE TO THE PATH FINDING PROBLEM IN RASTER GIS

A THESIS SUBMITTED TO THE GRADUATE SCHOOL OF
NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

MUZAFFER HAKBİLİR

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF MASTER OF SCIENCE

IN

GEODESY & GEOGRAPHIC INFORMATION TECHNOLOGIES

APRIL 2004

Approval of the Graduate School of Natural and Applied Sciences

_____

Prof. Dr. Canan Özgen

Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of master of science.

_____

Prof. Dr. Oğuz Işık

Chair of GGIT Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

_____

Asist. Prof. Dr. Zuhal Akyürek

Supervisor

Examining Committee Members (first name belongs to the chairperson of the jury and the second name belongs to the supervisor)

Prof. Dr. Vedat Toprak _____

Asist. Prof. Dr. Zuhal Akyürek _____

Prof. Dr. Oğuz Işık _____

Asist. Prof. Dr. Cevat Şener _____

Asist. Prof. Dr. Şebnem Düzgün _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully sited and referenced all metarial and results that are not orginal to this work.

Name, Last Name : Muzaffer Hakbilir

Signature :

ABSTRACT

IMPLEMENTING THE DIJSKTRA'S ALGORITHM WITH
PRIORITY QUEUE TO THE PATH FINDING
PROBLEM IN RASTER GIS

Hakbilir, Muzaffer

M.S.,Geodesy & Geographic Information Technologies

Supervisor: Assist.Prof.Dr. Zuhal Akyürek

April 2004, 98 pages

Network analysis in GIS is often related to finding solutions to transportation problems. In a GIS the real world is represented by either one of two spatial models, vector-based, or raster-based. Prefering raster or vector GIS is more a question of choice than of accuracy. A raster-based GIS model shows a better fit, when the problem is concerned with finding a path across terrain which does not have predefined paths.

The approach of this study is to translate the scenario into a 'least-cost path' graph with an associated cost function on the raster-based GIS layer. Sometimes, computation of shortest paths between different locations on a raster-based GIS has to be done in real-time. Therefore, knowing which shortest path algorithm runs fastest on real networks is needed. In order to meet this requirement, Dijsktra's algorithm with priority queue implementation is selected, because it reduces the time complexity of Dijsktra's algorithm from $O(V^2 \log V)$ to $O(E \log V)$. The run-time results of Dijsktra's algorithm, Dijsktra's algorithm with priority queue implementation and ArcMap Spatial Analyst Tool are compared for a number of raster GIS layers which have different number of nodes. Dijsktra's algorithm with priority queue implementation and Spatial Analyst tool of ArcMap show a linear relationship between node numbers and time, whereas Dijsktra's algorithm represents a quadratic relationship. Hence, when the

number of nodes and edges in graph is increased, the run-time performance of the Dijsktra's algorithm decreases rapidly.

Keywords : Shortest Path, Dijsktra, Priority Queue, Geographic Information Systems (GIS), Time Complexity.

ÖZ

DIJSKTRA ALGORITMASININ
ÖNCELİKLİ KUYRUK İLE GERÇEKLEŞTİRME YÖNTEMİNİ
KULLANARAK HÜCRESEL CBS'DE ROTA BULMAK

Hakbilir, Muzaffer

Yüksek Lisans, Jeodezi ve Coğrafi Bilgi Teknolojileri

Yüksek Lisans Proje Yöneticisi : Yrd. Doç. Dr. Zuhal Akyürek

Nisan 2004, 98 sayfa

Cografi Bilgi Sistemleri (CBS) ile ağ analizleri genellikle ulaşım problemlerine çözümler getirir. Gerçek dünya, CBS ile iki konumsal modelden (vektör-tabanlı, hücre-tabanlı) biri ile ifade edilir. Vektör veya hücre-tabanlı CBS'lerinin tercih edilmesi, hassasiyet probleminden çok tercih problemidir. Hücre tabanlı CBS modeli, önceden tanımlanmış rotaların bulunmadığı zaman arazi üzerinde rota bulma problemi için daha uygun görülmektedir.

Bu çalışmada kullanılmış olan yaklaşım, problemin hücresel harita üzerindeki maliyet fonksiyonu ile, "minimum maliyetli yol" grafiği problemine dönüştürülmesidir. Bazen hücre tabanlı CBS'lerinde iki yer arasındaki en kısa yol gerçek zamanlı olarak hesaplanabilir. Bu yüzden gerçek ağlar üzerinde, hangi "en kısa yol bulma" algoritmasının en hızlı çalışacabileceğini bilmek gerekmektedir. Bu gereksinimi karşılayabilmek için, Dijsktra algoritmasının öncelikli kuyruk yapısı ile gerçekleştirimi yöntemi tercih edilmiştir. Çünkü, bu Dijsktra algoritmasının zaman karmaşıklığını "$O(V^2 \log V)$"den "$O(E \log V)$"ye düşürmektedir. Farklı sayıda düğüm ve bağlantılardan oluşan belli bir sayıdaki hücre tabanlı CBS katmanı; Dijkstra algoritması, Dijsktra algoritmasının öncelikli kuyruk yapısı ile gerçekleştirimi yöntemi ve ArcMap programının Spatial Analyst aracının çalışma zamanlarına göre karşılaştırılmışlardır. Dijsktra algoritmasının öncelikli kuyruk yapısı ile gerçekleştirme

yöntemi ve ArcMap programının Spatial Analyst aracında, dügüm sayısı ve zaman arasında doğrusal bir ilişki görülmektedir buna karşın, Dijsktra algoritmasında ikinci dereceden bir ilişki sergilenmektedir. Sonuç olarak, grafik içerisindeki düğüm ve bağlantı sayısı arttıkça, Dijsktra algoritmasının çalışma zamanı performansı belirgin bir şekilde düşmektedir.

Anahtar Kelimeler : En Kısa Yol, Dijsktra, Öncelikli Kuyruk, Coğrafi Bilgi Sistemleri (GIS), Zaman Karmaşıklığı.

To My Parents

ACKNOWLEDGMENTS

I would like to express my gratitude to my supervisor Assist.Prof.Dr.Zuhal Akyürek, whose immense scope of knowledge, experience and dedication to academic life deeply impressed me. I tried to benefit from her knowledge, ideas, and feelings.

I would like to thank to Assist.Prof.Dr. Cevat Şener for his interest and comments on this thesis. I would also want to thank to Prof. Dr. Vedat Toprak and Prof. Dr. Oğuz Işık for their encouragement and support.

Thanks to Tahsin Alp Yanar for his suggestion, comments and technical support.

I would like to thank to my friends especially, Kadri Yetiş, Murat Bal, Ercüment Aksaray, Derya Fındıkoğlu, and Gökhan Erbay for their great support, patience, and encouragement.

Particularly, I am deeply grateful to my engage to be married, Tuba Hakbilir for her great support, patience, and encouragement. I offer sincere thanks for her unshakable faith in me and her willingness to endure with me the vicissitudes of my endeavors.

Finally, I would like to extend my gratitude to my mother and father without their support, I would not even be where I am today.

TABLE OF CONTEXT

APPENDICES

LIST OF TABLES

TABLE

LIST OF FIGURES

FIGURE

ABBREVIATIONS

DBMS	: Database Management System
DEM	: Digital Elevation Model
GIERS	: GIS-based Intelligent Emergency Response Systems
GIS	: Geographic Information Systems
HWA	: Hierarchical Wayfinding Algorithm
IBS	: Intelligent Building Systems
ITS	: Intelligent Transportation Systems
NHWA	: Non-Hierarchical Wayfinding Algorithm
OODBMS	: Object-Oriented Database Management System
OVRP	: Open Vehicle Routing Problem
PGS	: Personnel Guidance System
SDSS	: Spatial Decision Support System
USA	: United States of America
XML	: Extensible Markup Language
WTC	: World Trade Center

CHAPTER 1

INTRODUCTION

1.1 Development In GIS

GIS has been developed by a long tradition of map making. In many respects, modern GIS dramatically increases the amount of information that can be contained and manipulated in a map. On the other hand, many of the same cartographic conventions and limitations apply to digital maps.

Scientists wanted to study man's use of the land which led to the study of spatial distributions of such things as soil, people, vegetation, climate, etc. Before computers the spatial database was a drawing on a piece of paper or film. Various symbols, colors and text codes together with legends were used to display geographical entities.

The father of GIS is recognized to be Dr. Ian McArth, a landscape designer, who published a book in 1969 dealing with spatial representations of features in 2 and 3 dimensions (Heward et al., 1998). In the 1970s Dana Tomlin, a Ph.D. student at the Harvard Graphics Lab, developed the first raster GIS program called MAP. This software was in the public domain. By the help of other scientists, GIS technology is improved with a great speed.

Recent developments in GIS and computer technology have made the spatial analysis possible and GIS is now playing an invaluable role in identifying, monitoring and tackling problems and, indeed opportunities in the natural and human environments.

1.2 Problem Definition

Network and transportation analysis within a geographic information system (GIS) environment have become a common practice in many application areas with the development of GIS technology. Sometimes, computation of shortest paths between different locations on a network has to be done in real time. Therefore, knowing which shortest path algorithm runs fastest on real networks, is needed.

Raster-based GIS is not commonly known for network analysis applications. A network model can be defined as a line graph, which is composed of links, nodes, junctions, edges, and linear characters. On a real world model, these elements should be associated with a direction, impedence, resistance and travel cost along the network.

In raster-based GIS cartographic space is defined as a surface. In order to adapt a network structure, each cell may be seen as a node linked to its eight neighbouring cells. Each node can represent the cost traversing this cell. The resistance, friction and difficulty in crossing cell are expressed in terms of cost, time and distance.

Each entity on a source map is represented by a number of vertices in raster-based GIS. Thus, resulting in a graph with a great number of nodes and edges. Increasing the size of the map or the resolution of the raster application causes an increase in the number of nodes and edges of the graph. The graph size effects the time performance of the shortest path analysis.

The shortest path algorithms should be used with any type of applications. For the same study area, it may be requested to find the shortest path for a number of scenarios. For example, setting up the route for electric lines or pipelines, tracking activity and navigating the military troops from any location in terrain to another location are some types of scenarios for which the least-cost path analysis is necessary. The application should be dynamically adapted for a new scenario in order to find the shortest path.

1.3 Objective

In this study, it is aimed to develop an application program in order to find the least-cost path in raster GIS. Since raster GIS applications cover a considerable number of nodes and edges, it is intended to implement a fast shortest path algorithm.

It is planned to develop a number of raster-based GIS applications with different map sizes or resolutions. The raster-based GIS layer shall be automatically built by using the program. The shortest path algorithm is based on the fixed costs of the vertices in the raster-based GIS layer. The cost reflects the impedence of each cell to the movement. Fixed cost of the vertices are planned to be retrieved from the overlay of the appropriate layers before finding the least-cost path. The limitations and capabilities of raster-based GIS are discussed.

The performance of the application program is compared with the Dijsktra's shortest path algorithm and an application with a commercial GIS software namely, ArcGIS, Spatial Analyst Tool. It is planned to derive the time performance curves on two dimensional plane by using several raster-based GIS applications with different number of nodes. In this study, it is intended to increase the time performance while preserving the quality of the shortest path found.

In a previous study (Ünlü, 2002), the scenario was the movement of the troops between two distinct locations on a raster-based GIS application. But, in this study, it is intended to use the raster GIS application for other purposes such as: tracking, setting up the route for electric lines or pipelines, ie. For this reason, it is experienced to adapt the application to tracking activity.

1.4 Organization Of Thesis

The following chapters seek to define the subject in a wide perspective.
Chapter 2 examines the shortest path algorithms in the literature. The detailed description of Dijsktra's algorithm with priority queue implementation is presented. The binary heap method is explained in detail. The studies prepared by using shortest path algorithms are discussed.

Chapter 3 presents the application part of the study. The data layers are explained in detail. The application is adapted to tracking activity. The performance comparison of the algorithms and Spatial Analyst tool of ArcMap is also presented in this section. The resulting paths for each algorithm is evaluated according to the path quality criteria.

The study is completed with a brief section of conclusion in Chapter 4.

CHAPTER 2

SHORTEST PATH ANALYSIS

Traditionally, network analysis, path finding and route planning have been the domain of graph theory. Network analysis in GIS is often related to finding solutions to transportation problems. In a GIS, the real world is represented by either one of two spatial models, vector-based, or raster-based (Husdal, 2000). Real world networks, such as a road system, must be modelled appropriately to fit into the different spatial models. Even though the models differ, the solution to different transportation problems in either raster or vector GIS uses the same path finding algorithms, which are based on graph theory. Whether raster or vector GIS application is to be preferred is more a question of choice than of accuracy.

A vector-based network model is likely to be more suitable than a raster model for analysing precisely defined paths, such as roads and rivers or drainage canals, i.e. discrete entities that derive mainly from the built environment, and where attributes play a major role in determining the network. A raster-based network model, on the other hand, seems to be more fit, when the problem is concerned with finding a path across terrain that does not have predefined paths, and where the network does not consist of many attribute layers and artificial directional constraints. Hence, raster applications are more likely to be passed on movement on the surface (terrain) than movement along a vector-based network, since the general idea of finding the least-cost path in terrain is linked the cell to the cell, not along a finite line.

2.1 Graph Theory

A *graph* is a mathematical way of representing the concept of a network (Rodrigue, 2003). A network has points, connected by lines. In a graph, points are called as *vertices*

(sometimes also called nodes), and the lines are called as *edges*. Figure 2.1 shows a sample graph where edges are denoted by letter "e" and vertices are denoted by letter "v".

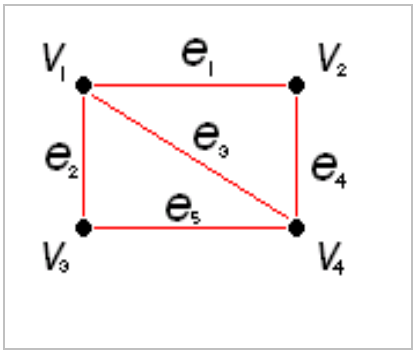


Figure 2.1 Sample Graph

In this graph, v1, v2, v3, v4 are vertices, and e1, e2, e3, e4, e5 are edges. Using the graph above as a guide, the definition of a graph can be formulated. A graph contains:

- a set of vertices, commonly called V.
- a set of edges, commonly called E.
- a relation *f* that maps to each edge a set of endpoints.

In the example described in Figure 2.1,

- V={v1, v1, v1, v4}
- E={e1, e2, e3, e4, e5}
- f such that e1 maps to {v1, v2}, e2 maps to {v1, v3}, e3 maps to {v1, v4}, e4 maps to {v2, v4}, and e5 maps to {v3, v4}.

Graphs are classified into different types. The edges described can have direction as well. Instead of a line, an arrow can be drawn; and instead of a set of vertices, ordered pairs are used. These types of graphs are called as directional graphs. In addition to that, if the edges of a graph have some kind of value associated to it, it is called as weighted graph. These types of graphs are used in path finding analysis. A path is a sequence of edges that are traveled in the same direction. For a path to exist between two vertices, it must be possible to travel an uninterrupted sequence of edges.

2.2 Shortest Path Algorithms

Shortest path algorithms are divided in two categories. They are all-pairs shortest path algorithms and single-source shortest path algorithms.

i. All-Pairs Shortest Path Algorithms

The problem that considers finding the shortest path between all pairs of vertices on a graph is called as all-pairs shortest path algorithms. Table 2.1 shows the all-pairs shortest path algorithms with their time complexities. It uses the symbology of "O(V)". "O" is used to denote the upper limit for the asymptotic time complexity and "V" is used for the number of vertices (number of iterations) over source map. Shortest path is computed by searching the vertices of the source map (grid data). So, in order to traverse all of the vertices of the grid map, computer iterates "V" times over the grid data. For example, for Dijsktra's algorithm, computer iterates V^3 (VxVxV) times in order to complete the algorithm. The number of iterations is used as a metric in order to estimate time complexity. The Dijsktra's single-source shortest path algorithm is converted to the all-pairs shortest path algorithm by running Dijsktra's single-source shortest path algorithm "V" times.

Table 2.1 Time complexities of all-pairs shortest path algorithms

Algorithms	**Time Complexity**
Floyd-Warshall	$O(\|V\|^3)$
Bellman-Ford	$O(\|V\|^4)$
Dijsktra	$O(\|V\|^3)$

ii. Single-source Shortest Path Algorithms

Single-source shortest path algorithms find the shortest path from a starting vertex to the destination vertex. As understood from the name (single-source shortest path), the path is found from the source vertex to all of the vertices. If the goal vertex is in the set of searched vertices than the path is found from starting vertex to the ending vertex.

There are several algorithms that find single-source shortest path. The most famous one is the Dijsktra's single-source shortest path algorithm (Dijkstra, 1959). Main idea of Dijkstra's algorithm is to maintain a set "S" of vertices whose final shortest path from the source have already been determined. The algorithm repeatedly selects a vertex "u", which is not in set "S". If this vertex has a minimum shortest path cost estimate then adds "u" to "S" and finds the neighborhoods of vertex "u". This process is repeated until all of the vertices have been searched. At the final step, a shortest path is found from source vertex to the goal for which the sum of the cost of the vertices on the path is minimum. This algorithm computes the shortest path in $O(V^2 \log V)$ iterations. A priority queue can be used to manage extracting the node with the smallest shortest-path estimate. This implementation of Dijsktra's algorithm reduces the time complexity from the degree of $O(V^2)$ iterations to $O(V)$. Dijsktra's algorithm with priority queue implementation is explained in detail in Section 2.4. In addition to that, A* and Hierarchical path planning algorithms are better ones compared to Dijsktra's algorithm in means of time complexity. Since, formulating the time complexities of these algorithms are difficult; the concept of these algorithms are explained briefly in order to have idea about the time complexities of them.

One of the shortest path algorithms, that is a popular one, is A* (Heyes and Jones, 2001) (The description of the algorithm is given in Appendix C). This algorithm uses a heuristic function in order to estimate the route that is going to the goal vertex. The concept of this algorithm is as follows: A* begins at a selected vertex. This is the starting vertex. A* then estimates the distance to the goal vertex from the searched vertex (current vertex). This estimate and the cost added together are the heuristic, which is assigned to the path, leading to this vertex. Basic heuristic function for path finding problem can be found by calculating the exact distance to the goal for each vertex that is searched. For A* algorithm, the time complexity of the algorithm is dependent on the quality of the heuristic function.

A* algorithm increases performance but on the other hand, the quality of the shortest path found is decreased. The reason of this quality problem and increase in performance is that, by using heuristic function some of the paths that are going to the goal vertex are not searched.

Hierarchical path planning (Pai and Reissell, 1998) is used as another method in order to improve the performance and making the path planning in a structured manner (Hierarchical path planning algorithm is given in Appendix D). In the hierarchical path planning, the map is converted to the grid table at different resolutions. For example there may be three levels. In the coarsest level (for example layer 3), the path is found by applying any single-source shortest path algorithms. Then, the path found is projected onto the next detailed layer (layer 2). This projected path is enlarged to a bigger region by applying a fixed margin to the path. The area found is used as the available vertices in layer 2. Shortest path algorithm is again used in this layer. Figure 2.1-Figure 2.3 visualize the hierarchical path-planning algorithm.

The time complexity of computing the path at level l is O(cdlogd), if Dijsktra's algorithm is used as a shortest path algorithm. Where d is the length of the path at level (l+1), and c is O(d) in the worst case. Time complexities of single-source shortest path algorithms, which have been explained, are summarized in Table 2.2. Since area (number of vertices) searched is reduced in hierarchical path planning, the time complexity of this algorithm is less than the Dijsktra's algorithm. Although the less time complexity; this algorithm has a disadvantage. It cannot find the correct path because detail information is lost in the coarse levels. So the path chosen at the coarsest level may not be the best path that can be chosen.

There are considerable empirical studies on the performance of shortest path algorithms reported in the literature (Dijkstra, 1959, Johnson, 1977, Fredman and Tarjan, 1987, Ahuja et al., 1990), but no clear answer is available as to which algorithm or a set of algorithms are appropriate for raster-GIS applications. Most of them uses Dijsktra's algorithm with a slight difference in the implementation of priority queue in order to get increase in time performance.

Table 2.2 Time complexities cf single-source shortest path algorithms

Algorithms	Time Complexity
Dijsktra	$O(\|V\|^2 \log \|V\|)$
Hierarchical Path Planning	O(cdlogd), where c, d are less than V.
A*	Dependent on the quality of heuristic function.

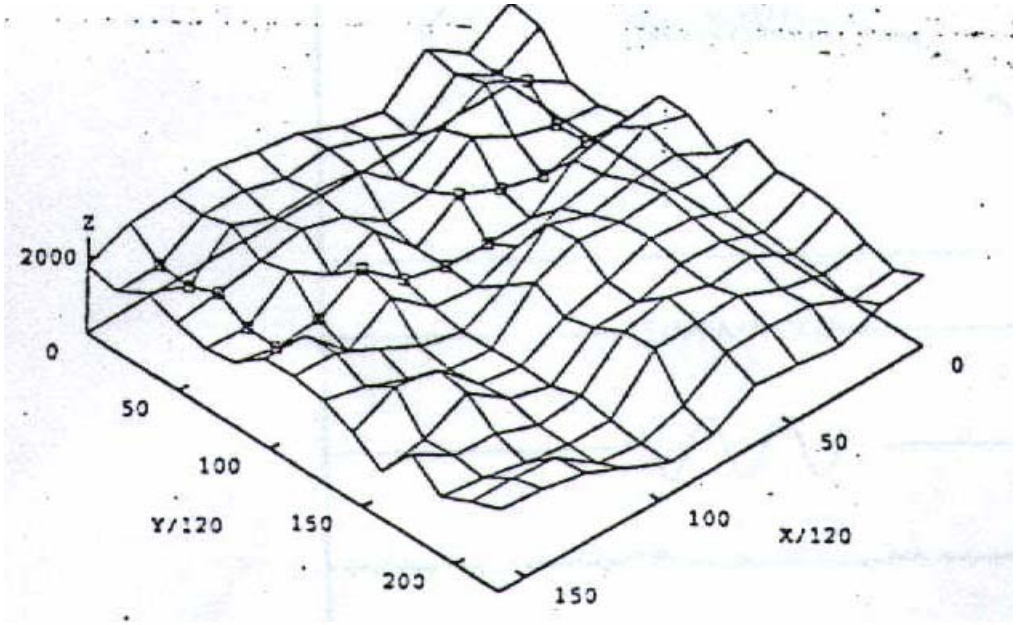


Figure 2.1 Level 4 of 4 paths in hierarchical path-planning algorithm

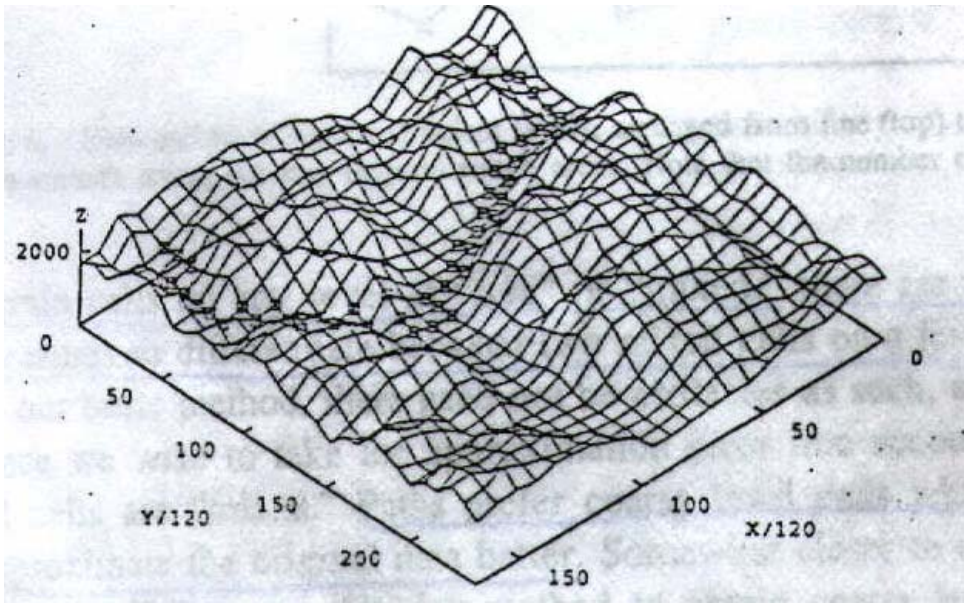


Figure 2.2 Level 3 of 4 paths in hierarchical path-planning algorithm

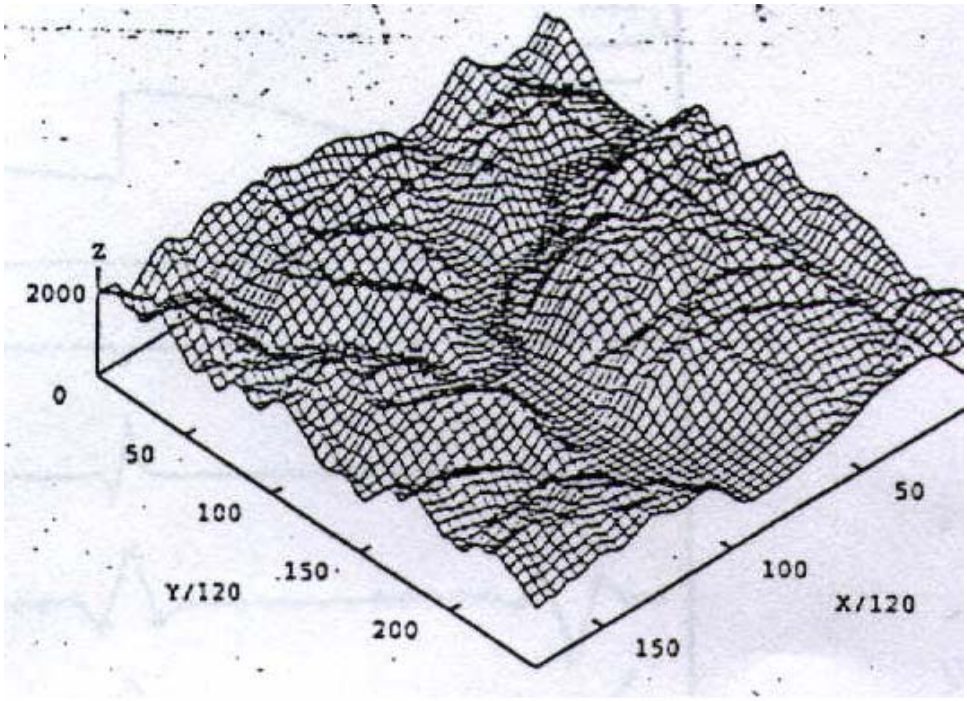


Figure 2.3 Level 2 of 4 paths in hierarchical path-planning algorithm

2.3 The Use of Shortest Path Analysis In GIS

Lee and Stucky, (1998) have implemented viewpaths by using the least-cost path procedure with visibility information. This study is constructed over a test site, which is a 200x200 DEM extracted from a USGS DEM of the Honolulu Beach, Hawaii area. It uses grid data in order to compute the least-cost path. In this study, four possible types of paths are computed. They are hidden paths, scenic paths, withdrawn paths, and strategic paths. In order to support the computation of these paths, two grids of visibility information are calculated. The first is a general viewgrid that records for each cell the number of cells visible. The second is a dominance viewgrid that holds for each cell the number of cells from which the cell is visible. The property of each path is explained here briefly.

- Hidden paths fall between the selected origin and the destination point, such that the path is minimally visible from all of the cells in the DEM. This path indicates the best route for military special forces operations or hidden roads and paths.

- The scenic path is located between the selected origin and the destination point such that the path has maximum visibility of all of the cells in the DEM. This path can be used to find the most scenic route for a bike trail, road, or hiking trail.

- The strategic path will always contain cells that have the best possibility of being minimally seen from other cells while maintaining maximum visibility to other cells. This path indicates the best route for military surveillance or reconnaissance.

- The withdrawn path is minimally visible from any cell, but it maintains minimum visibility of all of the cells in the DEM. This path indicates the best route for above-ground pipelines or high-power lines.

Golledge et al. (1998) have developed a GIS application for use in real time by blind travelers. This system is derived on the structure of Personnel Guidance System (PGS) for blind and vision-impaired travelers. It provides a database and a set of GIS functions that can be accessed in continuous real-time by a naive traveler moving through an unknown environment. While preparing the database, the needs of the blind users are taken into account. For example, the presence or absence of sidewalks, paths (e.g. across a park area), or trails (e.g. through a recreational area), etc. The GIS functions are as follows: partitioning, waypoint and route selection. Due to memory limitations, the entire map was partitioned. A buffer is drawn around the traveler. The detailed maps are loaded for the boxes, which are within the buffer. In waypoint method, the route is divided into a number of waypoints from start to stop. If the traveler reaches to the waypoint, he or she is informed by an audio response. A corridor is drawn in order to control the navigation of the traveler. If he or she leaves the corridor, hears a verbal warning message. Coming to the route selection, the route selection algorithm is a modified version of Dijsktra's algorithm. Once an origin waypoint and destination waypoint have been determined, it begins searching for possible routes. If more than one route is found, a route using fewest turns is given the highest priority for selection. This study is implemented by using ArcInfo.

Car et al. (2000) have developed a hierarchical way finding computational method on synthetic graphs and compared the hierarchical way finding algorithm to a non-hierarchical way finding algorithm (Dijsktra's algorithm). In this content, hierarchy has been used for partitioning the underlying network into smaller networks. This speeds up path computation, but the resulting paths are not always guaranteed to be optimal. Both of the algorithms are written in Borland C++ and integrated into ArcView.

The bottom-up hierarchization method is used to set a hierarchy. Level 0 represents the motorways, level 1 for main roads and level 2 is for local roads as seen in Figure 2.4.

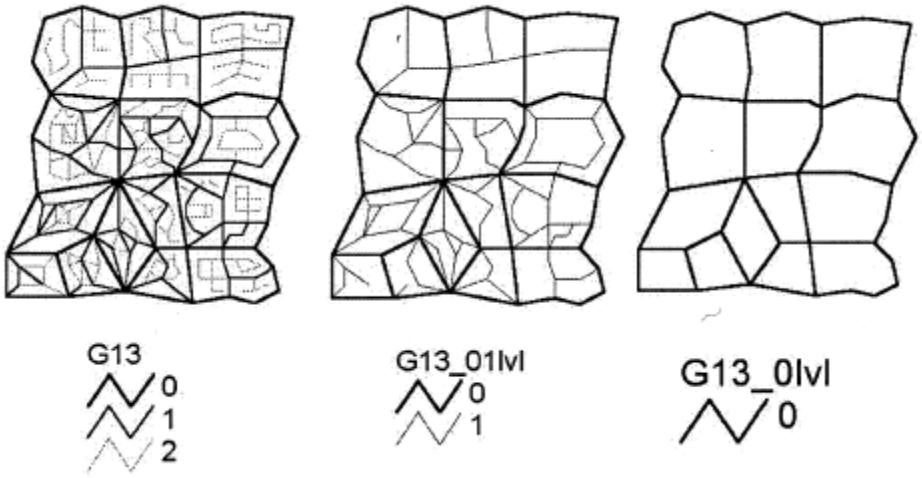


Figure 2.4 A graph with three hierarchical levels (Car et al., 2000)

As a result of this study, two conclusions have been stated.

- As the graph size (the number of vertices in a graph) increases, the benefit of using hierarchical wayfinding algorithm (HWA) over non-hierarchical wayfinding algorithm (NHWA) increases. For graphs with less than 300 nodes hierarchization does not make any significant difference.

- The level combination significantly influences the performance of HWA. The more levels traversed, the more likely that HWA and NHWA find different paths.

Tarantilis et al. (2002) have developed a spatial decision support system (SDSS) in order to solve the open vehicle routing problem (OVRP). The OVRP deals with the problem of finding a set of vehicle routes, for a fleet of capacitated vehicles to satisfy the delivery requirements of customers, without returning to the distribution center. The system is a spatial DSS (SDSS) that maps customers locations to vehicles, assigns customers to vehicles, determines the best sequence of deliveries for each vehicle so that all customers are serviced and the total distance traveled by the fleet is minimized. They used Dijkstra's algorithm in C++ and ArcView as a GIS tool.

Yu et al. (2003) have improved conventional algorithms for roadway planning by including the considerations of spatial distances, anisotropic costs and the presence of bridges and tunnels in the paths. Instead of finding the spatial distance on two-dimensional plane, it is found on three-dimensional plane by considering the horizontal and vertical distance between two adjacent vertices. Coming to anisotropic costs, since terrain surfaces are complex, slopes in different directions are not constant. Because of this reason, anisotropic costs accumulation method is used in this study. This method enables to calculate the cost for different directions individually. The bridges and tunnels are also used in cost accumulation. The algorithm, which is named as Smart Terrain, determines the need for a bridge or tunnel. It is based on the following concept: firstly, it determines the counter lines of the terrain. Then, if the straight connection between vertex A and one non-adjacent candidate vertex intersects the contour line only twice, this connection may be a bridge or tunnel. Based on this concept, it determines the bridges and tunnels between the starting vertex and ending vertex on a raster map. The Smart Terrain algorithm was tested on two small mountainous regions in Venango County of Pennsylvania, USA. The program is written in Java and ArcView is used for data visualization and mapping. According to the test results, it can be said that the Smart Terrain algorithm produces more realistic least-cost roadways compared to the conventional algorithms.

Lanthier et al. (2003) have implemented parallel implementation of shortest path algorithms. The terrain data is used as source map. In this approach, the source map (vertices and edges) is divided to the processors of computer. Each processor searches the nodes in its partition and calculates the path for its partition. The results are shared between processors by using a shared memory between processors.

The overall run time in this method, is the sum of the run times of:

- Initialization time,
- Session initialization time,
- Compute time,

- Idle time,
- Communication time.

The sum of the compute time, idle time and communication time is the total time it takes to a processor to find the path for its partition. Initialization time and session initialization time is for distributing the partitions to each processor. Since more than one computer is used in order to calculate the shortest path, the run time of parallel implementation will be less than sequential processing. If we think that the algorithm is run over p processors and using Dijsktra's algorithm then time complexity of the algorithm will be $O(n^2 / p)$.

After terrorist attacks at the World Trade Center (WTC) in New York City and the Pentagon on September 11, 2001, the researchers have focused on integrating the Intelligent Transportation systems (ITS) and Intelligent Building Systems (IBS) by using an operation center. Kwan and Lee (2003) have implemented a GIS-based intelligent emergency response systems (GIERS) that aim at facilitating quick emergency response to terrorist attacks on multi-level structures (e.g. multi-story office buildings). This system includes a navigable 3D GIS, a real-time geographic database, a suite of decision support functionalities, and a distributed information architecture that is implemented through wireless and mobile communications technologies (The system is implemented in Visual Basic). By using various types of sensors, route condition and traffic delays are sent to GIERS in real-time by Intelligent Transportation system (ITS). Similarly, some sensors are also used for Intelligent Building Systems (IBS) such as temperature and light-level detectors, movement or occupancy sensors, pressure pads, smoke or gas detectors, and fire detectors. So the condition of the route from dispatching location of the rescuers to the destination floor is known in real-time. In this study, it is intended to reduce the time delay over the route from dispatching location of the rescuers to the destination floor. This time delay is divided into three levels of uncertainty: (a) road network uncertainty; (b) entry point uncertainty; and (c) route uncertainty within a building. As an experiment, a study area is selected in downtown Columbus, Ohio (USA), located in the east of Scioto River. It is assumed that a 250-pound high-explosive bomb exploded on the 42th floor of Franklin County Municipal Building. The result of

this experiment is shown in Figure 2.5. So, by using real-time data and shortest path algorithm that evaluating weights of the nodes, the time delay in 911 calls are significantly reduced.

Source of uncertainty	Travel time without using GIERS (in min)	Travel time using GIERS (in min)	Delay (in min)
Road network	6.26	4.08	2.18
Entry point	3.73	0.00	3.73
In building	29.84	20.11	9.73
Total travel time or delay for $Route(a, c)$	39.83	24.19	15.64

Figure 2.5 Travel times with and without using GIERS (Kwan and Lee, 2003)

The hazmat transport models differ from other transport models by the following factors incident probability and population impacted. Kara et al. (2003) have proposed two methods in order to deal with the selection of the minimum risk path for transport of a hazardous material. One of the proposed procedures is a modified version of a well-known shortest path algorithm, and the other is an adaptation of a link-labeling algorithm developed for urban transportation. Firstly, incident probability is considered. Incident probability of a path is computed by adding the incident probabilities along each link of that path. The probabilities of incident on a given link, depends on the incident probabilities of all link leading up to that link. Kara et al. (2003) have proposed an extension of Dijsktra's node-labeling shortest path algorithm to find a minimum incident probability path. The algorithm adjusts the link probabilities by multiplying them with the probability of safely arriving at the starting node of the arc. Secondly, they have tried to find the population impacted. The impact area of an incident is usually assumed to be a circle with a substance-dependent radius centered at the incident location. In order to find the path with a minimum number of people that are impacted from incident, a link-labeling shortest path algorithm is used. While computing the path, overestimation is inevitable in the nodes that connect two arcs. Kara et al. (2003) used rectangular exposure zones in place of semicircular exposure zones. By using these methods, the error ratios are significantly reduced when rectangular exposure representation is selected.

Breunig and Baer, (2004) state that today's first commercial database management systems for mobile devices do not support spatial database queries. For this reason, they collect requirements and present a first implementation prototype of a mobile route planning system focusing on the support of spatial database queries. They presented the geo-extensions of an XML database management system and an OODBMS, respectively, in an application of a mobile bicycle route planning system. Extensions of Tamino DB (XML DBMS) and Objectivity/DB (OODBMS) have been implemented and evaluated by local bicycle routing software. The Objectivity/DB (OODBMS) extension provides better results in means of time performance of routing operation over XML DBMS.

2.4 Dijsktra's Algorithm With Priority Queue Implementation

Main idea of Dijkstra's algorithm is to maintain a set "S" of vertices whose final shortest path from the source have already been determined. The algorithm repeatedly selects a vertex "u", which is not in set "S". If this vertex has a minimum shortest path cost estimate then adds "u" to "S" and finds the neighborhoods of vertex "u". This process is repeated until all of the vertices have been searched. At the final step, a shortest path is found from source vertex to the goal for which the sum of the cost of the vertices on the path is minimum. The time complexity of this algorithm is $O(V^2 \log V)$.

Route planning can be used in GIS to develop several types of applications, e.g.; vehicle routing systems, ambulance management systems, underground supply power system, movement of troops etc. These applications require real time interaction with the user. Time is a critical parameter for these kinds of applications. In real time systems, system must respond in a pre-defined time limit. GIS applications are very critical in means of response time. For example, in ambulance management system, respond time must be very short since some of the roads may be closed to traffic and it may be necessary to find another path as soon as possible. Because of these reasons, an efficient shortest path implementation is necessary for GIS applications. Most of the GIS applications uses Dijsktra's shortest path algorithm. It is intended to implement a fast shortest path algorithm in this study. A modified version of Dijsktra's algorithm is used for this

purpose. It decreases the time complexity from $O(V^2 \log V)$, which is the output of the Dijsktra's algorithm, to $O(E \log V)$. Here, "V" stands for number of vertices over source map and "E" is used for the number of edges between vertices. Figure 2.6 shows the time complexity of the Dijsktra's algorithm with priority queue implementation according to different type of priority queues.

		Heaps			
Operation	Linked List	Binary	Binomial	Fibonacci *	Relaxed
make-heap	1	1	1	1	1
insert	1	log N	log N	1	1
find-min	N	1	log N	1	1
delete-min	N	log N	log N	log N	log N
union	1	N	log N	1	1
decrease-key	1	log N	log N	1	1
delete	N	log N	log N	log N	log N
is-empty	1	1	1	1	1

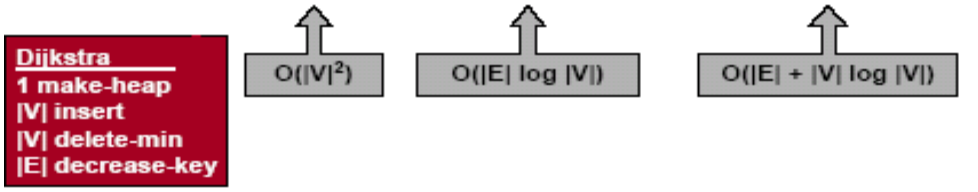


Figure 2.6 Time complexity of Dijsktra's algorithm according to the type of the priority queue, (Wayne, 2002)

The Dijsktra's shortest path algorithm with priority queue implementation grows a shortest-path tree from the source (Saunders and Takaoka, 2001). Every vertex maintains a shortest-path estimate and parent that indicates the final edge of a path with this estimate. At each step, add the node with the smallest estimate to the tree via the edge to its parent. Use a priority queue to manage extracting the node with the smallest shortest-path estimate. Dijkstra's algorithm with priority queue implementation and an exercise of it are presented in Appendix B.

The main difference in Dijkstra's algorithm with priority queue implementation is the usage of priority queue. The least-cost path estimations are stored in a queue, which is in priority queue implementation. Then the minimum estimate is selected from the queue. For every estimate, the parent vertex is also held.

There exist two kinds of priority queue implementations in the literature (Breymann, 2002). They are:

- Binary Heap
- Fibonacci Heap

For this study, binary heap method is selected. Binary heap is a modified version of binary tree structure. It is an "almost" complete binary tree as seen in Figure 2.7 (Because leaf nodes are filled from left to right). A *complete binary tree* is a tree in which all leaf nodes are at the same level and all internal nodes have degree 2. A *leaf* node is a node with no children (i.e., both children are empty binary trees) and the *degree* of a node is the number of children that node has; it is 0, 1 or 2. (Note: In this section, node is used for an element of the tree)

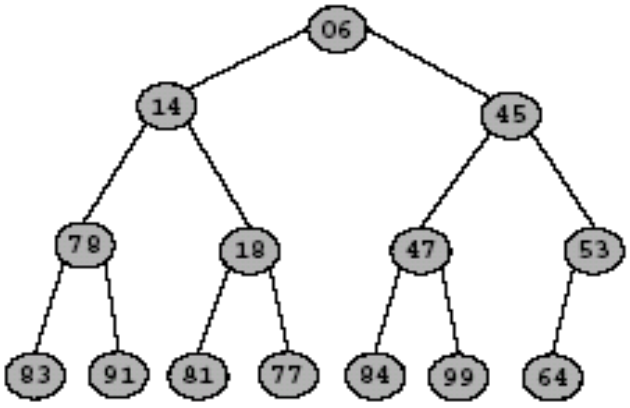


Figure 2.7 A sample binary heap

A binary heap can be conveniently represented in array structure. The first element in the array is the root node of the tree. Each node of the tree has a key value. The first element (root element) in the array has minimum key value. Key values in binary heap correspond to the cost of the vertices. The Figure 2.7 shows a sample of the binary heap tree.

Binary heaps can be represented in array structure by writing down its elements from left to right, top to bottom. The sample binary heap that is shown in Figure 2.7 is stored in an array structure as shown in Table 2.3.

Table 2.3 Array representation of the binary tree

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Key	06	14	45	78	18	47	53	83	91	81	77	99	64

The first element of the array, at index 0, is the heap's root (06, in this example). In binary heap algorithm, each element has a parent element and left, right childs. To get from the i'th element to its left child, simply multiply i by 2 and add 1; to get to its right child, multiply by 2 and add 2; to get to its parent, subtract one and divide by 2 and discard any remainder. For example, the element with key value 18 has an index value of 4. The index of the parent element is found by (4-1) / 2 = 1. So, the element with key value 14 is the parent. The left child of this element ha an index value of 4*2+1 = 9. The element with key value 81 is the left child. As seen clearly, the element with index 10 and key value 77 is the right element.

All of the operations can be done as long as binary heap property is true. The operations on binary heap, that is required for Dijsktra's algorithm, can be:

- Deleting the node with minimum key value (root of the tree)
- Insert a node
- Decrease the key value of the vertex
- Search if the heap is empty or not.

After each operation, the heap property is checked. If the heap property is valid than the result of the operation is successful.

i. Heap Property

For every vertex except the root, the key value in its parent vertex is less than or equal to its own key value. In Figure 2.7, the parent of each node has a key value that is less than the node's key value. For example, the node with key value 18 has a parent node with key value 14 which is also has a parent of key value 6. So, the heap property is valid for this binary heap. In Figure 2.8, the heap property is not valid. Because, the parent of node with key value 53 is the node with key value 42.

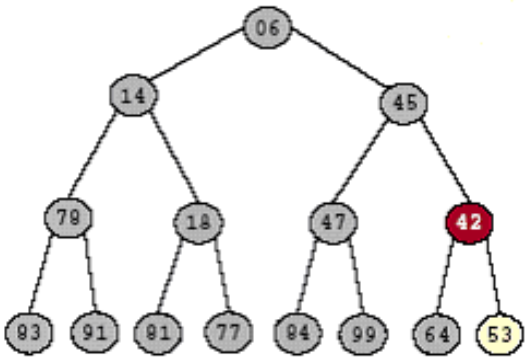


Figure 2.8 The heap property of this binary heap is not valid.

ii. Deleting Minimum

Delete node with the minimum key value from the heap.

- The vertex with the minimum key value is extracted.
- The root node is deleted and returned.
- The last node is copied in place of the root node.
- The heap property is set.

Figure 2.9 shows a sample of delete minimum operation. The time complexity of this operation is O(logV).

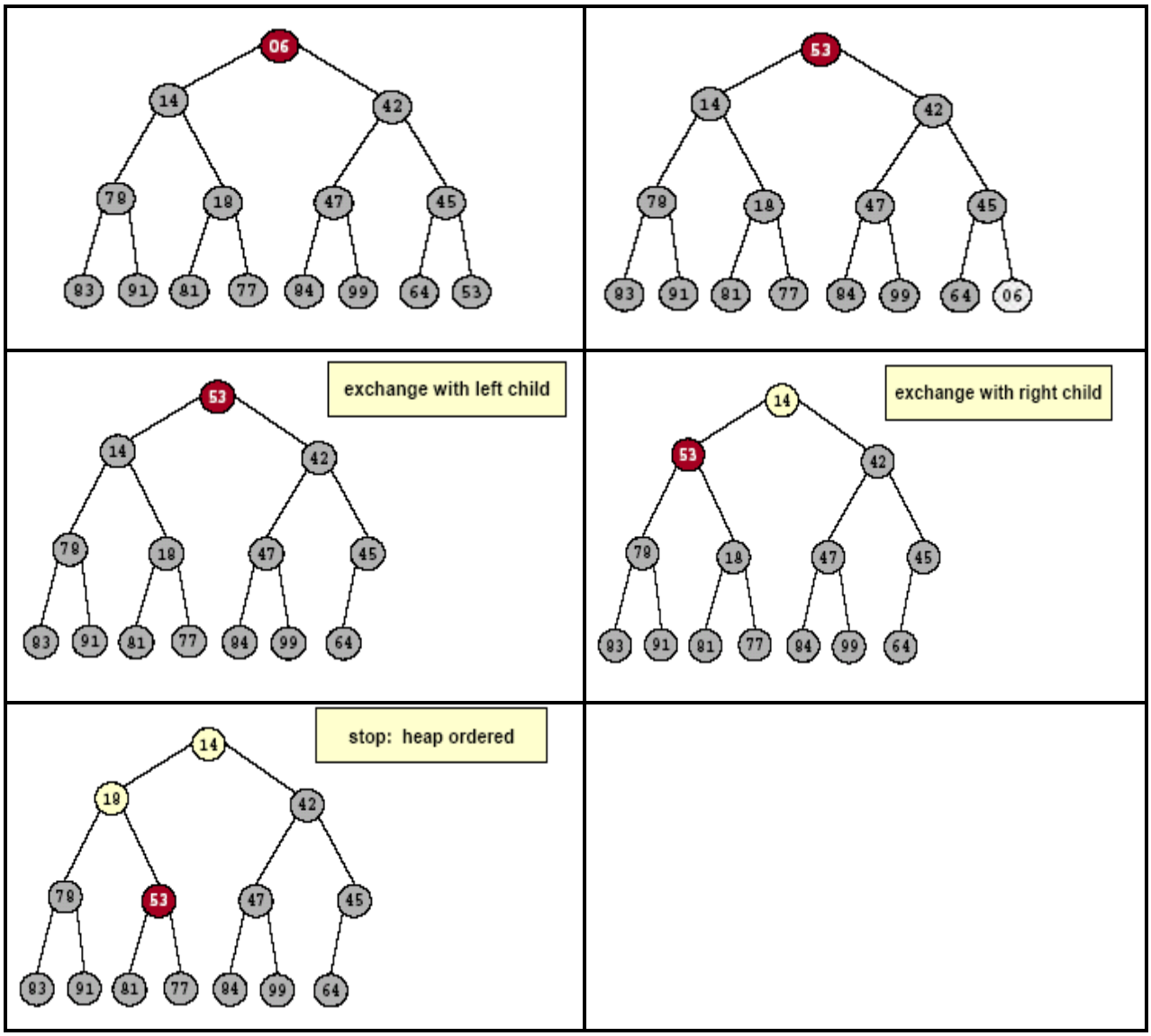


Figure 2.9 Steps of delete minimum operation

iii. Insert Node

Insert element x into heap.

- Insert into next available slot.
- Compare this element to its parent. If it is smaller than its parent, swap this element with its parent element. Do this operation until the root node is reached or the parent node is higher.

Figure 2.10 shows a sample of insert node operation. This operation takes O(logV) iterations.

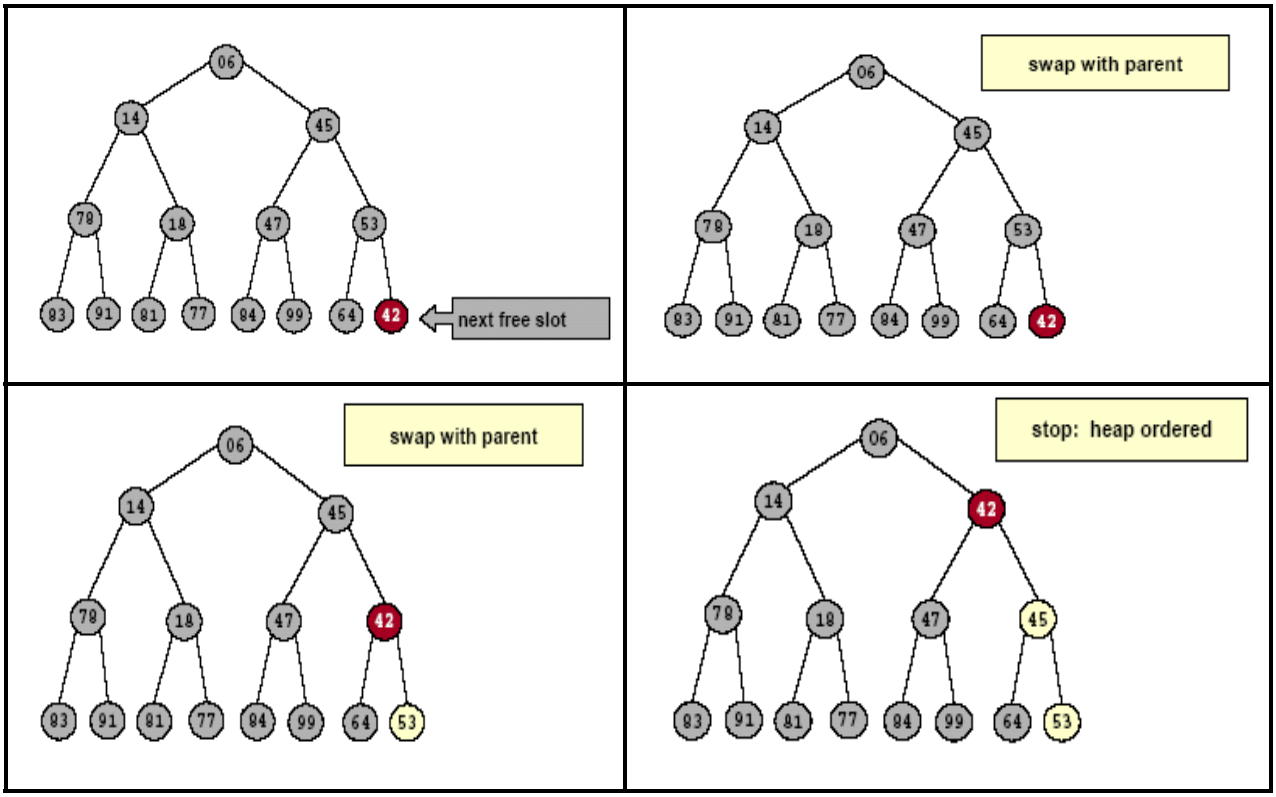


Figure 2.10 Steps of insert node operation

iv. Decrease Key Value of A Node

Decrease the key value of the node.

- Decrease the key value of the node
- Compare this element to its parent. If it is smaller than its parent, swap this element with its parent element. Do this operation until the root node is reached or the parent node is higher.

For example, if we reduce the key value of element 45 to 35, the key of element 45 is set 35 and it is swapped with its parent (element 42) according to the rules presented above. The time complexity of this operation is O(logV).

CHAPTER 3

IMPLEMENTING THE ALGORITHM

3.1 GIS Layers and Analysis

The purpose of this study is to increase the performance of a GIS application by replacing the shortest path algorithm. In order to achieve this goal, the previous study (Ünlü, 2002) is considered and the data used in that study is also used in this study to make the comparison on the performance of the applied algorithm.

Table 3.1 shows the speed performance of the Dijsktra's algorithm over different number of nodes. While the number of nodes increases, the speed performance of the algorithm decreases rapidly. As the GIS and Remote Sensing Technologies are improved, the amount of GIS data and detail information gathered are increased and the resolution of the remotely sensed data is getting higher. Hence, the number of grid vertices used in shortest path analysis are increasing. It is required to have an algorithm that finds the least-cost path (shortest path) with less time complexity. For this reason, Dijsktra's algorithm with priority queue implementation is used in this study. Except for this, the quality of the shortest path is also significiant. In this study, it is intended to decrease the time complexity of the algorithm while protecting the quality of the shortest path.

A commercial GIS software, namely ArcGIS can also be used in order to find the least-cost path. The time performance of the spatial Analyst Tool of ArcMap is given in Section 3.3.1. Although the advantage that is provided in means of time performance, this program suffers from the difficulties of the user interface. Because; in order to specify a start point and a goal point, two different layers must be created. Then, the cost direction and cost distance methods are run over raster data in order to find the least-cost path. In this study, it is intended to provide an easy use program with the efficiency in the time performance.

Table 3.1 Test runs (Ünlü, 2002)

Computer Configurations	3000 nodes	10.500 nodes	11.600 nodes
Pentium-1 (233MHz) 64Ram	15 sec	159 sec	235 sec
Pentium-2 (897MHz) 256Ram	2 sec	18 sec	43 sec

In this section, the concept of the application and the enhancements applied to the application in order to satisfy the needs of this study, are explained.

3.1.1 **Data Layers**

Data layers refer to the various overlays of data, each of which normally deals with one thematic topic. GIS uses data layers to control how many features are displayed at one time. In this study, all of the data layers are in MapInfo Tab format with a projection of Universal Transverse Mercator(ED50) Zone 37. The data layers cover an area of 157 km2. Table 3.2 shows the data layers used, where, each data layer has an attribute. For river, bridge and slope distribution layers, the attribute values are used in the overlaying operation. It will be detailed in Section 3.1.2.

Table 3.2 Data layers used in this study

Layer Name	Column	Value	Unit	Is Attribute Overlaid[1]	Description
Motorway	width	40	meter	No	It covers the width of the highways.
Road	width	15	meter	No	It covers the width of the roads.
Bridge	width	min:2 max:10	meter	No	It covers the width of the bridges.
River	width	min:0.5max:10	meter	Yes	It covers the depth of rivers.
Slope dist.	slope	min:5 max:90	percent.	Yes	It covers the slope of the terrain.
Dam				No	
Railroad				No	
Village				No	
Tumulus				No	
High voltage				No	

All of the layers are in vector data model except the slope layer, which is in raster data model. In order to use overlay capability of MapInfo, all of the data layers that have vector data model are converted to the polygons. To convert the data layers of arcs (lines) to polygons, buffering operation is performed. A buffer is created that defines all the areas, which are within some specified distance from the point or line object. By using buffering operation, the line and point objects are converted to areas (polygons). The shortest path algorithm is running over grid data, which is explained in Section 3.1.3. Since, spatial queries are used to find the intersection of vertices (grid point) with every vector layer one by one; all of the vector layers should be in polygon shape. If a vector layer includes point and line objects, it is not possible to intersect them to a vertex. Therefore, buffering operation is used to convert vector layers (motorway, road, bridge and tumulus) that contain point or line objects to polygons. Each data layer is explained by the help of the figures that are derived from the application, which is prepared by Ünlü (2002).

[1] If the attribute of the layer is used in overlaying operation then it is shown by “Yes”, otherwise “No"

i. Motorways

This layer has line objects. Because of this reason, a buffer is created for this layer with a distance of 50m. Motorway is the best region to pass through according to the scenario used by Ünlü (2002).

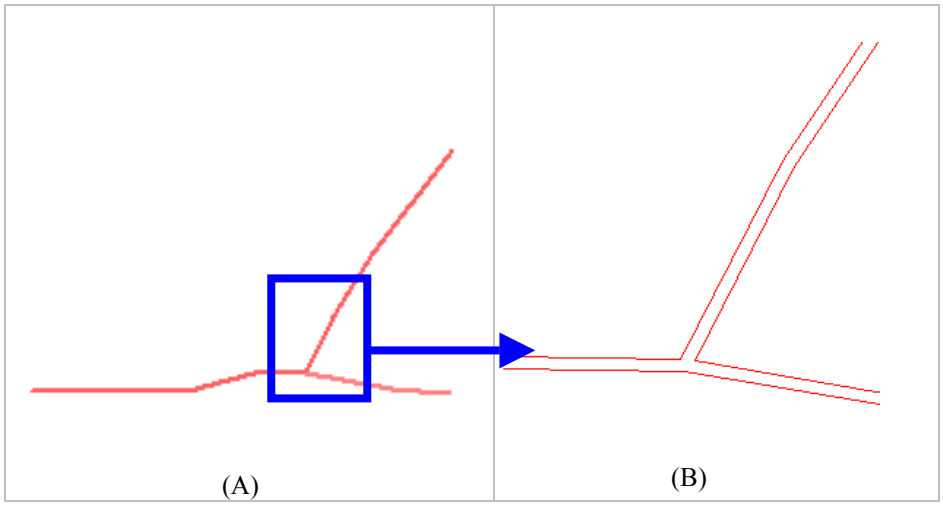


Figure 3.1 Motorway (A) and buffered motorway (B) layers used in this study

ii. Roads

This layer also has line objects. Because of this reason, a buffer is created for this layer with a distance of 50m. The region covered by roads is good places to pass through according to the scenario used by Ünlü (2002).

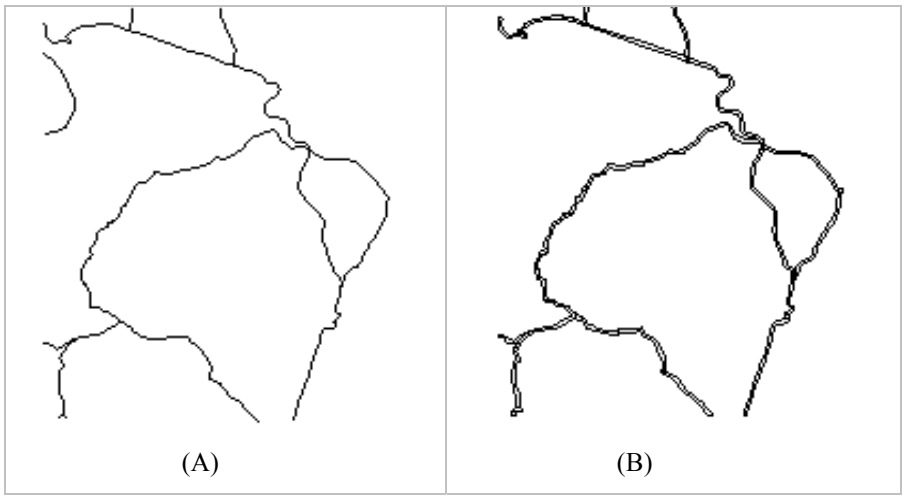


Figure 3.2 Roads(A) and Buffered roads (B) layers used in this study

iii. Bridge

This layer has symbols, which indicate the bridges, and line objects. Because of this reason, a buffer is created for this layer with a distance of 50m. Bridge is not a good region to pass through according to the scenario used by Ünlü (2002). If motorways or roads exist as an alternative, these regions should be selected; otherwise bridge can be selected as passageway.

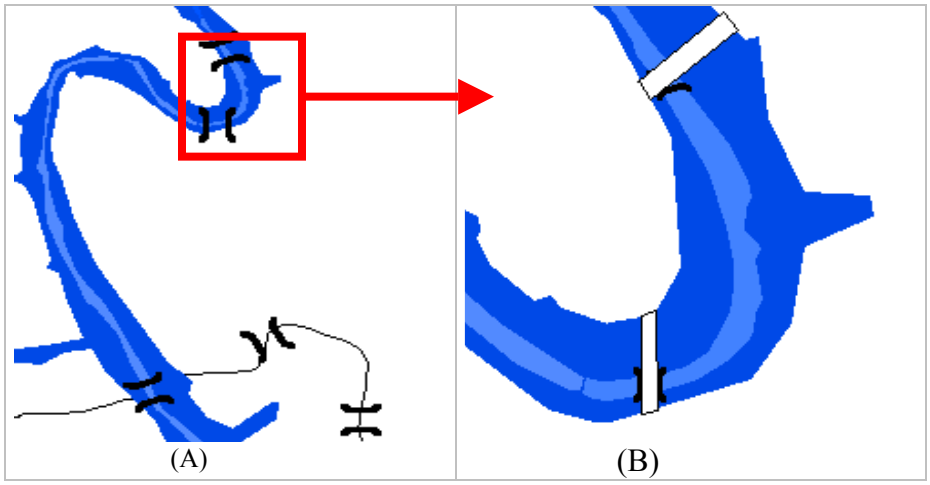


Figure 3.3 Bridge (A) and Buffered bridge (B) layers used in this study

iv. Tumulus

This layer has point objects. So, buffering operation is done with a distance of 50m. According to the scenario used by Ünlü (2002), the objects in buffered tumulus layer are forbidden to pass through.

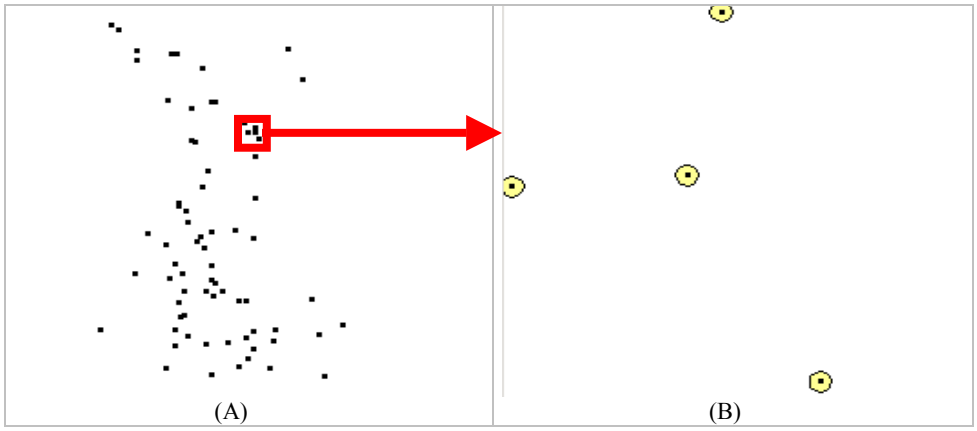


Figure 3.4 Tumulus(A) and Buffered tumulus (B) layers used in this study

v. River

This layer has an attribute; depth, which is used in, overlay operations. According to the scenario used by Ünlü (2002); if the depth of the river is less than 50 cm then this river is a bad place as a passageway. But, it can be passed through. If it is between 50cm and 1m then this is the worst place as passageway. Otherwise, it is forbidden to pass through.

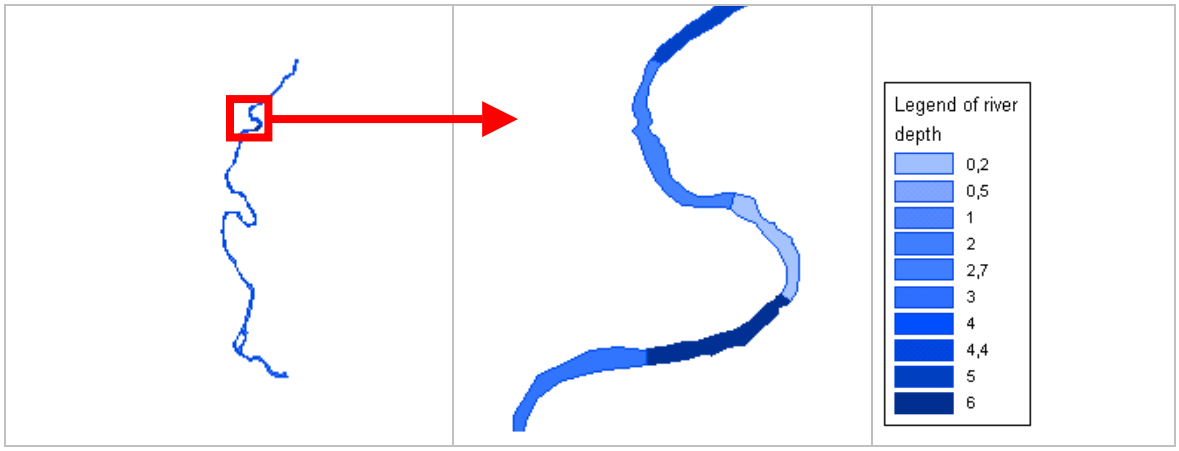


Figure 3.5 River layer used in this study

vi. Slope Distribution

This layer has an attribute slope percent, which is used in overlay operations. According to the scenario used by Ünlü (2002); if slope is between 0 and 10, this is partially good region as passageway. If it is between 10 and 30, it is not a good region as a passageway. Finally if slope is between 30 and 60, this is a bad place to pass through. Otherwise, if slope is higher than 60, passage from this region is forbidden.

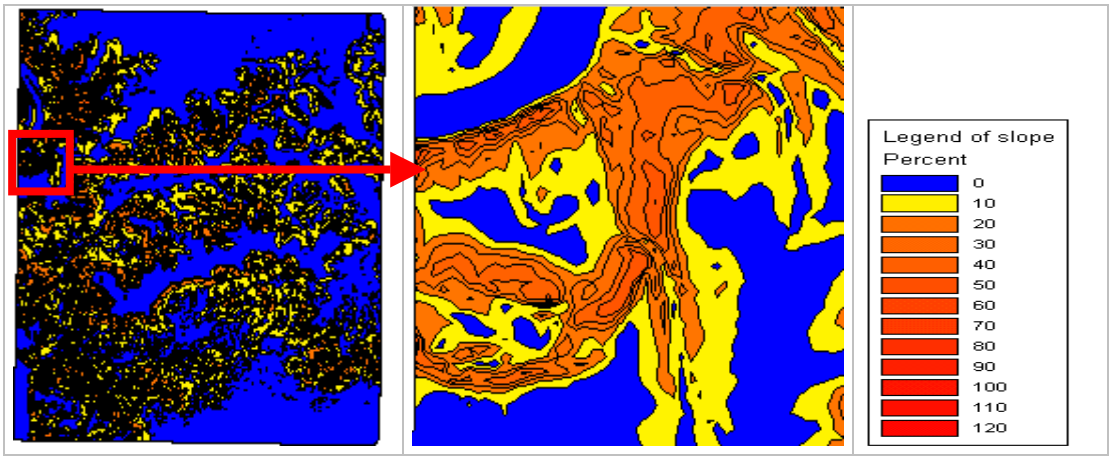


Figure 3.6 Slope distribution layer used in this study

vii. Dam, Village, Railroad and High Voltage

These layers are forbidden to pass through according to the scenario used by Ünlü (2002). The reasons are:

- As the name implies, passing through dam is impossible
- Village is forbidden because military troops cannot go through cities or town to the assembly area.
- Vehicles cannot go through railroad.
- High voltage areas are dangerous to pass through.

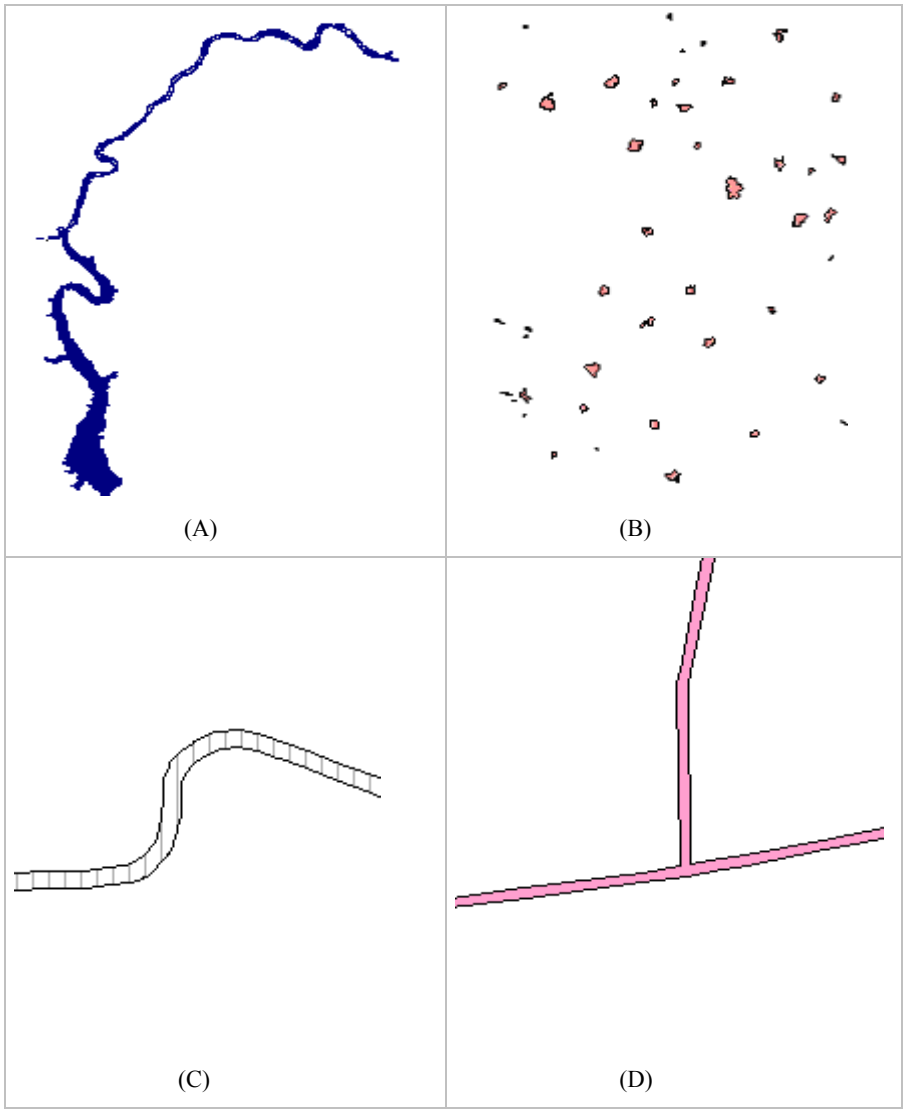


Figure 3.7 Dam (A), Village (B), Railroad (C) and High Voltage (D) layers used in this study

3.1.2 Overlay

Overlay is conceptually finding geographically coinciding areas that match the required criteria from different layers of information put on top of each other. In other words, the integration of different data layers involves a process called overlay. The GIS overlays that produce a new data outcome are more complex and require a range of spatial operations that depend upon the data structure and methodological design. There are two kinds of overlay operations (Heywood et al., 1998).

i. Vector overlay

Methodologically and technically vector overlays are more complex than raster overlays. Geometry is used to define new objects in a topological sense. The three types of vector overlay are: point-in-polygon, line-in-polygon, and polygon-on-polygon (Clarke, 1999).

ii. Raster Overlay

Raster overlays are performed using map algebra (mathematics). The mathematical operations are performed on values of overlaying pixels from different layers. Adding, subtracting, multiplication and other operations can be performed in order to produce the output values that are normally saved into a new layer.

In this study, overlaying operation is performed on raster map (The code written for overlay analysis is given in Appendix F). The raster map contains vertices that are evenly distributed over the study area. By using overlaying, the geographical phenomenon (elevation, land use, etc.) can be best reflected over each vertex. Each vertex is assigned a cost in order to represent the geographical property of the area, which is represented by this vertex. Hence, the shortest path algorithm runs over raster data. Figure 3.8 shows the flowchart of the overlay analysis.

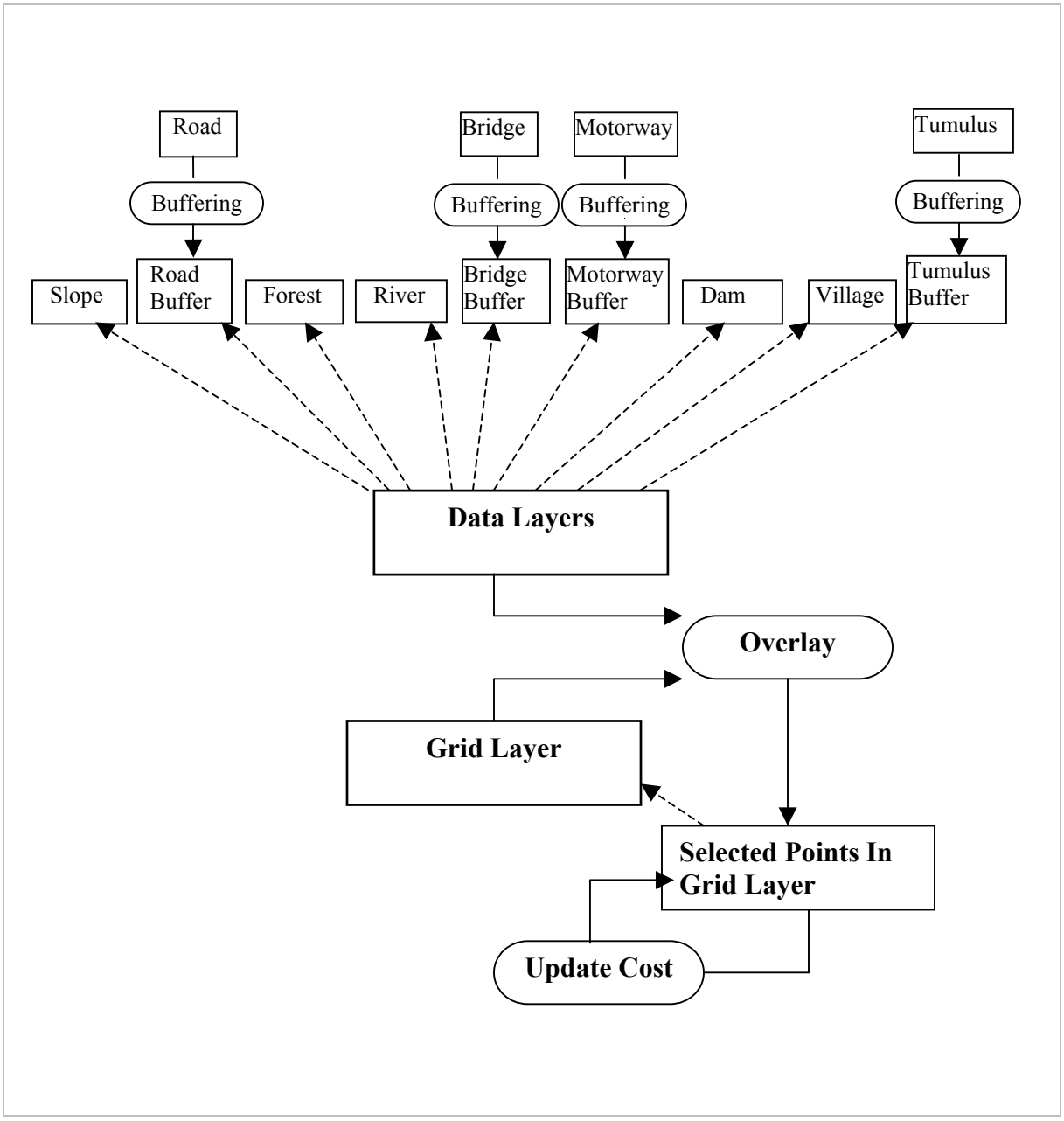


Figure 3.8 Flowchart of the overlay analysis

In overlay analysis; all of the layers are searched for each vertex. If the vertex “A” (in grid map) intersects to a region from data layer “B”, then the cost of the data layer “B” is set to the cost of the vertex. Table 3.3 shows the cost criteria for each layer. The river and slope layers have more than one alternative. For example, if the depth of the river is less then 10m then the cost of the vertex, which is inside of this area, is “5”. If the depth of the river is more than 10m, then the cost of the vertex is “9”.

The cost values from “1” to “5” represents the regions that can be passed through. “1” is assigned to the vertices that represents the areas that are best places to pass through. “5” is for the places least eligible to pass through. The cost values through “1” and “5” represents the places from most eligible to the least eligible one. “9” represents the places that cannot be passed through or forbidden as a passageway.

In this study, overlay operation is updated. Because, the shortest path algorithm can be run for different types of applications. For example; navigating military troops, setting up the routes of the electric lines or pipelines, tracking, etc. In order to use this study for different range of scenarios, the overlay logic is enhanced. If a point (in grid map) intersects with more than one layer, than the cost of this point is determined according to the following rules.

- If there are at least one layer that have a cost values of “9”, then the cost value of this point is “9” regardless of the other layers. This rule has an exception for only bridge layer. Because, bridges are over rivers and dam, which are usually given a cost value of “9”.
- If the cost value of all of the layers, which are coinciding with the searched point, are between “1” and “5”, then the least-cost value is assigned to the searched point.

Table 3.3 Overlay criterion (Ünlü, 2002)

Layer Name	Criterion	Cost	Category
Motorway	All	1	Very Good
Road	All	2	Good
Bridge	>5	3	Partially Good
River	<=0.5	4	Partially Good
	0.5< >=1	5	Partially Good
	>1	9	Bad
Slope	<10	3	Partially Good
	10-30	4	Partially Good
	30-60	5	Partially Good
	>60	9	Bad
Dam	All	9	Bad
Railroad	All	9	Bad
Village	All	9	Bad
Tumulus	All	9	Bad
High voltage	All	9	Bad

3.1.3 Grid Layer

Grid layer is constructed from a set of vertices. The distance between two vertices is related to the resolution of the grid map with reverse ratio. For the same study area, if the resolution of the grid map increases, the number of points (vertices) increases. Since buffering operation is performed in a radius of 50m, the cell size of the grid map can not exceed 100m. Figure 3.9 shows a sample grid layer. Each grid location in grid map corresponds to a vertex. The directions that leaving a vertex and entering to another vertex are called as edges. The shortest path algorithm is run over grid data. This algorithm finds the path and returns the vertices, on which the shortest path goes through.

In this study, the grid layer is named as “gridharitasi1.tab” (The code written for preparing grid map is given in Appendix E). While preparing grid map, at first the “gridharitasi1.tab” layer is deleted. Then, this layer is created again. So, creation of a new grid layer is guaranteed. Each grid point has a number value, which identifies the grid. Grid numbers are starting from zero. This enables the shortest path program (The program that runs shortest path algorithm) to convert the indices between one and two dimensional grid arrays easily. Grid map is hold as one-dimensional array in the MapBasic program segment, which is the application part. But, it is stored on a two dimensional array in the Visual C++ program (Deitel, 1992), which is containing the Dijsktra’s shortest path algorithm with priority queue implementation. Hence, starting grid numbers from zero enables to locate the same grid vertex in one and two-dimensional arrays easily (The code written for finding shortest path is given in Appendix G).

Figure 3.9 Grid table containing the indices

3.1.4 Program Architecture

In this study, the data layers are prepared in MapInfo. As shown in Appendix A, a toolbar is prepared in MapBasic in order to perform several GIS operations and provide a user interface. The following GIS operations are performed in MapBasic program.

- Preparing Grid Map (Creating grid pixels)
- Overlay Analysis

The MapBasic program also presents a user interface. User can select the starting point and ending point by using MapBasic program. When a path finding operation is requested by user, the MapBasic program writes the grid data, starting and ending grid points to a file and calls the program which runs the shortest path algorithm (This program is written in Visual C++). The sequence diagram (Schmuller, 1999) of the software is shown in Figure 3.10.

3.1.5 Program Usage

The user manual of the application is given in Appendix A. The icons "Save Symbol" and "Save Path" are added to the application, in this study. "Save Symbol" icon is used to save the symbols on the symbol layer to the disk. Similarly, by using "Save Path" icon, the shortest path is saved as well. These icons are necessary, because two applications are compared in means of performances and cost estimation metrics. For example, the path found by the application that uses Dijsktra's algorithm is copied to the disk in order to open the same path on the other application (Dijsktra's algorithm with priority queue implementation).

i. Creating Grid Map

By clicking the "Prepare Grid Map" icon[2], a rectangle is drawn on the study area. After determining the end point of the rectangle, the "Grid Menu" dialog box appears. The cell

[2] All of the icons are given in Figure A-1 in Appendix A.

size of the grid map will be entered to the text box in “Grid Menu” window. We assume that the cell size is 100m. Then, program starts to produce the grid map. Figure 3.11 – 3.12 shows the creation steps of the grid map and Figure 3.13 shows the created grid map (“gridharitasi1” layer with “number” column set).

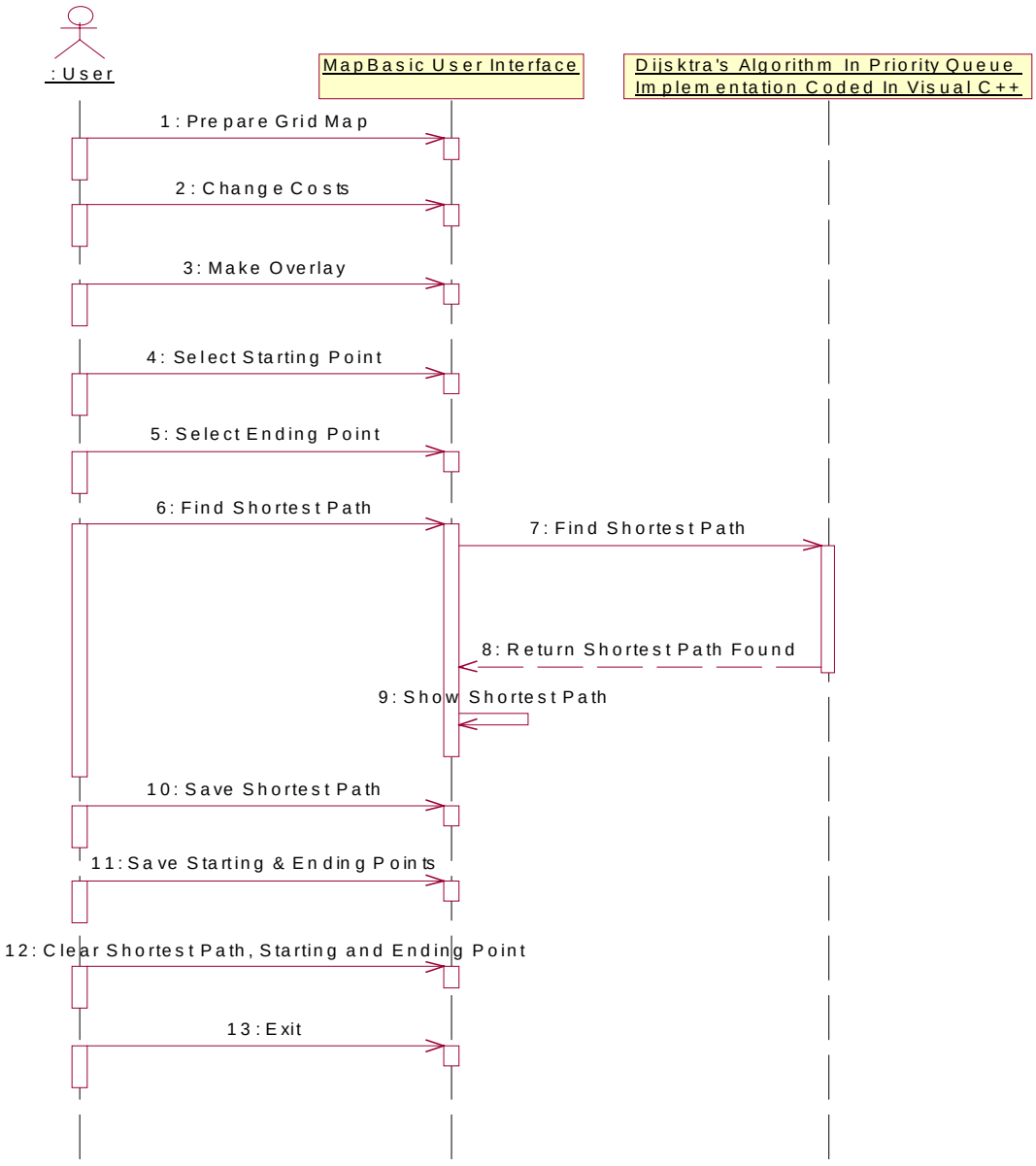


Figure 3.10 Sequence diagram of the application

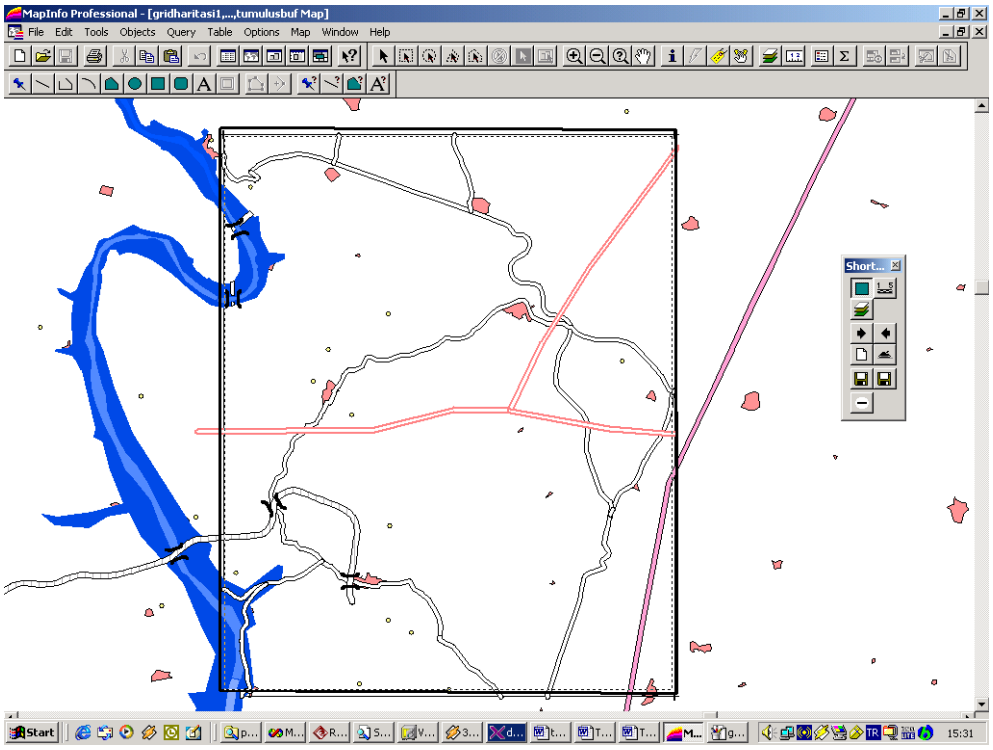


Figure 3.11 Select regions for grid layer.

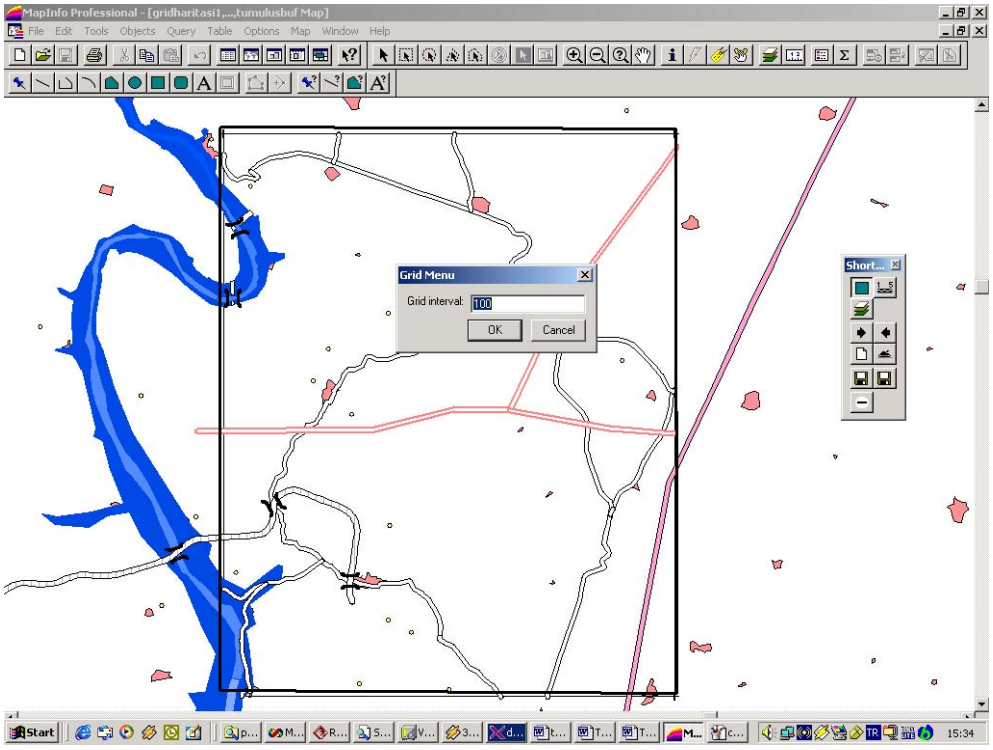


Figure 3.12 Entering grid interval

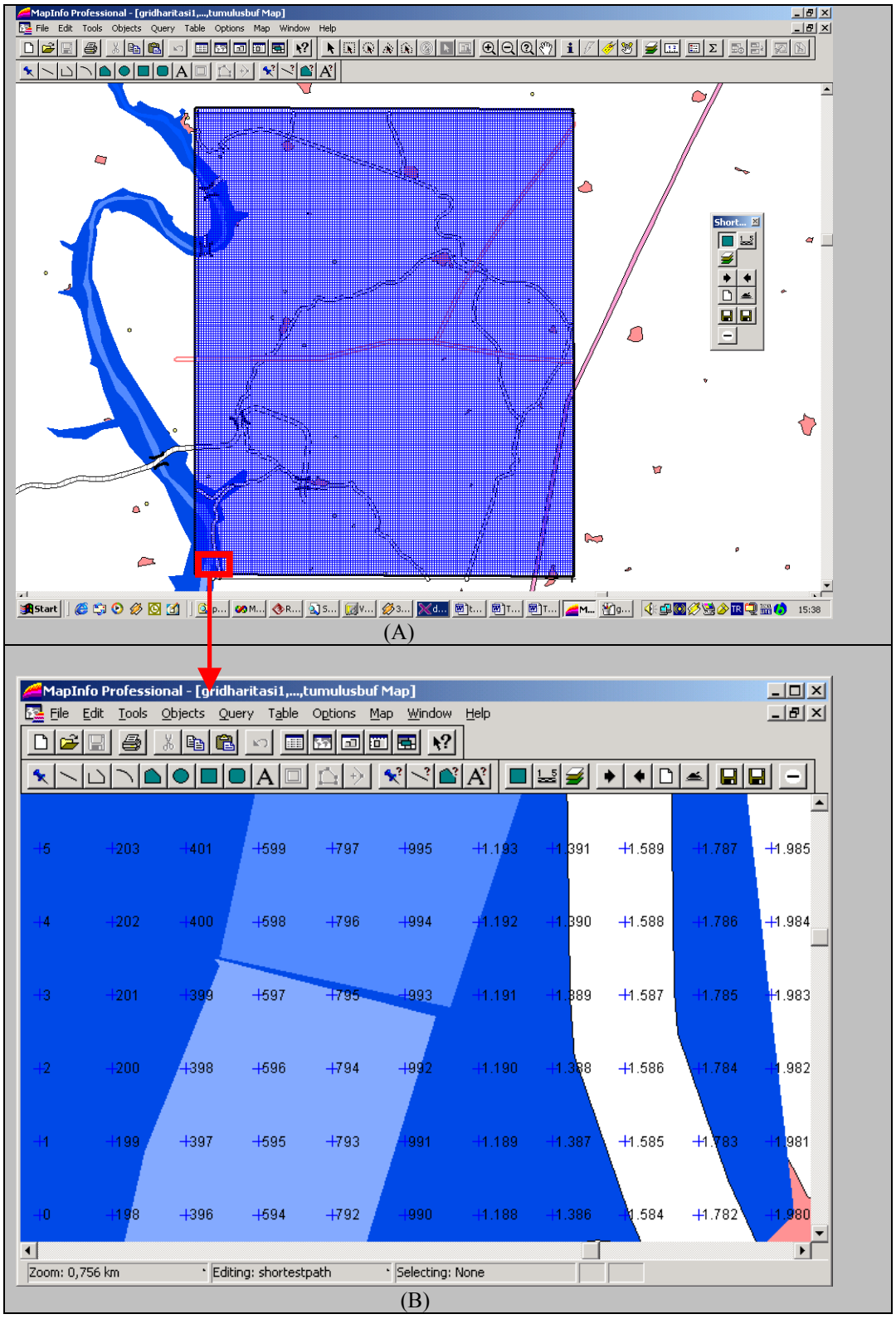


Figure 3.13 Created grid layer (A) and detailed representation of grid layer (B).

ii. Running Overlay Analysis

By using the “Change Cost” window, user can assign the cost according to the needs. Then, he or she presses the “Overlay Analysis” icon. The “Overlay Analysis” window will be shown. Select the type of the overlay, “Road & Terrain” or “Road”. The overlay analysis operation lasts for a few seconds. The preparation steps of the overlay analysis are shown in Figure 3.14 and the result of the overlay (“gridharitasi1” layer with “cost-value” column set) is in Figure 3.15.

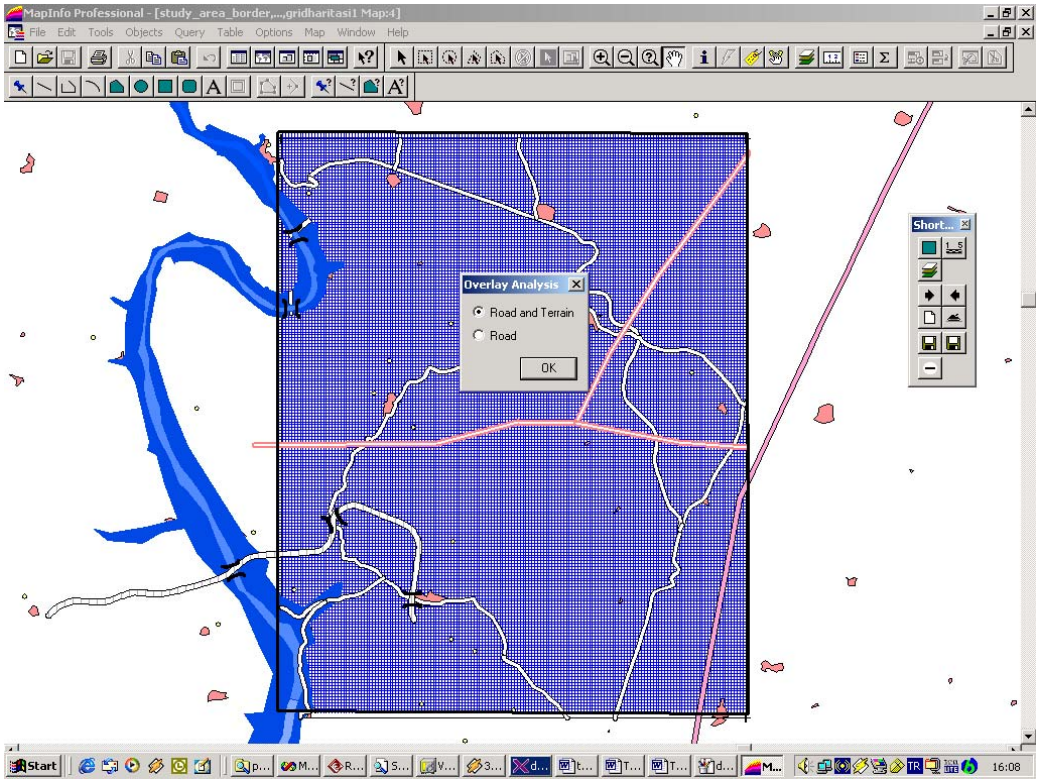


Figure 3.14 Overlay analysis menu.

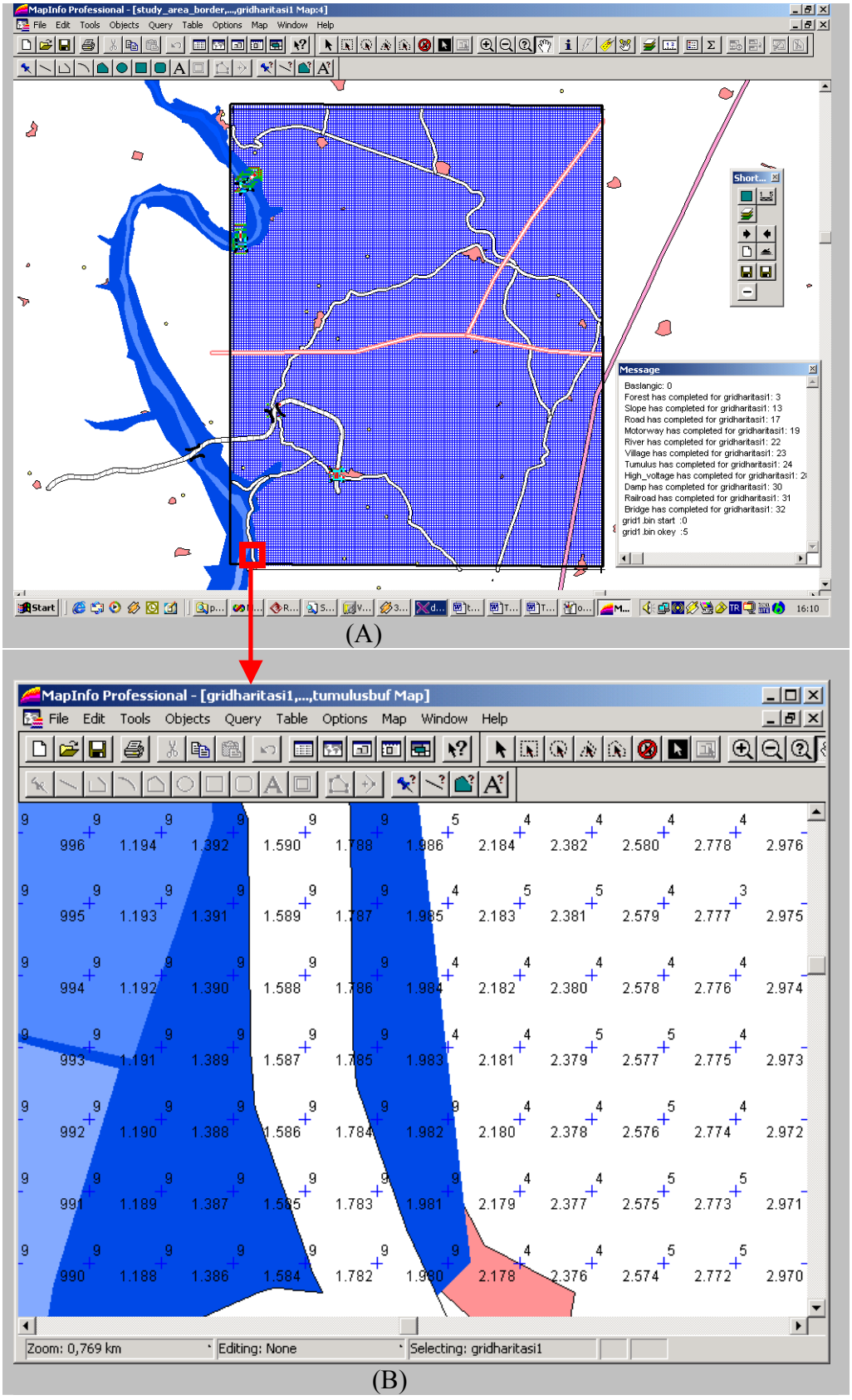


Figure 3.15 The gridharitasi1 layer with the cost values after overlaying operation

iii. Finding Shortest Path

At first, user selects the starting point by using the “Start Point” icon. This icon draws an ellipse over the region that user selects. Then, a symbol is placed on top of a point in that region. The ending point is also selected similarly. Finally, “Draw Path” icon is pressed in order to find the shortest path. The path found is shown in red line. Figure 3.16 shows the selected start and end points, Figure 3.17 shows the shortest path found.

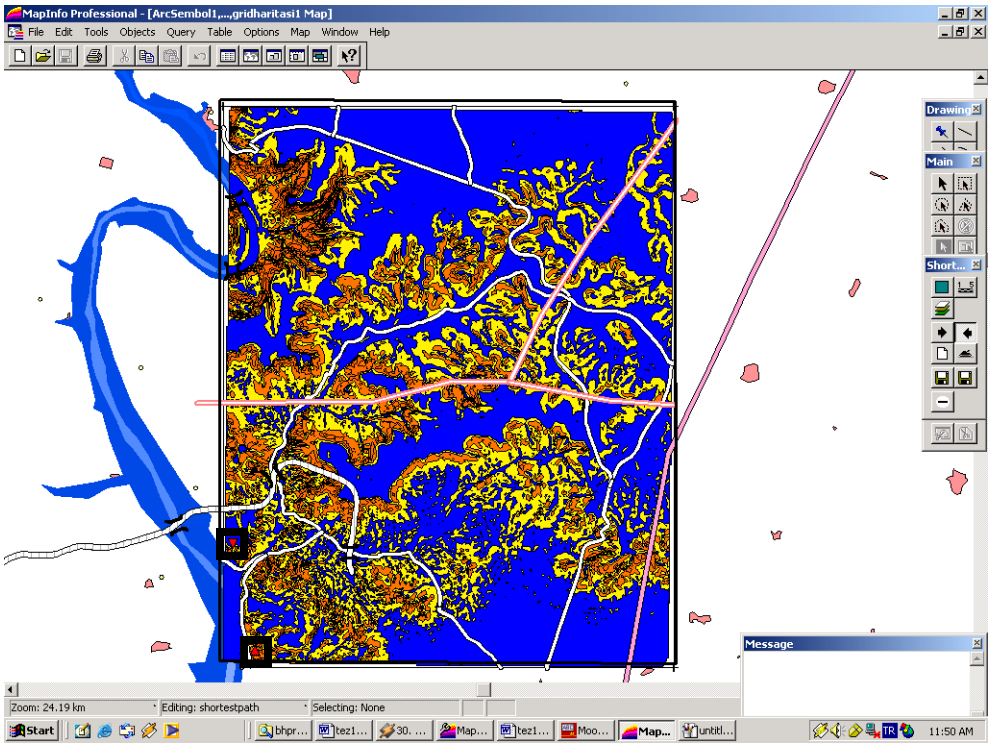


Figure 3.16 Starting and ending points

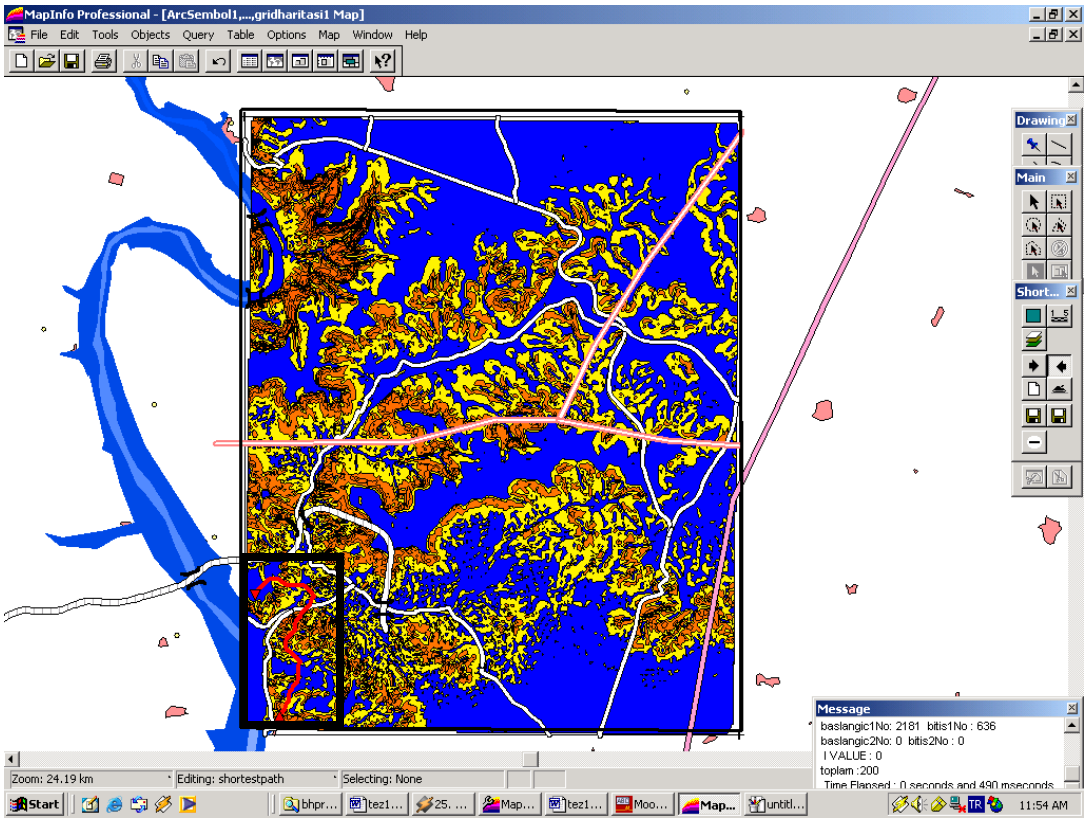


Figure 3.17 Shortest path on the left bottom side.

3.2 Scenarios For Implementation

In the previous study (Ünlü, 2002), the concept was to manage the movements of the military troops from the rally point to the assembly area. However, the shortest path algorithms can be implemented in other application areas. This flexibility is provided by this application in order to use the shortest path algorithms for other purposes. For this reason, the “change cost” window is used to assign cost values to the grid points. So, user can enter his/her criteria. In this study, tracking activity is applied to the application. But, some other scenarios can also be applied. For example, setting up the route for electric lines or pipelines. The cost values should be assigned to the layers needed in particular applications by the experts of the related fields.

A group of people that are joined to tracking activity is considered. The criteria are assigned according to these requirements. Figure 3.18 shows the cost assigned to each layer.

- Since people cannot use roads, railroads and motorways they are set to "9". Dam is also assigned to "9", because man cannot walk over dam.
- The most eligible place is village for human. "1" is assigned to them
- For river layer, people can walk through the river if the depth of the river is less than 1m. So; cost value of "3" is assigned for river regions that have depth between 0m and 0.5m, cost value of "4" is assigned for river regions that have depth between 0.5m and 1m, cost value of "5" is assigned for river regions that have depth values more than 1m.
- For slope layer, people can climb in all slope values, including 90^o by using several equipments (rope, hook etc.). So; cost value of "2" is assigned for regions that have slope values between 0^o and 10^o, cost value of "3" is assigned for regions that have slope values between 10^o and 30^o, cost value of "4" is assigned for regions that have slope values between 30^o and 60^o, cost value of "5" is assigned for regions that have slope values more than 60^o.
- People can use bridges, but it slows down the movement speed. Because, if a vehicle is going over, people must wait until it passes. So, "4" is assigned for this layer indicating that it is not a good area as a passageway.
- Tumulus region is forbidden to pass through so "9" is assigned.
- High voltage areas are dangerous places but can be passed carefully. So, "5" is assigned as a cost value.

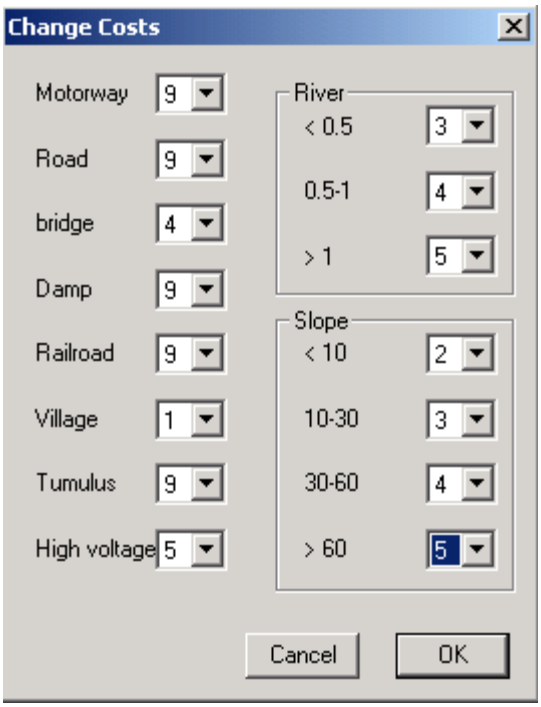


Figure 3.18 Cost values for tracking

After the cost values are determined, the overlay analysis is performed according to the criteria determined for tracking. A few scenarios are prepared to find the shortest path.

i. Case 1

In the first case, shortest path is found between two points that are very close to motorway. Since people cannot use the motorways and roads, the program cannot find the shortest path. Hence, it reports this by a warning message "Route cannot be drawn. There are obstacles". However, in the previous scenario that is prepared for military troops, this path can be found. Figure 3.19 shows the warning message, starting and ending points of the path.

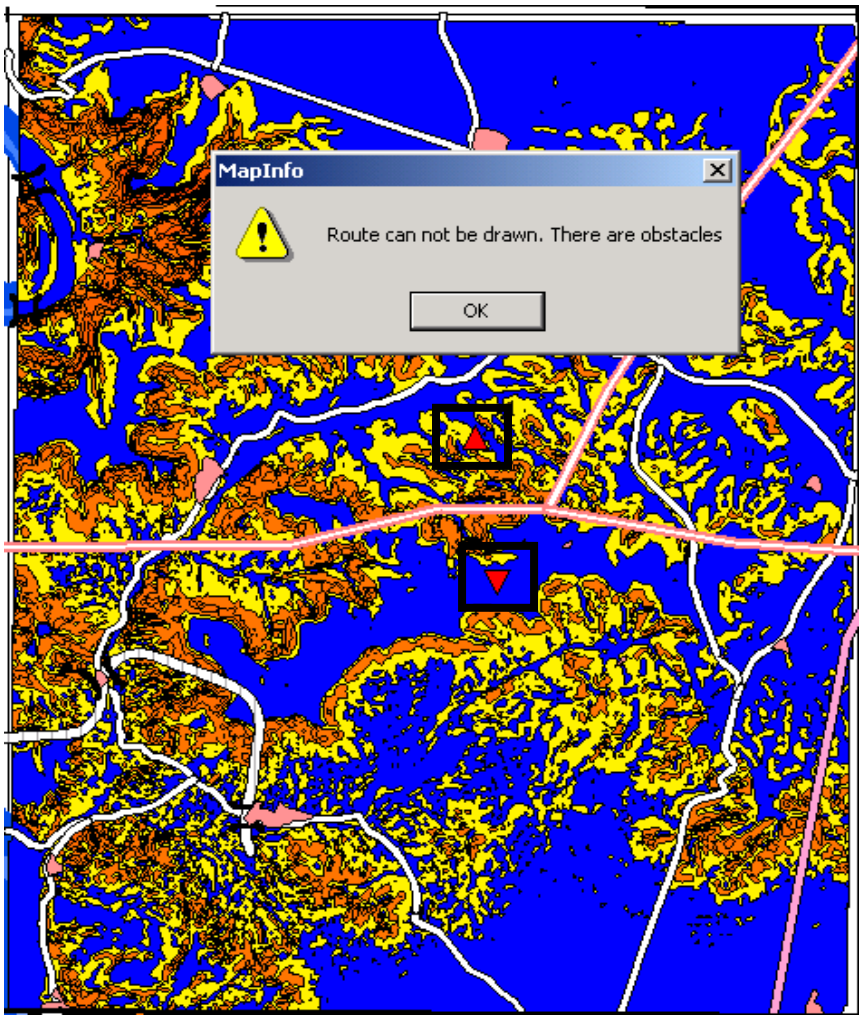
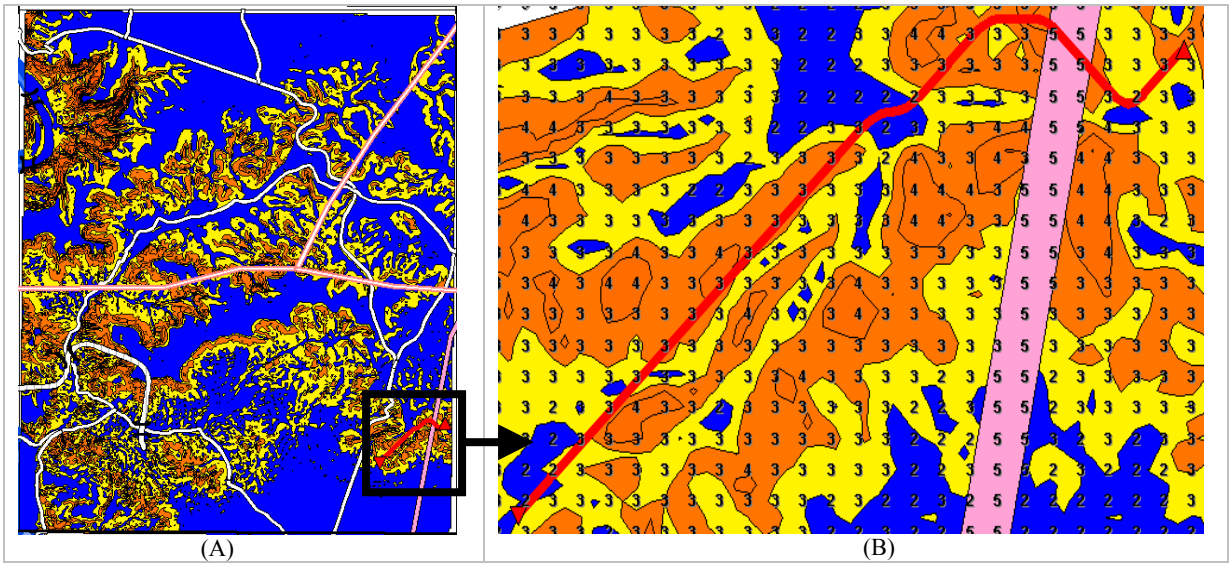


Figure 3.19 Shortest path cannot be found from one side of the motorway to the other

ii. Case 2

In this case, a path is found from one side of the high voltage area to the other side. In the default scenario, the high voltage area was a forbidden region. In this case, shortest path passes through the high voltage area as seen in Figure 3.20.



(A) (B)

Figure 3.20 Shortest path from one side of the tumulus area to the other side.

iii. Case 3

The shortest path is found from one side of a road to the other side as seen in Figure 3.21. Because, the roads are considered as obstacle, the shortest path does not go across the road. It finds the bridges and passes to the other side of the road by using the bridges.

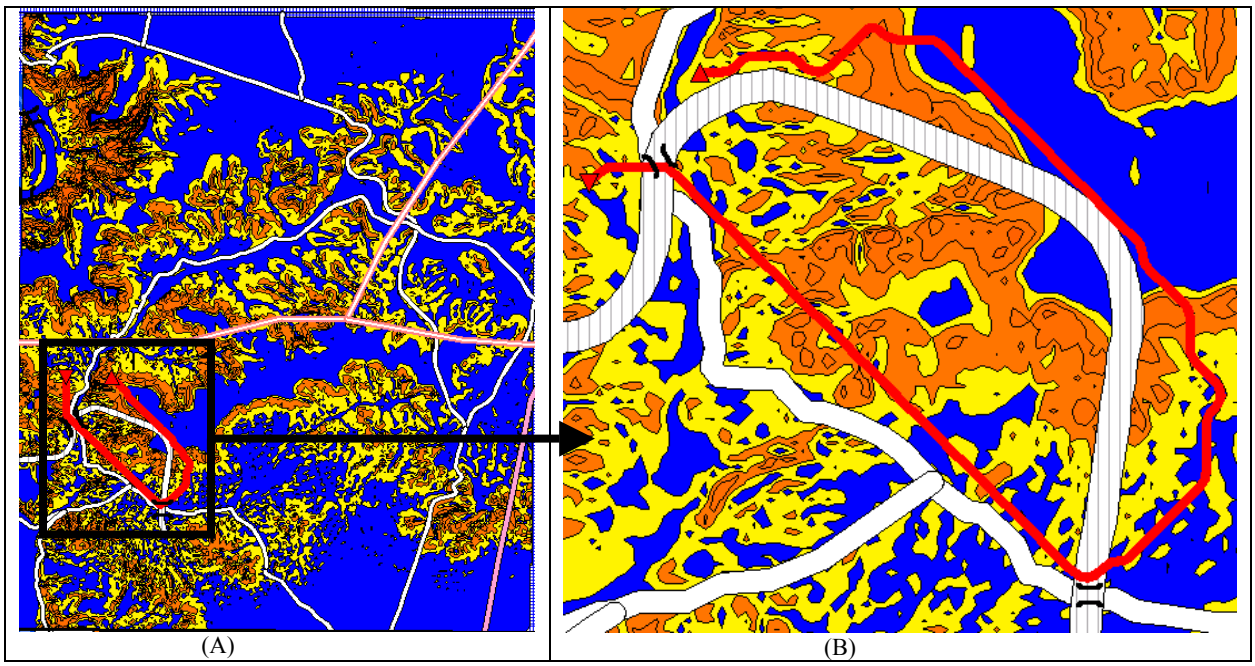


Figure 3.21 Shortest path that passes through bridges.

3.3 Discussion Of The Results

The shortest path application programs[3] are compared in means of performance and shortest path estimation.

3.3.1 Performance Comparison

The purpose of this study is to increase the performance of the shortest path analysis in a GIS application. In order to achieve this goal, the application that navigates the military troops is selected. The code segment of the "Prepare Grid Map", "Overlay" and "Draw Path" icons in "Shortest Path" toolbar is changed. "Save Path" and "Save Symbol" icons are added into the application. Two different shortest path programs are used behind this application. The first program has a shortest path algorithm of Dijsktra and the other program uses the Dijktra's algorithm with priority queue implementation. In the second one, the priority queue is implemented by using the binary heap algorithm as explained in Section 2.4.

[3] The application programs use dijsktra's algorithm, dijsktra's algorithm with priority queue implementation and Spatial Analyst Tool of ArcMap.

The grid map used in this application is translated to the ArcMap in shape data format. Then, it is converted to raster data. The starting point and ending points are constructed by creating two different layers. A program is coded in ArcMap by using Visual Basic (The code written for finding shortest path by using Spatial Analyst tool of ArcMap is given in Appendix H). This program gets the data layers that containing starting point, ending point and the raster data. It finds the cost direction and cost distance based on the raster data. The Spatial Analyst tool incorporates the costpush and costgrow methods in order to compute the least-cost path. The costpush method uses the pushbroom algorithm and costgrow method uses the growth algorithm, which are linear algorithms (Huber, 2000). The program[1] gets the time before starting to the shortest path operation and after the shortest path found. It writes the beginning and ending times to a file. This program is used in order to find the time performance of the Spatial Analyst tool of ArcMap over "gridharitasi1" layer.

In order to compare the performance of the programs, five different samples of "gridharitasi1" data layer are prepared in different number of points (vertices). The first sample has 5800 vertices, the second has 11600 vertices, the third one has 18000 vertices, the fourth one has 24600 vertices and the last one has 31600 vertices. Each of the data layers is converted to ArcMap in raster data format. These data layers (grid map and equivalent raster maps) are shown in Figure 3.22-Figure 3.26 respectively. In Table 3.4, the properties of each sample data layer are presented.

[1] The program that finds shortest path by using Spatial Analyst tool of ArcMap

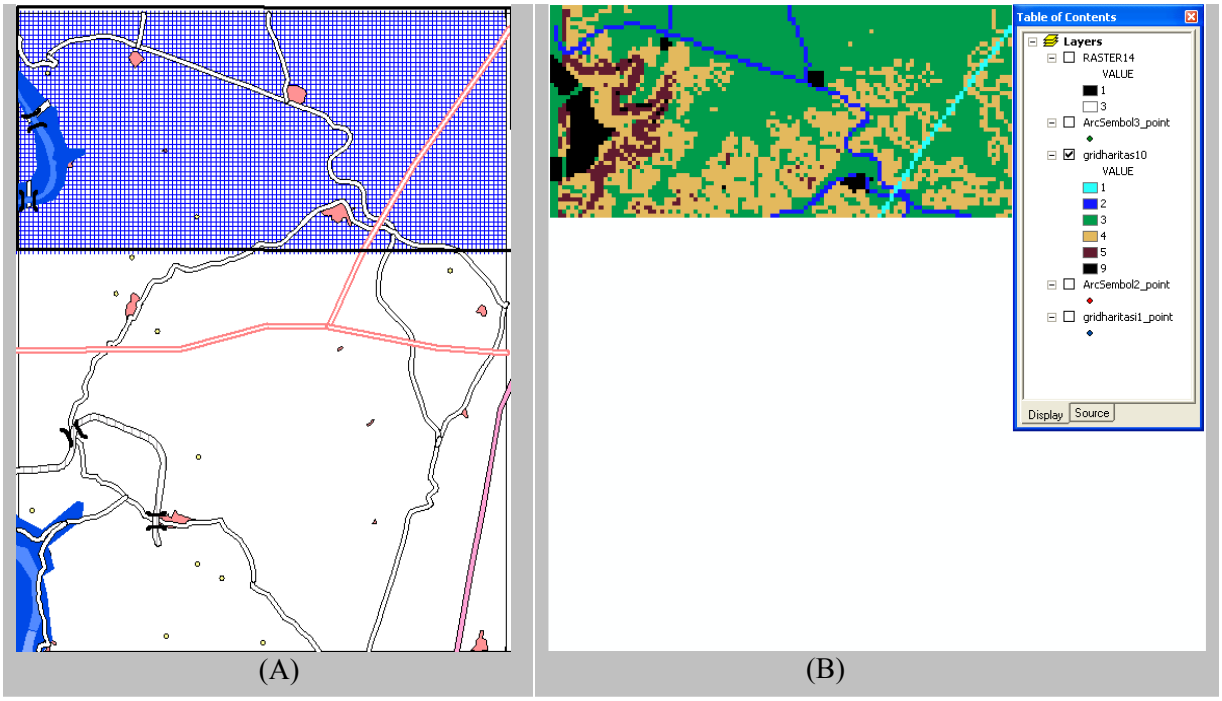


Figure 3.22 Grid (A) and raster (B) data layers containing 5800 vertices

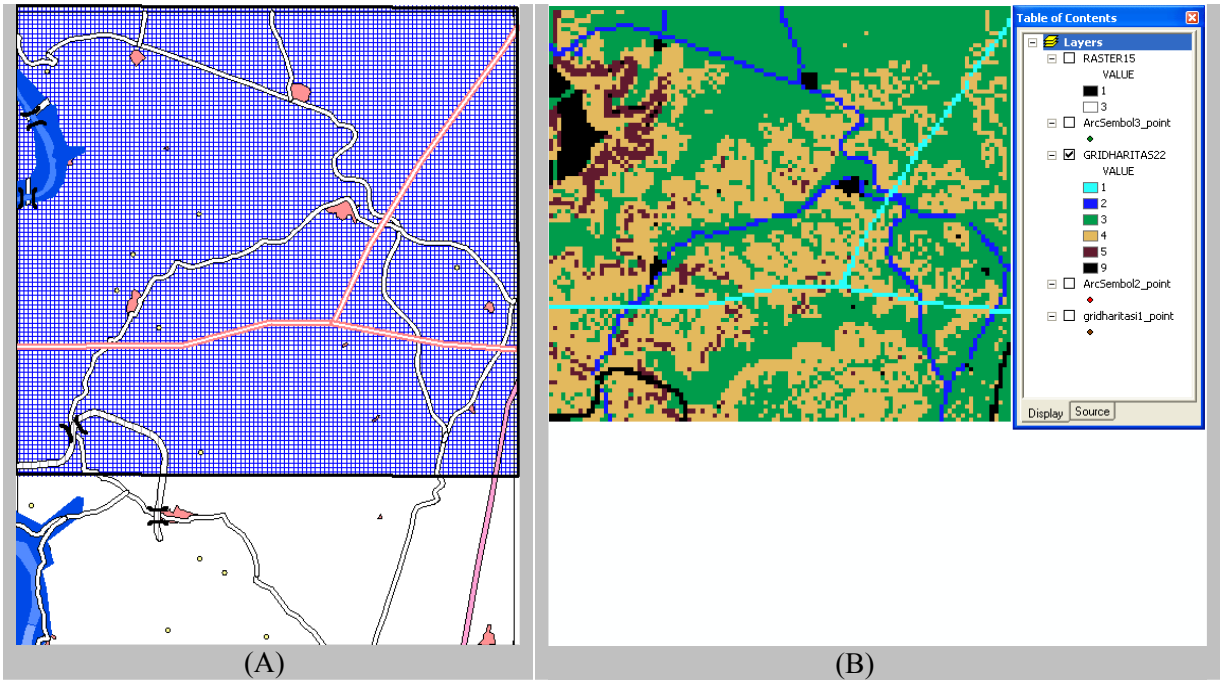
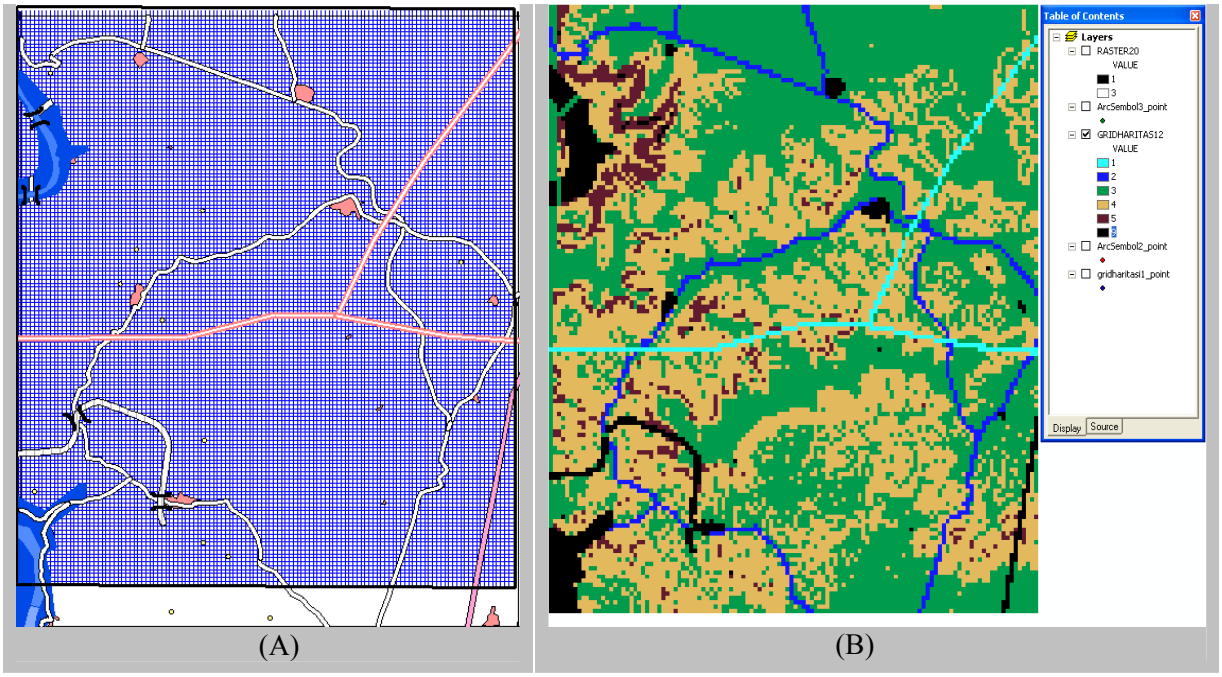
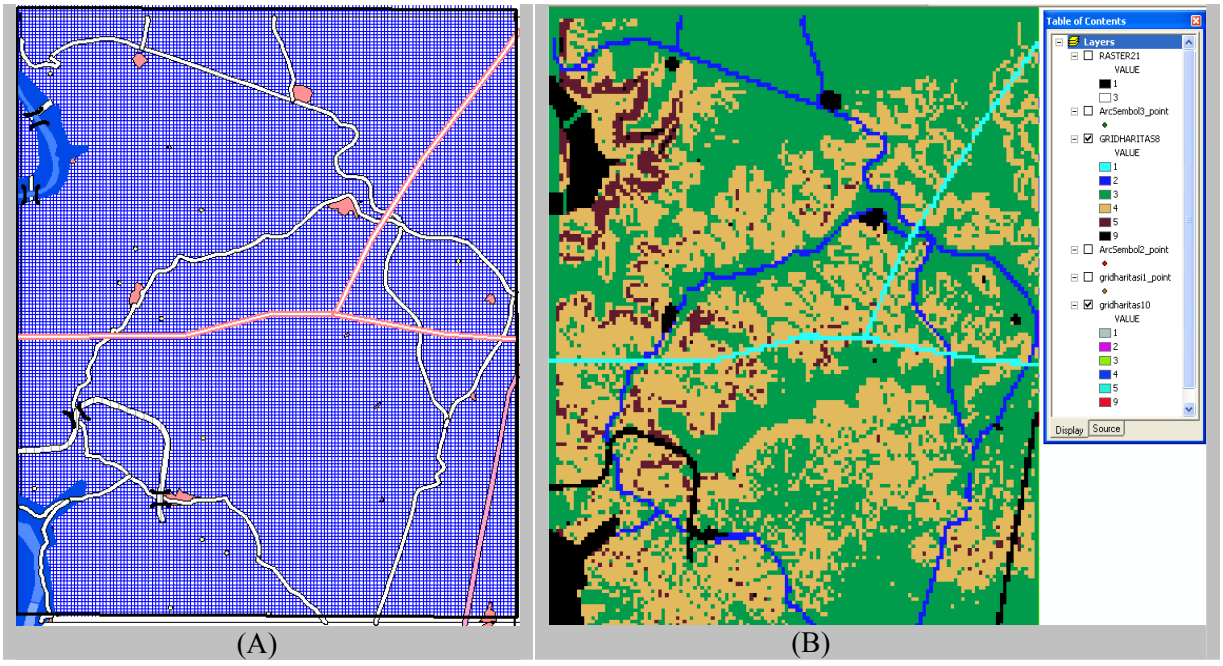


Figure 3.23 Grid (A) and raster (B) data layers containing 11600 vertices



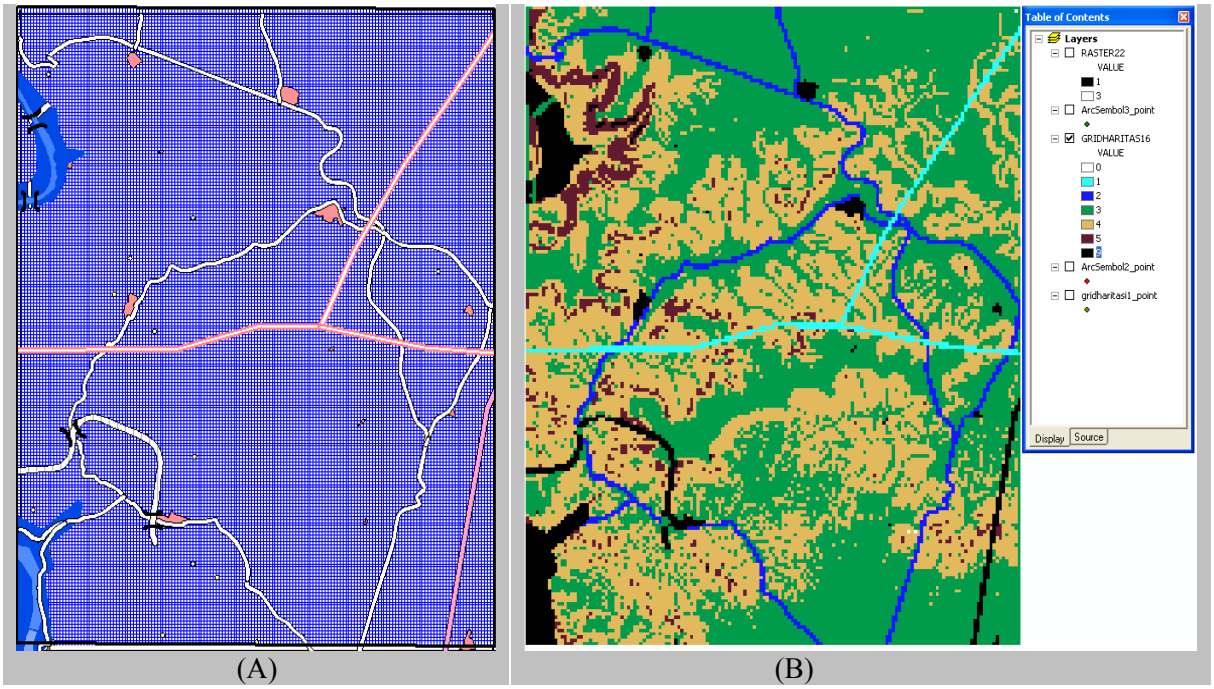
(A) (B)

Figure 3.24 Grid (A) and raster (B) data layers containing 18000 vertices



(A) (B)

Figure 3.25 Grid (A) and raster (B) data layers containing 24600 vertices



(A) (B)

Figure 3.26 Grid (A) and raster (B) data layers containing 31600 vertices

From the smallest sample to the largest one, the cell size of the grid map is decreased. Because the slope layer covers only 157km^2 region, the numbers of vertices in "gridharitasi1" layer are increased by decreasing the cell size of the grid map. Because of this reason, starting from "SampleGrid3" map (18000 vertices), the cell size is decreased in order to ensure the increase in number of points.

These five data layers are prepared in all of the applications (Dijsktra's algorithm, Dijsktra's algorithm with priority queue implementation, Spatial Analyst tool of ArcMap) identically. For the application, which uses Dijsktra's algorithm, grid data layer is created by using the "Prepare Grid Map" icon. Then, this newly created map is copied to the application that uses Dijsktra's algorithm with priority queue implementation. At this step, the cost values are computed in both applications. Because of this reason, for each sample grid layer, the position of the vertices and the cost of the vertices are completely the same in both applications (Dijsktra's algorithm and Dijsktra's algorithm with priority queue implementation). The raster map, which is used by Spatial Analyst tool of ArcMap, is found by translating the grid data to raster format in ArcMap with the same resolution. All of the algorithms are run several times for each sample grid layer. Finally,

the average of the run times found, are computed for each layer. The results are compared in Table 3.5. The run time results are presented graphically in the Figure 3.27. The "y" axis shows the time in milliseconds and the "x" axis shows the sample maps (the number of vertices). As seen from the figure, when the number of vertices increases, the increase in Dijsktra's algorithm with priority queue implementation and Spatial Analyst tool of ArcMap is linear. But, the increase in Dijsktra's algorithm is exponential.

Table 3.4 Properties of sample grid layers

Name Of The Grid map	Number Of Vertices	Extend (km^2)	Cell Size (m)
SampleGrid1	5800	58	100
SampleGrid2	11600	116	100
SampleGrid3	18000	145	90
SampleGrid4	24600	157	80
SampleGrid5	31600	155	70

Table 3.5 Performance comparison

Name Of The Grid map	C Program using Dijsktra's Algorithm	Spatial Analyst Tool Of ArcMap	C Program using Dijsktra's Algorithm With Priority Queue Implementation
SampleGrid1	3 seconds 750 milliseconds	7 seconds	32 milliseconds
SampleGrid2	15 seconds 235 milliseconds	7 seconds	62 milliseconds
SampleGrid3	36 seconds 359 milliseconds	7 seconds	93 milliseconds
SampleGrid4	68 seconds 703 milliseconds	8 seconds	125 milliseconds
SampleGrid5	114 seconds 578 milliseconds	8 seconds	164 milliseconds

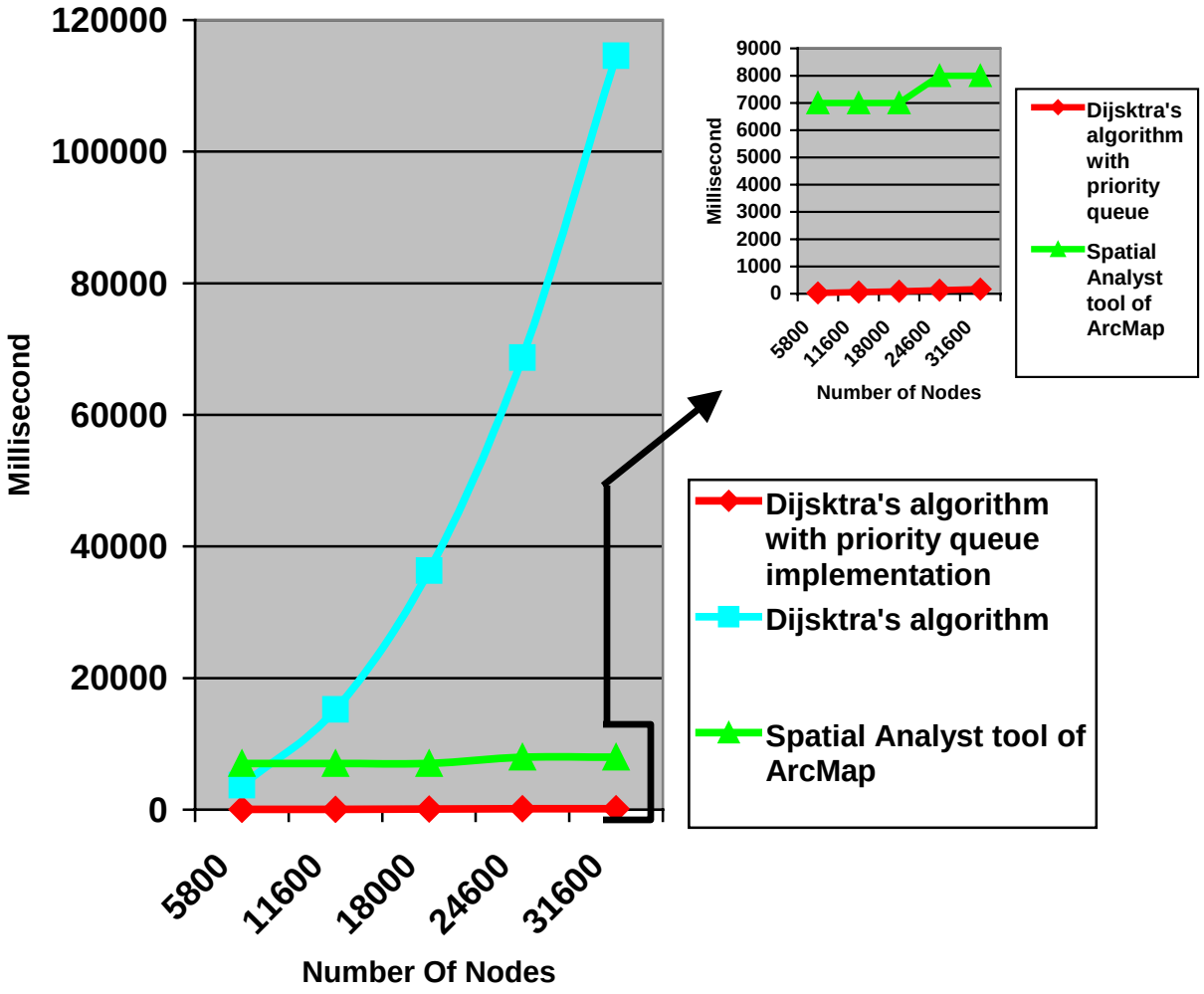


Figure 3.27 Performance comparison in terms of time.

3.3.2 **Shortest Path Estimation**

When computing the shortest path (least-cost path) following two rules must be satisfied.

- The cost of the path must be minimum
- If there is more than one path with minimum cost, then the path with shortest distance from starting vertex to the ending vertex should be selected.

In this perspective, at first, two shortest path algorithms (Dijsktra's algorithm and Dijsktra's algorithm with priority queue implementation) are compared in five scenarios (cases). Then all of the algorithms are compared in a single scenario.

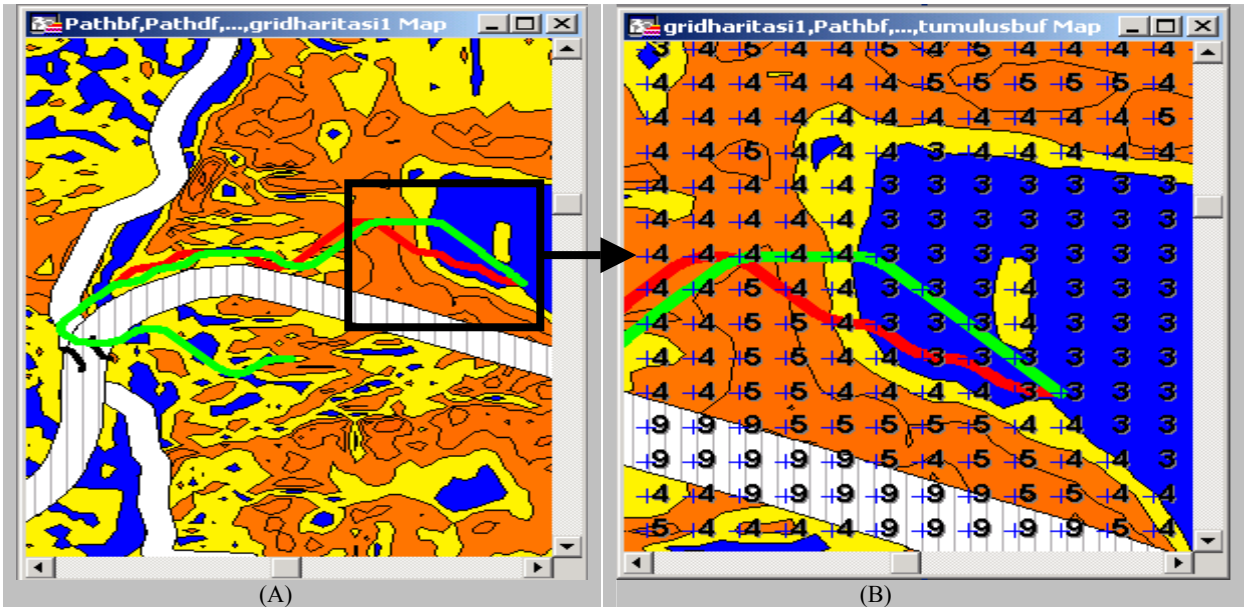
i. Comparison Of The Dijsktra's algorithm and Dijsktra's algorithm with priority queue implementation

All the scenarios are tested by "SampleGrid5" map which has 31600 vertices. In each case (scenario), shortest path is initially found by using the application that uses Dijsktra's algorithm. Then, by using "Save Path" icon in "Shortest Path" toolbar, the path found is stored to the current folder. This application is closed and the application that uses Dijsktra's algorithm with priority queue implementation is run. The path that previously saved is opened and the starting and ending vertices are found. The shortest path is similarly run by this application. Finally the path found is also stored to the current folder. This is repeated for each case. The path found by the application that uses Dijsktra's algorithm is colored as red and the path created by the application that uses Dijsktra's algorithm with priority queue implementation is colored by green. Table 3.6 summaries the results found in all cases. It can be said that the application that runs Dijsktra's algorithm with priority queue implementation finds better paths. However, it should be reported that there is not any difference between Dijsktra's algorithm and Dijsktra's algorithm with priority queue implementation. The difference found here is due to implementations for both of the algorithms that are prepared for this study and the previous study (Ünlü, 2002).

Table 3.6 Path quality comparison

Scenario	Cost Of The Application using Dijsktra's Algorithm (Unit)	Cost Of The Application using Dijsktra's Algorithm with Priority Queue implementation (Unit)	Result
Case 1	118	118	Path lengths are the same
Case 2	188	188	Path length is shorter by DAPQ program
Case 3	114	114	Path length is shorter by DAPQ program
Case 4	463	463	Path length is shorter by DAPQ program
Case 5	533	533	Path length is shorter by DAPQ program

Case 1: In this case, a path is searched from one side of the railroad to the other side. Since railroad is a region that cannot be passed through, both applications find a path passing through the bridge. The cost is 118 units for both of the applications. So according to the rules specified above, the length will be compared. As it is seen in the Figure 3.28-A, two different paths are found in the squared region. This region is shown in detail in Figure 3.28-B. When we compare the lengths of the paths for this region, it is seen that two paths have the same length. Because, there are 3 straight movement and 4 diagonal movement for both paths.



(A) (B)

Figure 3.28 Path quality comparison for case 1

Case 2: In this case, a path is searched beginning from a point near to the bridge at the bottom to another point that is in the upper side of the other bridge. For both applications, the path goes through the bridges and finds a path, which has a cost of 188 units. Figure 3.29 shows the path in general and detailed formats. As it is seen from Figure 3.29, paths are separated at the region that is close to the upper point. Similar to the previous case, the length of the paths is searched. As it is seen clearly in Figure 3.29-B, green path is shorter than red path. So, the application that runs Dijsktra's algorithm with priority queue implementation finds a better path.

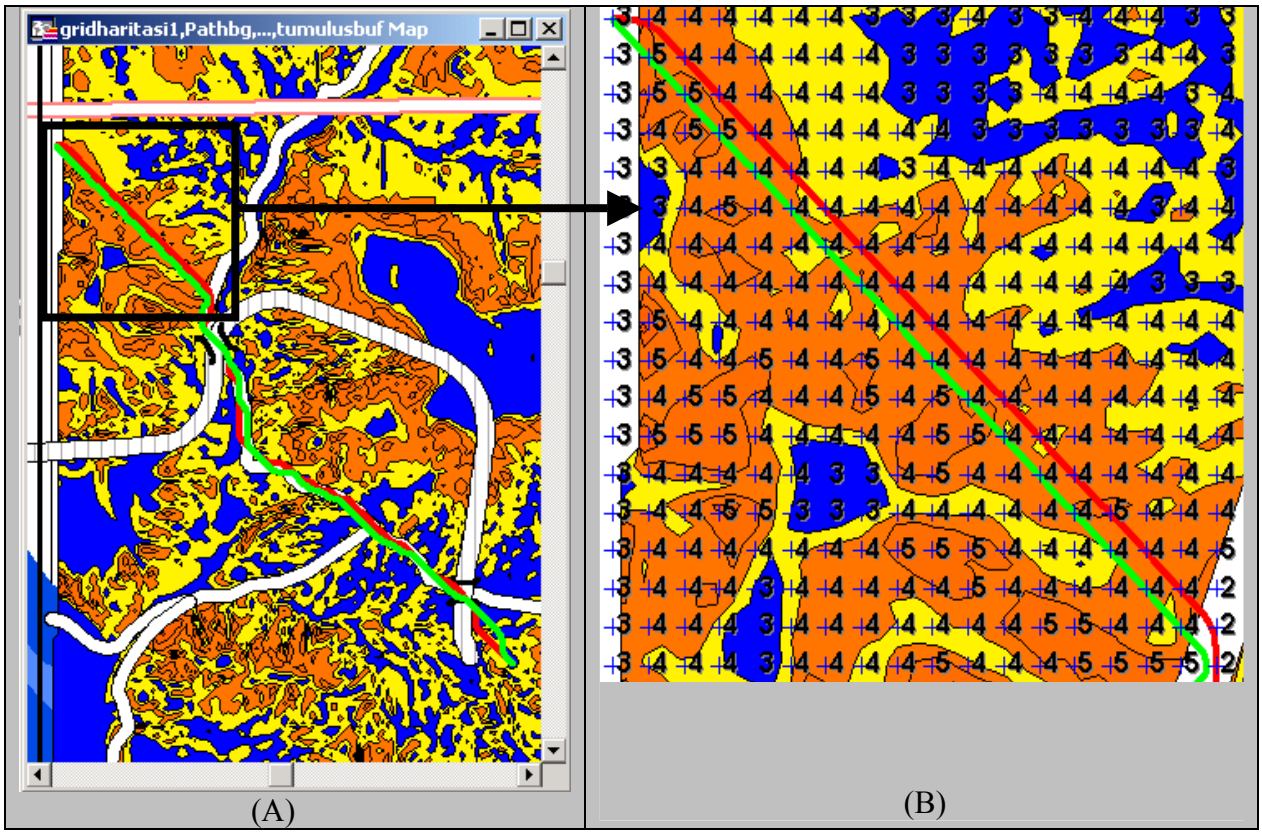


Figure 3.29 Path quality comparison for case 2

Case 3: In this case, a path is searched from road to road in terrain data. The cost of the path is 204 units. So according to the rules specified above, the length will be compared. The paths differ in two regions, which is shown in Figure 3.30-A. The two figures in Figure 3.30-B shows the difference in paths in more detail. Green path is shorter than red path. So, the application that runs Dijsktra's algorithm with priority queue implementation finds a better path.

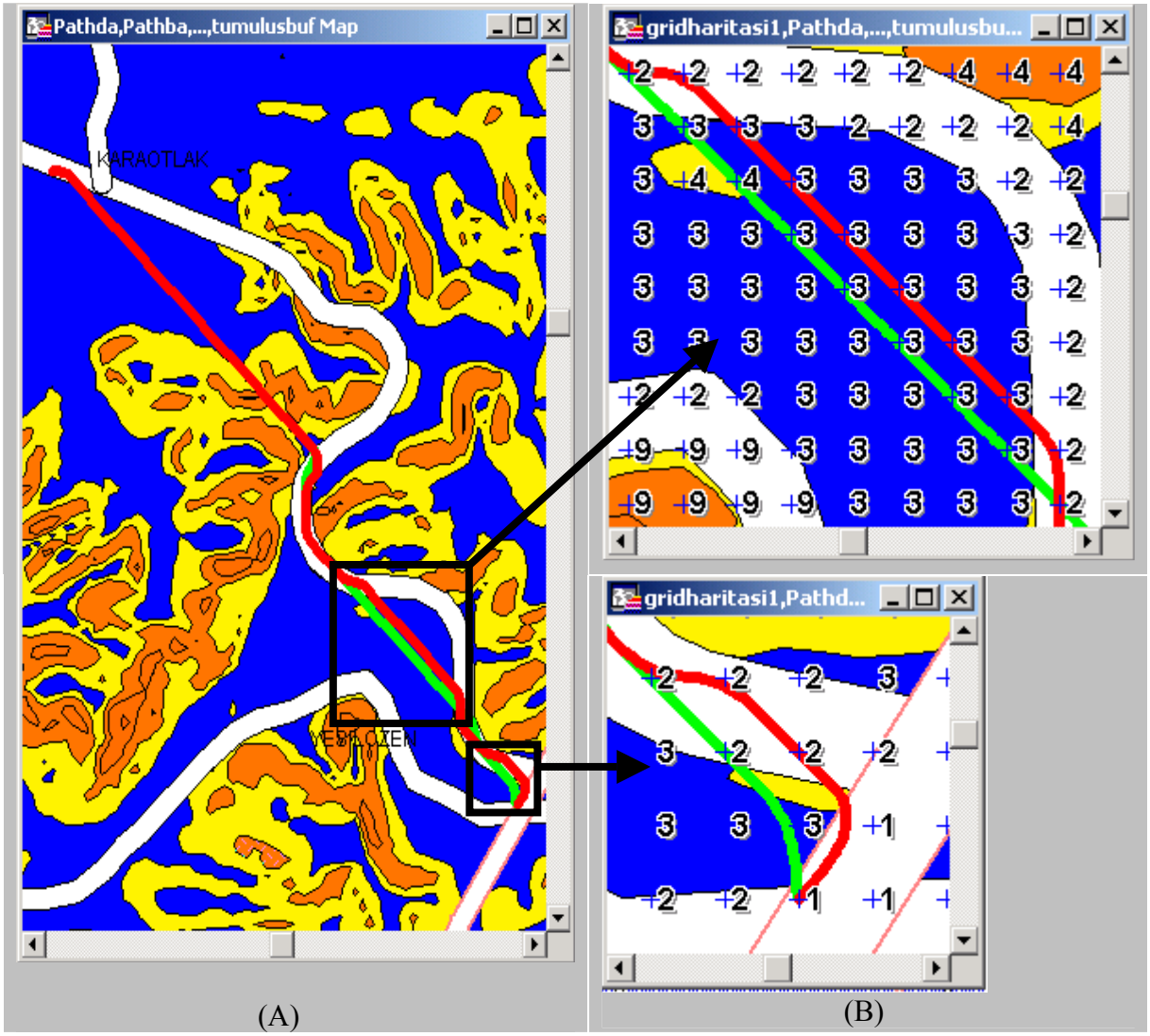


Figure 3.30 Path quality comparison for case 3

Case 4: In this case, a path is searched in long distance in terrain data. The cost of the path is found as 463 in both applications. Then, the lengths are compared. As, it is seen in Figure 3.31, green path is shorter than the red path. So, the application that runs Dijsktra's algorithm with priority queue implementation finds a better path.

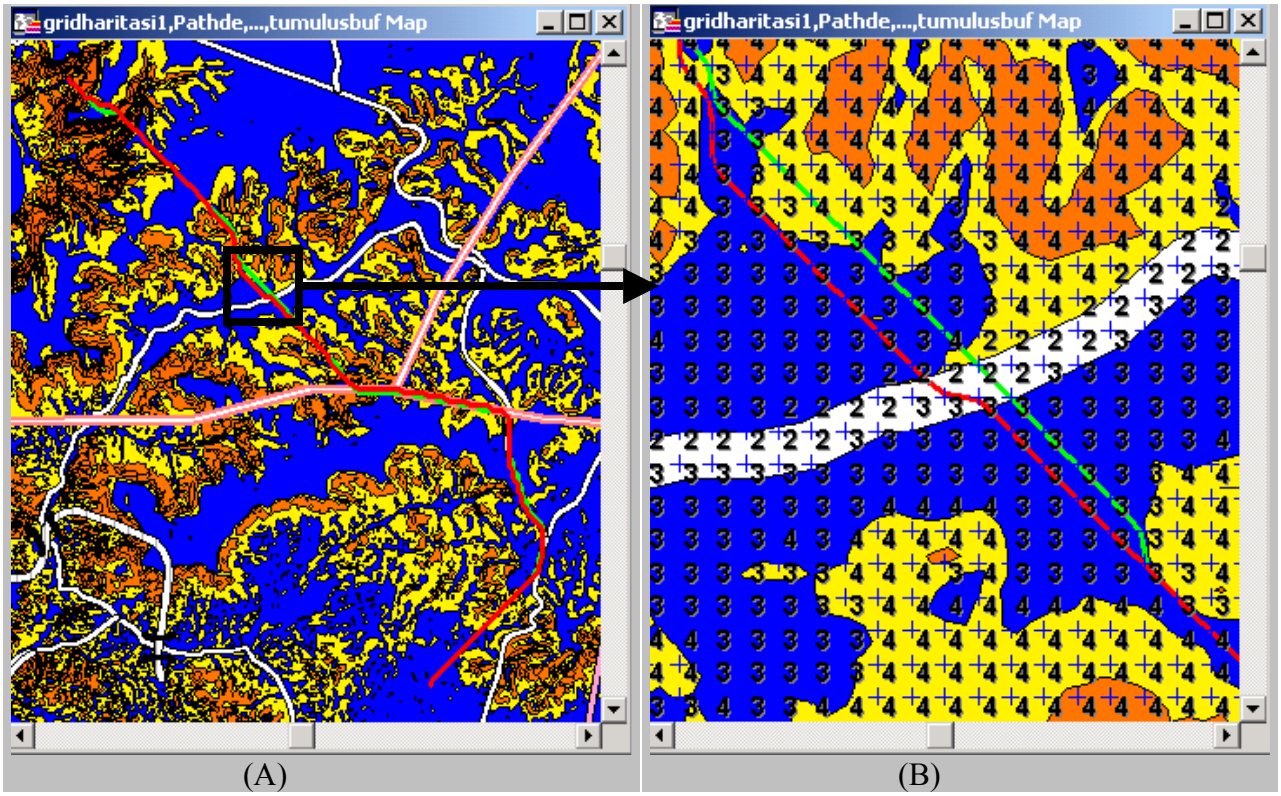


Figure 3.31 Path quality comparison for case 4

Case 5: In this case, a path is searched in long distance in terrain data. The cost of the path is found as 533 units in both applications. Then, the lengths are compared. As, It is seen in Figure 3.32, green path is shorter than red path. In addition to that, the difference in length is bigger in this case compared to the above cases. Again, in this case, the application that runs Dijsktra's algorithm with priority queue implementation finds a better path.

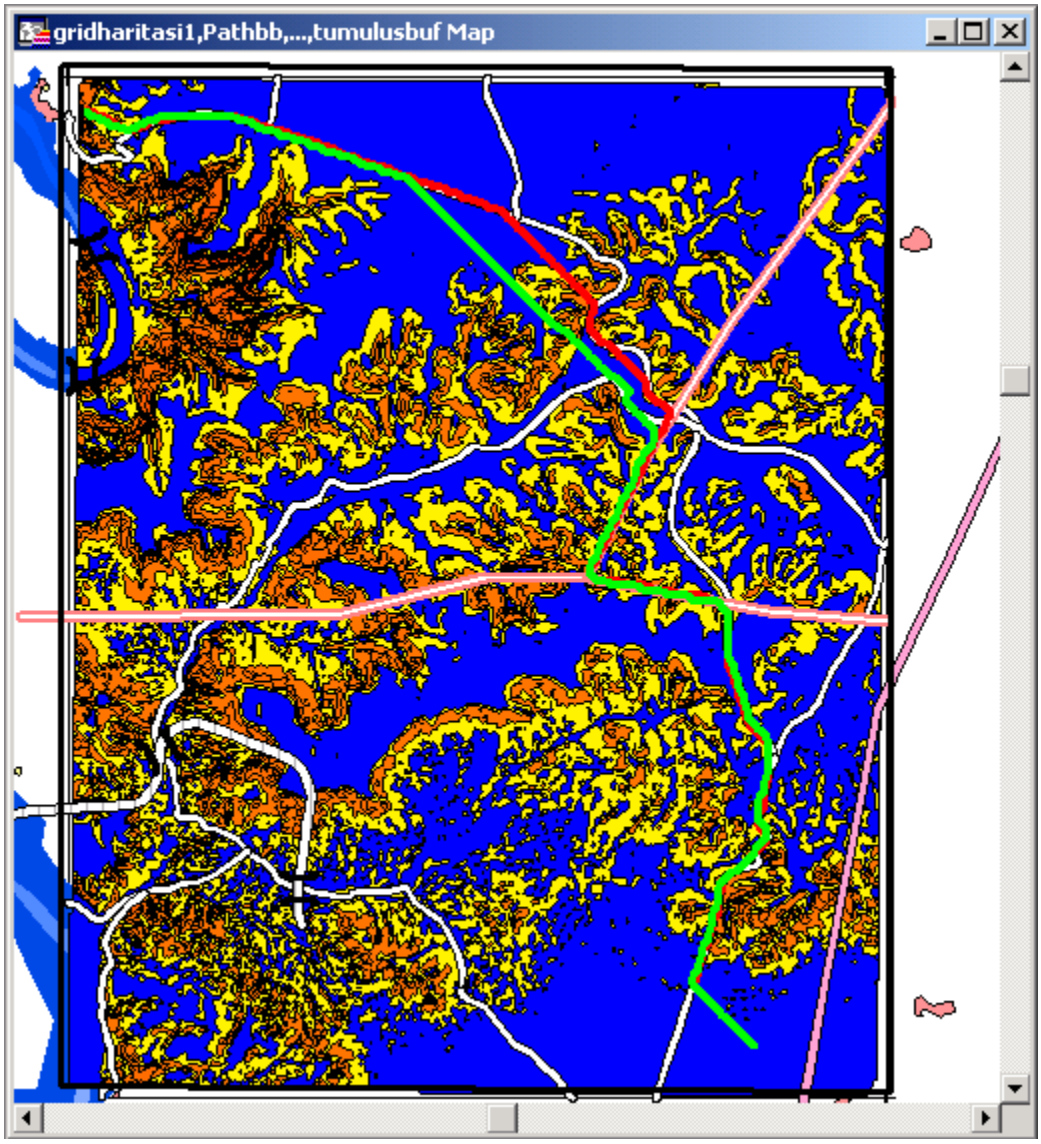


Figure 3.32 Path quality comparison for case 5

ii. Comparison Of All Of The Algorithms

Since; the cost value "9" is treated as an obstacle in the applications that uses Dijsktra's algorithm and Dijsktra's algorithm with priority queue implementation, the shortest path (least-cost paths) found for them and Spatial Analyst tool of ArcMap are different. In order to compare the shortest path found by Spatial Analyst tool of ArcMap to the other applications, the bottom side of the SampleGrid5 map is selected. Because, there are not any obstacles in that region which have a cost value of "9". It is wanted to show that all

of the algorithms find the same path if the cost values are completely the same in that region. The path found by Spatial Analyst tool of ArcMap is shown in Figure 3.33 and the paths found for each application are shown in Figure 3.34 in different colors. Yellow line shows the path found by Spatial Analyst tool of ArcMap, blue and green lines show the path found by the applications that uses the Dijsktra's algorithm and Dijsktra's algorithm with priority queue implementation, respectively.

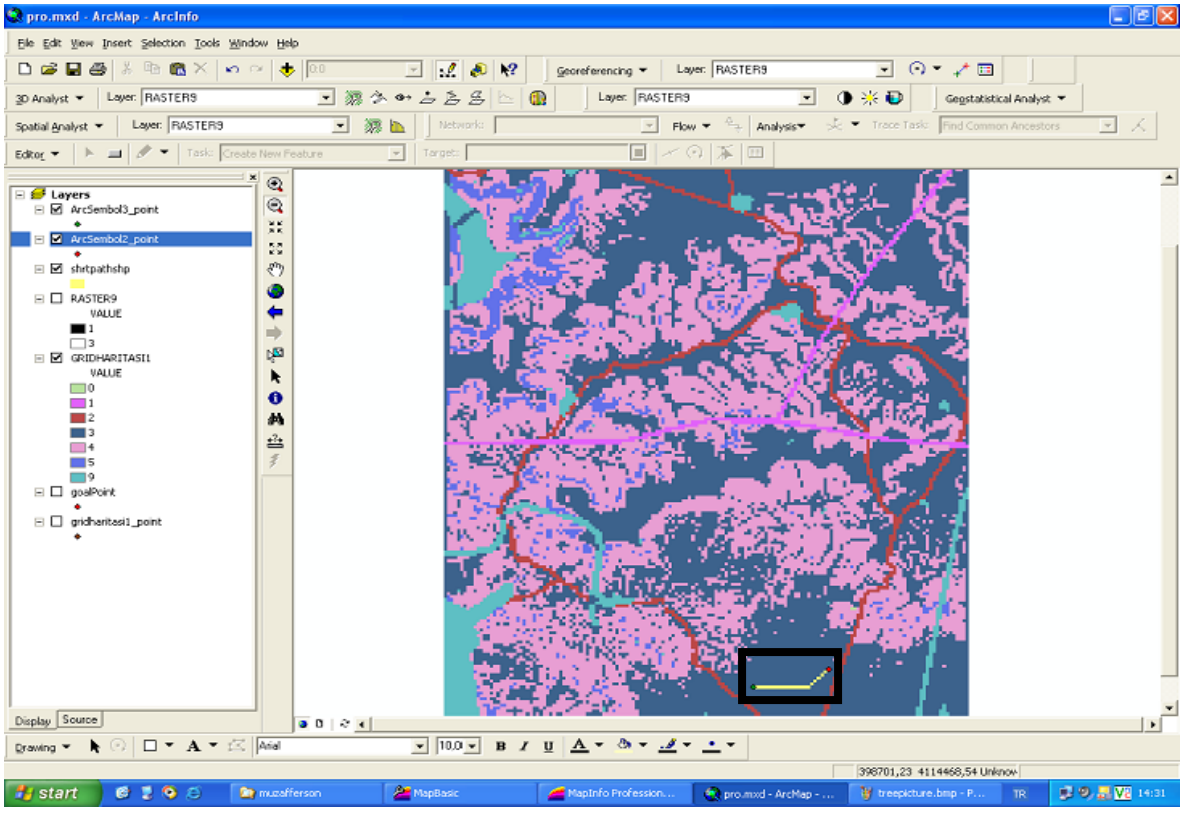


Figure 3.33 The path found by Spatial Analyst tool of ArcMap

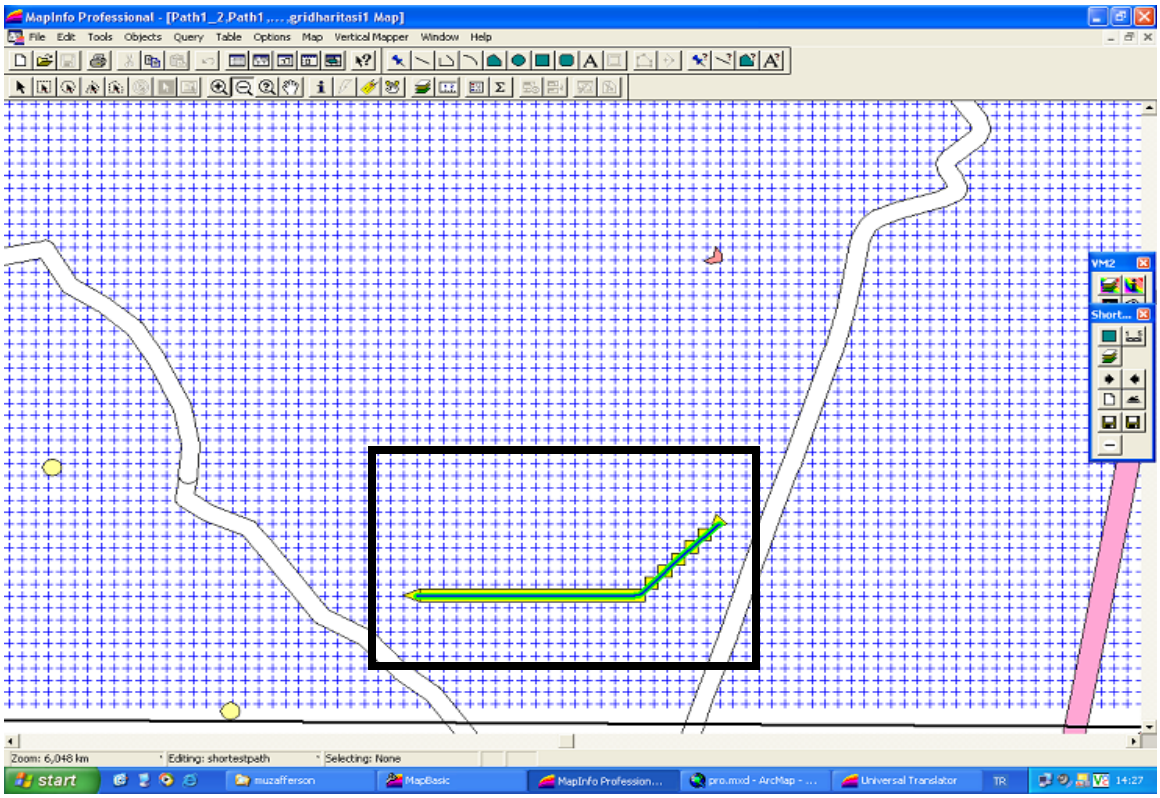


Figure 3.34 All of the paths are overlapped on top of each other.

CHAPTER 4

CONCLUSION AND RECOMMENDATIONS

4.1 Conclusion

The network analysis have many application areas such as path finding, route planning, etc. Raster-based GIS is not commonly known for network analysis applications. Raster-based GIS is location oriented, where each cell is part of a tessellated continuous surface that describes a given attribute. A raster-based GIS fits to the path finding problem across terrain data. In this study, raster-based GIS layer has appropriately mirrored the terrain properties. As the cell size of the raster data layer is conveniently selected, all of the attribute layers have contributed to the cost values of the raster data layer.

There are a number of algorithms that solves the shortest path problem in raster-based GIS. Some of the algorithms improve the time performance while descreasing the quality of the path significantly. In this study, the Dijsktra's algorithm with priority queue implementation is prefered. Because, the priority queue implementation reduces the number of iterations of Dijsktra's algorithm from $O(V^2 \log V)$ to $O(E \log V)$. The time complexities are proven empirically for each application by using raster-based GIS layers with the same number of vertices. The Spatial Analyst tool of ArcMap is also compared with Dijsktra's algorithm with priority queue implementation. It is seen that this algorithm improves the time performance of the shortest path analysis compared to Spatial Analyst tool of ArcMap.

The raster-based GIS layers are constructed for both of the algorithms and Spatial Analyst tool of ArcMap with different number of vertices for each case. The time performance curves can be empirically derived by using the results of each case. As a

result, the Dijsktra's algorithm with priority queue implementation and Spatial Analyst tool of ArcMap have graphical representations of linear lines. For Dijsktra's algorithm, the graphical representation forms a quadratic line. This result also proves the time complexities of the algorithms.

The least-cost paths that found by the algorithms are compared in means of path quality. A path is considered as a least-cost path if the total cost is minimum and the total distance from starting vertex to the ending vertex is minimum. Based on this criterion, the paths are compared for both algorithms. It is seen that this application finds better paths compared to the previous application (Ünlü, 2002) that uses Dijsktra's algorithm.

This raster-based GIS application was initially prepared for use of navigating military troops on terrain. But, it is adapted to other scenarios, by changing the costs of the raster data layer. As an experiment, the cost values assigned to the raster data layer is changed according to the needs of tracking activity. It is seen that the application can dynamically change the cost values and find paths based on the new criteria.

4.2 Recommendations

In this study, the Dijsktra's algorithm with priority queue implementation is experienced over raster-based network model. It is seen that the algorithm work quite fine with this data model. As raster-based network model, vector-based network model can be used in many types of applications. It can be experienced to integrate Dijsktra's algorithm with priority queue implementation to an application that uses vector-based network model.

The priority queue of the Dijsktra's algorithm is based on binary heap. In literature, there are some other algorithms that can be used instead of binary heap algorithm. Some of them are binomial heap, fibonacci heap and relaxed heap algorithms. In order to compare the time performance of the binary heap algorithm to other algorithms (binomial heap, fibonacci heap or relaxed heap), it can be replaced with each of them. The experimental results for each of the priority queue implementation can be found.

The system can be extended to enable extra capabilities by the use of several hardware devices in real-time. It could based on the integration of satellite derived image, geographic information system (GIS), global positioning system (GPS) and global system for mobile communication (GSM) and weather conditioning sensor technologies. The real-time update of the satellite image enables the program to detect the changes in terrain. In addition, the weather conditioning sensors send real-time data indicating if it is raining, snowing or etc. The images taken from satellite and the information coming from weather conditioning sensors enables a better estimation of the real-world environment in real-time. The GPS and GSM technologies will be used to transmit the exact positions of travelers to the GIS application. Each traveler will be equipped with a GPS receiver to determine its exact position based on the signal transmitted by satellites. Also, each traveler will have a GSM modem in order to transmit its position to the GIS application. As the path finding algorithm finds the shortest path in a very short time, the system can be developed in real-time. By using these technologies, the program on the main command center can control the travelers according to the changing conditions in the real world.

The overlaying program section used in this study finds the overlay of the data layers in real-time. This program is based on a constant topography and true slope. For this reason, it uses the path finding approach, which simply finds the best route on the surface of terrain without any cut and fill. In real world, engineering cut and fill operations are necessary. Roads, bridges, settlement places are constructed by using the cut and fill operations. Finding the best path over terrain, when cut and fill operations could be performed, is the concept of route finding. By improving the shortest path algorithm used in this study, the best route can be found while regarding the cut and fill operations and the changes in the topography considerering.

REFERENCES

1. Ahuja, R.K., Mehlhorn, K., Orlin, J., Tarjan, R.E. (1990). Faster algorithms for the shortest path problem, *Journal of the ACM (JACM), Volume 37, Issue 2, pp. 213 - 223.*

2. Breunig, M. and Baer, W. (2004). Database support for mobile route planning systems, *Computers, Environment and Urban Systems. Under Press*

3. Breymann, U. (2002). Designing Components with the C++ STL, *Pearson Education Limited (2002) pp. 141-150*

4. Car, A., Taylor, G., Brunsdon, C. (2000). An analysis of the performance of a hierarchical wayfinding computational model using synthetic graphs, *Computers, Environment and Urban Systems 25 pp. 69-88*

5. Clarke, C. (1999). Getting Started with Geographic Information Systems, *2nd ed. G70.212 .C57 pp 146-151*

6. Deitel, H.M. (1992). C How To Program, *Prentice-Hall, Inc. pp. 559-683.*

7. Dijkstra, E. W. (1959) A Note on Two Problems in Connection with Graphs. *Numeriche Mathematik, pp. 269-271*.

8. Fredman, M.L. and Tarjan, R.E., (1987). Fibonacci heaps and their uses in improved network optimization algorithms, *Journal of the ACM (JACM), v.34 n.3, pp.596-615.*

9. Golledge, G., Klatzky, L., Loomis, M., Speigle, J., Tietz, J. (1998). A geographical information system for a GPS based personal guidance system, *Int. J. Geographical Information Science, vol. 12, no. 7, pp. 727- 749.*

10. Heward, I., Cornelius, S., Carver, S. (1998). An Introduction to Geographical Information Systems. *Longman, pp.13-56.*

11. Heyes, J. and Jones, J. (2001). A* algorithm tutorial, *visited on December 2003* *http://www.geocities.com/jheyesjones/astar.html*

12. Heywood, I., Cornelius, S., Carver, S. (1998). An Introduction to Geographical Information Systems, *G70.212. H49x 1998 pp. 110-117.*

13. Huber, B. (2000). A Review Of IDRISI, *visited on April 2004* http://www.directionsmag.com/features.php?feature_id=40

14. Husdal, J. (2000). Network analysis - network versus vector A comparison study, *http://www.husdal.com/mscgis/network.htm#network*

15. Johnson, D.B. (1977). Efficient Algorithms for Shortest Paths in Sparse Networks, *Journal of the ACM (JACM), v.24 n.1, pp. 1-13.*

16. Kara, B.Y., Erkut, E., Verter, V. (2003). Accurate calculation of hazardous materials transport risks, *Operations Research Letters, pp. 285-292.*

17. Kwan, M.P. and Lee, J. (2003). Emergency response after 9/11: the potential of real-time 3D GIS for quick emergency response in micro-spatial environments, *Computers, Environment and Urban Systems, pp. 1-15.*

18. Lanthier, M., Nussbam, D., Sack, J.R. (2003). Parallel implementation of geometric shortest path algorithms, *Parallel Computing Volume 29, Issue 10 , pp. 1445-1479.*

19. Lee, J. and Stucky, D. (1998). On applying viewshed analysis for determining least-cost paths on Digital Elevation Models , *Int. J. Geographical Information Science, vol. 12, no. 8, pp. 891-905.*

20. Pai, K. and Reissell, L.M. (1998). Multiresolution Rough Terrain Motion Planning, *IEEE Transactions On Robotics and Automation, Vol 14, pp. 19-33*

21. Rodrigue, J.P. (2003). Graph Theory: Definition and Properties, *visited on April 2003, http://people.hofstra.edu/geotrans/eng/ch2en/meth2en/ch2m1en.html*

22. Saunders, S. and Takaoka, T. (2001). Improved Shortest Path Algorithms for Nearly Acyclic Graphs, http://www.elsevier.nl/locate/entcs/volume42.html, *visited on October 2003.*

23. Schmuller, J. (1999). UML in 24 Hours, *A Division of Macmillan Computer Publishing, pp. 103-117.*

24. Tarantilis, C.D. and Kiranoudis, C.T. (2002). Combination of geographical information system and efficient routing algorithms for real life distribution operations, *European Journal of Operational Research 152 (2004) pp. 437–453.*

25. Ünlü, M. (2002). Planning The Tactical And Administrative Movements By Using GIS, *Middle East Technical University Library, pp. 21-54.*

26. Yu, C., Lee, J., Munro-Stasiuk, J. (2003). Extensions to least-cost path algorithms for roadway planning, *Int. J. Geographical Information Science, vol . 17, no. 4, pp. 361–376.*

27. Wayne, K. (2002). Binary and Binomial Heaps, *visited on March 2004* *www.cs.princeton.edu/~wayne/cs423/ lectures/heaps-4up.pdf*

Appendix A: User Manual Of The Application

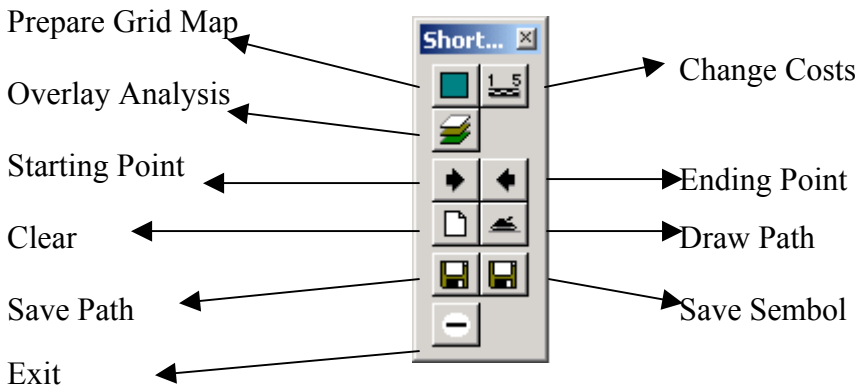


Figure A - 1 Shortest Path Toolbar

Prepare Grid Map: By using this icon, user can draw a rectangle. This rectangle specifies the boundaries of the grid map that will be constructed. When the ending point of the rectangle is selected, a dialog box appears as seen in Figure A-2. By this dialog box, user can enter the "grid interval". Grid interval is the size of a cell in grid map. Finally a grid map is constructed.

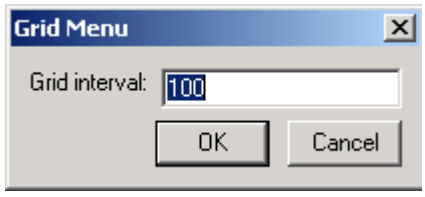


Figure A - 2 Grid Menu Window

Change Costs: By using this icon user can change the costs. The cost is updated by using the window, which is shown in Figure A-3.

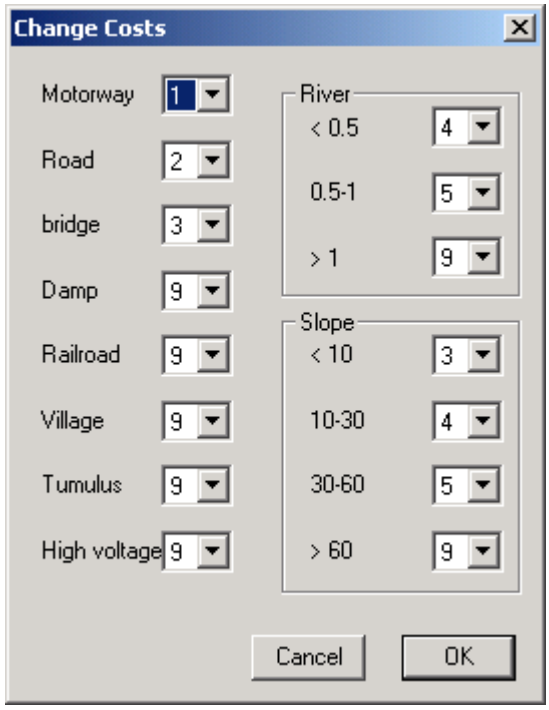


Figure A - 3 Change Cost Window

For each layer, there exist at least one cost opportunity. For slope and river layers several cost values can be entered according to the criteria specified. For instance, the criterion of slope layer is the slope of the region. Different cost values can be entered for each criterion. Criteria for slope layer are "slope is less than 10", "slope is between 10 and 30", "slope is between 30 and 60", "slope is less than 60"

Overlay Analysis: By using this icon, user can do the overlay operations according to the cost values specified by user. There are two kinds of overlay operations for this application. One is making overlay operations on whole terrain (terrain and road) or the other is only road. When user clicks this icon the window shown in Figure A-4 appears.

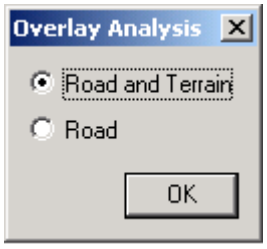


Figure A - 4 Overlay Analysis Window

The grid vertices are assigned to the cost values according to the overlay operation. The vertices that cannot be passed are assigned to cost value of “9”. For road only overlay operation, all of the vertices in terrain are assigned to cost value of “9”. Because, in road only terrain analysis case, terrain cannot be used as a passageway.

Starting Point: User select a vertex as a starting point by using this icon. This icon enable the user to draw an ellipse on the region user wants. The program then searches the region and selects the first vertex, for which cost is not "9", as the starting point.

Ending Point: User selects a vertex as an ending point by using this icon. This icon enable the user to draw an ellipse on the region user wants. The program then searches the region and selects the first vertex, for which cost is not "9", as the ending point.

Clear: It clears any paths and symbols on the map.

Draw Path: This button is used for finding the path. In order to find the path, starting point and ending points should be selected previously. The path found is shown in different colors.

Save Path: The path found is saved in a predefined file name structure. This icon saves the “ShortestPath” layer onto the current directory in “Pathxx” file name format.

Save Symbol: The symbols on the map are saved in a predefined file name structure. This icon saves the “Symbols” layer onto the current directory in “ArcSymbolxx” file name format.

Exit: Closes the application.

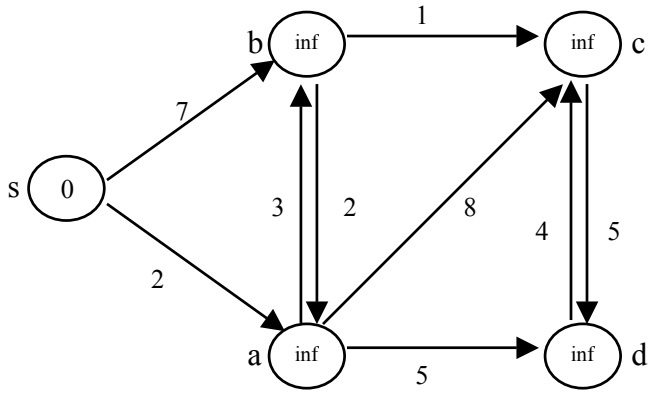
Appendix B: Dijsktra's Algorithm With Priority Queue

Dijkstra's Shortest Path with Priority Queue Algorithm and an example verifying this algorithm.

```
1     PriorityQueue.Snext(ǿ)
2     Snext.insert(s)
3     While (Snext <> ǿ) do
4       w = Snext.delete(Snext.min);
5       mark w as VISITED;
6       for each neighbor v of w do
7         if w is FREE and not VISITED then
8           if v.Dsmax > w.Dsmax + {v.TerrainCost} then
9             v.Dsmax := w.Dsmax + {v.TerrainCost};
10            v.PreviousCell := w;
11            if v ϵ Snext then
12              Snext.decreasekey(v);
13            else
14              Snext.insert(v);
15            end if
16          end if
17        end if
18      end for
19    end while
20    if goal g is marked as VISITED then
21      read off path by following
22      the PreviousCell pointers;
```

```
23      else
24          report failure;
25      end if
```

Example :

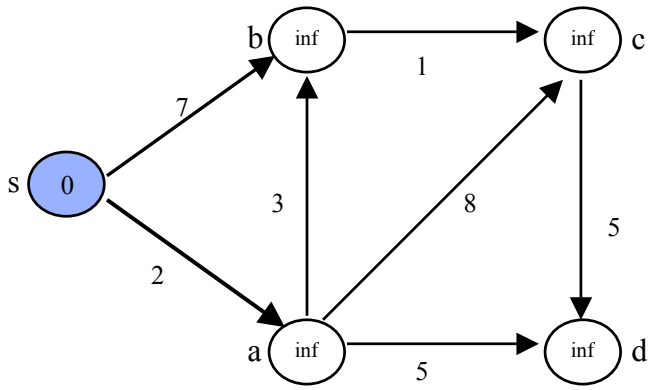


Step 0 : Initialization

v	s	a	b	c	d
v.Dsmax	0	inf	inf	inf	inf
v.PreviousCell	nil	nil	nil	nil	nil
v.VISITED	False	False	False	False	False

Priority Queue :

v	s
v.Dsmax	0

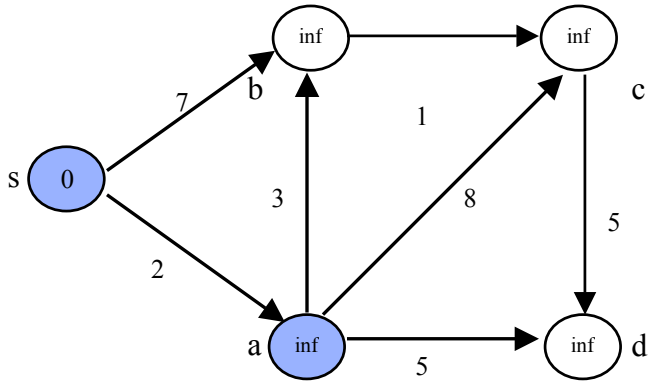


Step 1 : As Adj[s]={a,b} work on a, b and update information.

v	s	a	b	c	d
v.Dsmax	0	**2**	**7**	inf	inf
v.PreviousCell	nil	**s**	**s**	nil	nil
v.VISITED	**True**	False	False	False	False

Priority Queue :

v	a	b
v.Dsmax	**2**	**7**

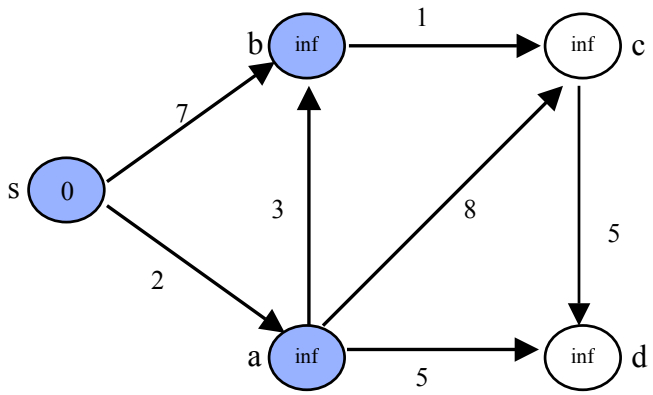


Step 2 : After step 1, a has the minimum cost in the priority queue. As Adj[a] = {b,c,d}, work on b,c,d and update information.

v	s	a	b	c	d
v.Dsmax	0	2	**5**	**10**	**7**
v.PreviousCell	nil	s	**a**	**a**	**a**
v.VISITED	True	**True**	False	False	False

Priority Queue :

v	b	c	d
v.Dsmax	**5**	**10**	**7**

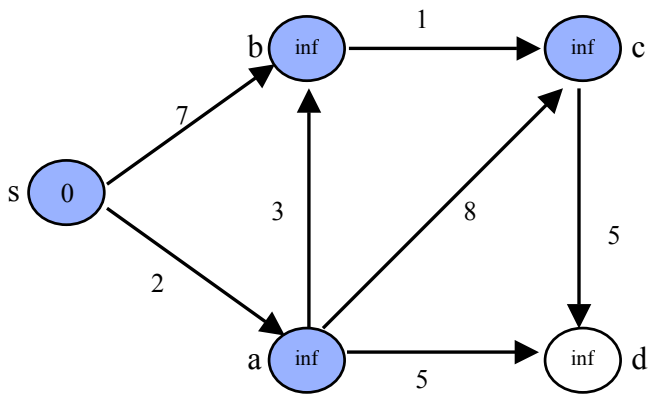


Step 3 : After step 2, b has the minimum cost in the priority queue. As Adj[b] = {c}, work on c and update information.

v	**s**	**a**	**b**	**c**	**d**
v.Dsmax	0	2	5	**6**	7
v.PreviousCell	nil	s	a	**b**	a
v.VISITED	True	True	**True**	False	False

Priority Queue :

v	**c**	**d**
v.Dsmax	**6**	7

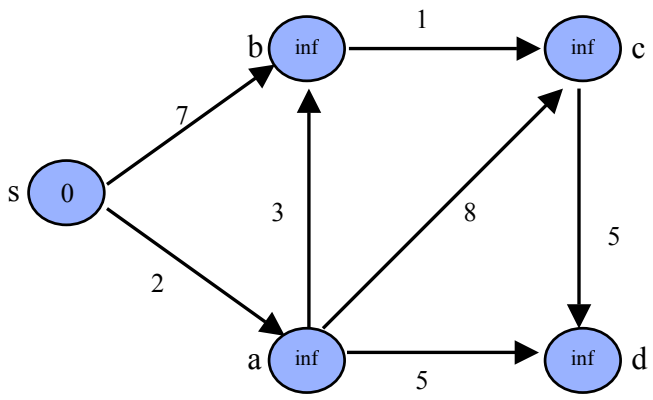


Step 4 : After step 3, c has the minimum cost in the priority queue. As Adj[c] = {d}, work on d and update information.

v	**s**	**a**	**b**	**C**	**d**
v.Dsmax	0	2	5	6	7
v.PreviousCell	nil	s	a	B	a
v.VISITED	True	True	True	**True**	False

Priority Queue :

v	**d**
v.Dsmax	7



Step 5 : After step 4, c has the minimum cost in the priority queue. As Adj[c] = {d}, work on d and update information.

v	**s**	**a**	**b**	**c**	**d**
v.Dsmax	0	2	5	6	7
v.PreviousCell	nil	s	a	b	a
v.VISITED	True	True	True	True	**True**

Priority Queue : Q = {}

v	
v.Dsmax	

Appendix C: A* Algorithm

Execution Steps Of A* Algorithm

```
1  Create a vertex containing the goal state vertex_goal
2  Create a vertex containing the start state vertex_start
3  Put vertex_start on the open list
4  while the OPEN list is not empty
5  {
6      Get the vertex off the open list with the lowest f and call it vertex_current
7      if vertex_current is the same state as vertex_goal we have found the solution; break
       from  the      while loop
8      Generate each state vertex_successor that can come after vertex_current
9      for each vertex_successor of vertex_current
10     {
11         Set the cost of vertex_successor to be the cost of vertex_current plus the cost
           to get to   vertex_successor from vertex_current
12         Find vertex_successor on the OPEN list
13         If vertex_successor is on the OPEN list but the existing one is better then
           discard this successor and continue
14         If vertex_successor is on the CLOSED list but the existing one is better then
           discard this successor and continue
15         Remove occurences of vertex_successor from OPEN and CLOSED
16         Set the parent of vertex_successor to vertex_current
17         Set h to be the estimated distance to vertex_goal (Using the heuristic function)
18         Add vertex_successor to the OPEN list
19     }
```

```
20      Add vertex_current to the CLOSED list
21 }
```

Appendix D: Hierarchical Path Planning Algorithm

```
1       Mark all level L cells as FREE
2       Find optimal path  p* through FREE level L cells
        /* Hierarchical Planning */
3       for l = L-1 downto 0 do
4               Mark level l neighborhood of  p*l+1 as FREE
5               Find optimal path p*l through FREE level l cells
6       End for
```

Appendix E: Prepare Grid Map Procedure.

```
sub grid_haritalari_hazirla

dim e as integer
dim k,m as float
dim z1,z2,kalan as float
set map coordsys window frontwindow()
e=frontwindow()
x1 = round(commandinfo(1),100)
y1 = round(commandinfo(2),100)
x2 = round(commandinfo(5),100)
y2 = round(commandinfo(6),100)
z1 = minimum(x1,x2)
z2 = maximum(x1,x2)
x1 = z1
x2 = z2
z1 = minimum(y1,y2)
z2 = maximum(y1,y2)
y1 = z1
y2 = z2

dialog title "Grid Menu"
        control statictext title "Grid interval:"
        control edittext into d value 100
        control okbutton
        control cancelbutton

if not commandinfo(1) then
        exit sub
end if

print "Koordinatlari yaz :"
print "x1:"+x1 + " y1:"+ y1+ " x2:"+x2 + " y2:"+y2
print "x farki : " + (x2-x1) + "y farki : " + (y2-y1)

drop table gridharitasi1
```

```
create table gridharitasi1
( deger SmallInt,
  no SmallInt,
  egim_pct SmallInt )
create map for gridharitasi1 coordsys window frontwindow()
add map layer gridharitasi1

set map layer gridharitasi1 editable on
set map layer gridharitasi1 display graphic
set map redraw off
numberOfXNodes = 0
numberOfYNodes = 0
set style symbol MakeSymbol( 49,blue, 12)
for k = x1 to (x2-d) step d
numberOfYNodes = 0
for m = y1 to (y2-d) step d
          numberOfYNodes = numberOfYNodes + 1
          Create point (k,m)
next
numberOfXNodes = numberOfXNodes + 1
if k mod 1000 = 0 then
          print k+"   "+m
end if
next
dim x, tsize as integer
set coordsys window frontwindow()
set map layer gridharitasi1 editable on
tsize=Tableinfo("gridharitasi1", 8)
for x= 1 to tsize
fetch rec x from gridharitasi1
update gridharitasi1 set no=x-1 where rowid=x
next

set map redraw on
add column gridharitasi1(egim_pct integer) from slope set to Lower where contains

print "numberOfXNodes :"+ numberOfXNodes  + "  numberOfYNodes :"+
numberOfYNodes
commit table gridharitasi1

end sub
```

Appendix F: Overlay Analysis Procedure.

```
sub Analiz
dialog title "Overlay Analysis "
control radiogroup title "Road and Terrain;Road" id 1
control okbutton calling overlay
end sub

sub overlay

dim zaman as integer
dim dikd as object
dim sx as integer
sx=readcontrolvalue(1)
do case sx

case 1
zaman = timer()
print " Baslangic: " + (timer() - zaman  )
update gridharitasi1 set deger =10

' road
select * from gridharitasi1,roadbuf  where gridharitasi1.obj within roadbuf.obj into good
fetch first from good
Do While Not EOT(good)
if (good.deger = 9 or yol = 9) then
        update good set deger=9 where rowid=good.rowid
else
  if (yol < good.deger ) then
                update good set deger=yol where rowid=good.rowid
  end if
end if
fetch next from good
loop
print " Road has completed for gridharitasi1: " + (timer() - zaman  )
```

```
'motorway
select * from gridharitasi1,motorwaybuf  where gridharitasi1.obj within
motorwaybuf.obj into good
fetch first from good
Do While Not EOT(good)
if (good.deger = 9 or mot = 9) then
        update good set deger=9 where rowid=good.rowid
else
  if (mot < good.deger ) then
                update good set deger=mot where rowid=good.rowid
  end if
end if
fetch next from good
loop
print " Motorway has completed for gridharitasi1: " + (timer() - zaman  )

'river
select * from gridharitasi1,river where gridharitasi1.obj within river.obj into sel
select * from sel where derinlik>1 into good
fetch first from good
Do While Not EOT(good)
if (good.deger = 9 or nehir3 = 9) then
        update good set deger=9 where rowid=good.rowid
else
  if (nehir3 < good.deger ) then
                update good set deger=nehir3 where rowid=good.rowid
  end if
end if
fetch next from good
loop
select * from sel where derinlik <=1 and derinlik >0.5 into good
fetch first from good
Do While Not EOT(good)
if (good.deger = 9 or nehir2 = 9) then
        update good set deger=9 where rowid=good.rowid
else
  if (nehir2 < good.deger ) then
                update good set deger=nehir2 where rowid=good.rowid
  end if
end if
fetch next from good
loop
select * from sel where derinlik <=0.5 into good
fetch first from good
Do While Not EOT(good)
if (good.deger = 9 or nehir1 = 9) then
```

```
        update good set deger=9 where rowid=good.rowid
else
  if (nehir1 < good.deger ) then
                update good set deger=nehir1 where rowid=good.rowid
  end if
end if
fetch next from good
loop
print " River has completed for gridharitasi1: " + (timer() - zaman  )

' village
select * from gridharitasi1,village where gridharitasi1.obj within village.obj into good
fetch first from good
Do While Not EOT(good)
if (good.deger = 9 or vil = 9) then
        update good set deger=9 where rowid=good.rowid
else
  if (vil < good.deger ) then
                update good set deger=vil where rowid=good.rowid
  end if
end if
fetch next from good
loop
print " Village has completed for gridharitasi1: " + (timer() - zaman  )

' tumulus
select * from gridharitasi1,tumulusbuf where gridharitasi1.obj within tumulusbuf.obj into
good
fetch first from good
Do While Not EOT(good)
if (good.deger = 9 or tum = 9) then
        update good set deger=9 where rowid=good.rowid
else
  if (tum < good.deger ) then
                update good set deger=tum where rowid=good.rowid
  end if
end if
fetch next from good
loop
print " Tumulus has completed for gridharitasi1: " + (timer() - zaman  )

' high_voltage
select * from gridharitasi1,high_voltagebuf where gridharitasi1.obj within
high_voltagebuf.obj into good
fetch first from good
Do While Not EOT(good)
```

```
if (good.deger = 9 or hig = 9) then
        update good set deger=9 where rowid=good.rowid
else
  if (hig < good.deger ) then
                update good set deger=hig where rowid=good.rowid
  end if
end if
fetch next from good
loop
print " High_voltage has completed for gridharitasi1: " + (timer() - zaman  )

'dam
select * from gridharitasi1,damp where gridharitasi1.obj within damp.obj into good
fetch first from good
Do While Not EOT(good)
if (good.deger = 9 or dam = 9) then
        update good set deger=9 where rowid=good.rowid
else
  if (dam < good.deger ) then
                update good set deger=dam where rowid=good.rowid
  end if
end if
fetch next from good
loop
print " Dam has completed for gridharitasi1: " + (timer() - zaman  )

'railroad
select * from gridharitasi1,railroadbuf where gridharitasi1.obj within railroadbuf.obj
into good
fetch first from good
Do While Not EOT(good)
if (good.deger = 9 or rail = 9) then
        update good set deger=9 where rowid=good.rowid
else
  if (rail < good.deger ) then
                update good set deger=rail where rowid=good.rowid
  end if
end if
fetch next from good
loop
print " Railroad has completed for gridharitasi1: " + (timer() - zaman  )

'bridge
select * from gridharitasi1,bridgebuf where gridharitasi1.obj within bridgebuf.obj into
good
fetch first from good
```

```
Do While Not EOT(good)
  if (kopru < good.deger ) then
                update good set deger=kopru where rowid=good.rowid
  end if
fetch next from good
loop
print " Bridge has completed for gridharitasi1: " + (timer() - zaman  )

'slope
select* from gridharitasi1 where egim_pct<10 into good
fetch first from good
Do While Not EOT(good)
if (good.deger = 10) then
        update good set deger=egim1 where rowid=good.rowid
end if
fetch next from good
loop
select* from gridharitasi1 where egim_pct<30 and egim_pct>=10 into good
fetch first from good
Do While Not EOT(good)
if (good.deger = 10) then
        update good set deger=egim2 where rowid=good.rowid
end if
fetch next from good
loop
select* from gridharitasi1 where egim_pct<60 and egim_pct>=30 into good
fetch first from good
Do While Not EOT(good)
if (good.deger = 10) then
        update good set deger=egim3 where rowid=good.rowid
end if
fetch next from good
loop
select* from gridharitasi1 where egim_pct>=60 into good
fetch first from good
Do While Not EOT(good)
if (good.deger = 10) then
        update good set deger=egim4 where rowid=good.rowid
end if
fetch next from good
loop
print " Slope has completed for gridharitasi1: " + (timer() - zaman  )

call  dugumyaz()

case 2
```

```
'create rect into variable dikd
(tableinfo(gridharitasi1,TAB_INFO_MINX),tableinfo(gridharitasi1,TAB_INFO_MINY))
(tableinfo(gridharitasi1,TAB_INFO_MAXX),tableinfo(gridharitasi1,TAB_INFO_MAXY))
update gridharitasi1 set deger =9

zaman = timer()
print " Baslangic: " + (timer() - zaman  )

' road
select * from gridharitasi1,roadbuf  where gridharitasi1.obj within roadbuf.obj into good
fetch first from good
Do While Not EOT(good)
  if ((good.deger<> 9) and (yol <>9) ) then
                if (yol < good.deger) then
                        update good set deger=yol where rowid=good.rowid
                end if
  else
                update good set deger=yol where rowid=good.rowid
        end if
fetch next from good
loop
print " Road has completed for gridharitasi1: " + (timer() - zaman  )

'motorway
'set map layer motorwaybuf editable on
'Create Object As Buffer From motorway Width 50 Units "m" Resolution 12 Into Table
motorwaybuf Group by Rowid
select * from gridharitasi1,motorwaybuf  where gridharitasi1.obj within
motorwaybuf.obj into good
fetch first from good
Do While Not EOT(good)
  if ((good.deger<> 9) and (mot <>9) ) then
                if (mot < good.deger) then
                        update good set deger=mot where rowid=good.rowid
                end if
  else
                update good set deger=mot where rowid=good.rowid
        end if
fetch next from good
loop
print " Motorway has completed for gridharitasi1: " + (timer() - zaman  )

'bridge
'select * from bridge into selection
'set map layer bridgebuf editable on
```

```
'Create Object As Buffer From selection Width 50 Units "m" Resolution 12 Into Table
bridgebuf Group by Rowid
select * from gridharitasi1,bridgebuf where gridharitasi1.obj within bridgebuf.obj into
good
update good set deger =kopru
fetch first from good
Do While Not EOT(good)
  if ((good.deger<> 9) and (kopru <>9) ) then
                if (kopru < good.deger) then
                        update good set deger=kopru where rowid=good.rowid
                end if
  else
                update good set deger=kopru where rowid=good.rowid
        end if
fetch next from good
loop
print " Bridge has completed for gridharitasi1: " + (timer() - zaman  )


call dugumyaz()

'select * from gridharitasi1 where deger=9 into selection
'shade window frontwindow() selection with deger values  9 Symbol (34,6316128,12)
default Symbol (40,0,12)
close table selection

end case

end sub
```

Appendix G: Dijsktra's Algorithm with Priority Queue Implementation Section of Visual C++ Program

```
class Pqueue
{
public:
  Pqueue(void);
  ~Pqueue(void);
  void setsize(int n);              // Initialize with 'n' elements
  void insert(double key, int n);      // Insert a node 'n' with key 'key'
  void extract(double &key, int &n);   // Extract a node 'n' with key 'key'
  void decreasekey(double key, int n); // Decrease the key of node 'n' to 'key'
  inline int empty(void) { return size==0; }

  inline bool exist(int s) { if (inverse[s] == -1) return false;  return true;}
private:
  struct qdata                  // Queue data
  {
    double key;
    int node;
  };
  int size;                     // Current size of heap
  int len;                      // Total allocated size
  qdata *A;                     // Heap data
  int *inverse;                 // Inverse mapping nodes->queue_pos

  inline int parent(int i) { return (i-1)/2; }
  inline int left(int i)   { return 2*i+1; }
  inline int right(int i)  { return 2*i+2; }
  void heapify(int);
  void heapup(int);
};

Pqueue::Pqueue(void)
{
  A = NULL;
  inverse = NULL;
```

```
  size = len = 0;
}

Pqueue::~Pqueue(void)
{
  delete[] A;
  delete[] inverse;
}

void Pqueue::setsize(int n)
{
  A = new qdata[n];
  inverse = new int[n];
  for (int i=0; i <n; i++) inverse[i] = -1;
  size = 0;
  len = n;
}

void Pqueue::insert(double key, int node)
{
  assert(size < len);

  int i = size++;

  while (i > 0  &&  A[parent(i)].key > key)
  {
    A[i] = A[parent(i)];
    inverse[A[i].node] = i;

    i = parent(i);
  }

  A[i].key = key;
  A[i].node = node;
  inverse[A[i].node] = i;
}

void Pqueue::extract(double &key, int &node)
{
  assert(size > 0);

  key = A[0].key;
  node = A[0].node;

  A[0] = A[--size];
  inverse[A[0].node] = 0;
```

```
  heapify(0);
}

void Pqueue::decreasekey(double key, int node)
{
  assert(A[inverse[node]].node == node);

  A[inverse[node]].key = key;
  heapup(inverse[node]);
}

void Pqueue::heapify(int i)
{
  int l = left(i);
  int r = right(i);
  int smallest = i;

  if (l < size  &&  A[l].key < A[i].key)
    smallest = l;
  if (r < size  &&  A[r].key < A[smallest].key)
    smallest = r;

  if (smallest != i)
  {
    qdata tmp = A[i];
    A[i] = A[smallest];
    inverse[A[i].node] = i;
    A[smallest] = tmp;
    inverse[A[smallest].node] = smallest;

    heapify(smallest);
  }
}

void Pqueue::heapup(int i)
{
  while (i > 0  &&  A[parent(i)].key > A[i].key)
  {
    int p = parent(i);

    qdata tmp = A[i];
    A[i] = A[p];
    inverse[A[i].node] = i;
    A[p] = tmp;
    inverse[A[p].node] = p;
```

```
      i = p;
    }
  }

// The program segment where Dijsktra's algorithm with priority queue implementation
// takes place

totalcost[sx][sy].cost = 0;

Q.insert(totalcost[sx][sy].cost, Network[sx][sy].name);
int don_say = 0;

while (!Q.empty())
{
  don_say++;

  Q.extract(cost,w);
  cevir(w, wx, wy);
  visited[wx][wy] = 1;

  for (int vx=(wx-1); vx<=(wx+1); vx++)
        for (int vy=(wy-1); vy<=(wy+1); vy++)
        {

          if ((getNode(vx,vy) < 9) && (!((vx==wx) && (vy==wy))))
          {

                if (!notFree[vx][vy]  && !visited[vx][vy])
                {

                      if (totalcost[vx][vy].cost >
                            ( totalcost[wx][wy].cost + Network[vx][vy].cost +
calculateDist(wx,wy,vx,vy)))
                      {
                            totalcost[vx][vy].cost =
                                  totalcost[wx][wy].cost + Network[vx][vy].cost +
calculateDist(wx,wy,vx,vy);
                            preCell[vx][vy] = w;

                            if (Q.exist(Network[vx][vy].name))
Q.decreasekey(totalcost[vx][vy].cost, Network[vx][vy].name);
                            else Q.insert(totalcost[vx][vy].cost,
Network[vx][vy].name);

                      }
                }
```

```
            }
          } // end for
} // end while

 //  This will search the path back, to see by which route we reached the goal-node.
 //  It is not possible to reach the goal-node from the start-node.

if (visited[gx][gy] == 1)
{
  result<<goal<<endl;
  currentNode = preCell[gx][gy];
  while (currentNode != start)
  {

          result<<currentNode<<endl;
          cevir(currentNode, cx, cy);
          currentNode = preCell[cx][cy];
  }
  result<<start<<endl;
  cout << "sonnnn" << endl;
  result.close();
}
else printInvalidResult();
```

Appendix H: Finds Shortest Path By Using Spatial Analyst Tool Of ArcMap

```
Sub FindShortestPath()

    'Get the focused map from MapDocument
    Dim pMxDoc As IMxDocument
    Set pMxDoc = ThisDocument
    Dim pMap As IMap
    Set pMap = pMxDoc.FocusMap

    'Get the input source data from the first layer in ArcMap
    Dim pSourceGeoDataset As IGeoDataset
    Dim pLayer As ILayer
    Dim pFeatureLayer As IFeatureLayer
    Dim pRasLayer As IRasterLayer

    Set pLayer = pMap.Layer(0)
    If TypeOf pLayer Is IFeatureLayer Then
       'MsgBox "Point is Feature"
       Set pFeatureLayer = pLayer
       Set pSourceGeoDataset = pFeatureLayer.FeatureClass
    ElseIf TypeOf pLayer Is IRasterLayer Then
       'MsgBox "Point is Raster"
       Set pRasLayer = pLayer
       Set pSourceGeoDataset = pRasLayer.Raster
    Else
          Exit Sub
    End If

    'Get the COST (backlink) raster from the third layer
    Dim pCostDataset As IGeoDataset

    Set pLayer = pMap.Layer(1)
    If Not TypeOf pLayer Is IRasterLayer Then
       Exit Sub
    End If
    Set pRasLayer = pLayer
```

```
Set pCostDataset = pRasLayer.Raster

'Get the COST (backlink) raster from the third layer
Dim pGoalGeoDataset As IGeoDataset

Set pLayer = pMap.Layer(2)
If TypeOf pLayer Is IFeatureLayer Then
   'MsgBox "Point is Feature"
   Set pFeatureLayer = pLayer
   Set pGoalGeoDataset = pFeatureLayer.FeatureClass
ElseIf TypeOf pLayer Is IRasterLayer Then
   'MsgBox "Point is Raster"
   Set pRasLayer = pLayer
   Set pGoalGeoDataset = pRasLayer.Raster
Else
      Exit Sub
End If

'Create a RasterDistanceOp operator
Dim pDistanceOp As IDistanceOp
Set pDistanceOp = New RasterDistanceOp

' Declare the output raster object
Dim pOutputRaster As IGeoDataset

Dim baslangic
baslangic = Time
Dim basyaz As String
basyaz = Str(baslangic)

' Calls the method
Set pOutputRaster = pDistanceOp.CostDistance(pSourceGeoDataset, pCostDataset)
Dim distRaster As IRaster
Set distRaster = pOutputRaster

' Declare the output raster object
Dim pOutputRaster1 As IGeoDataset

' Calls the method
Set pOutputRaster1 = pDistanceOp.CostBackLink(pSourceGeoDataset, pCostDataset)
Dim backRaster As IRaster
Set backRaster = pOutputRaster1

Dim pOutRasLayer As IRasterLayer

'Find the shortest path
```

```
    Dim pOutRaster As IRaster
    Set pOutRaster = pDistanceOp.CostPath(pGoalGeoDataset, pOutputRaster,
pOutputRaster1, esriGeoAnalysisPathForEachCell)

    Dim son
    son = Time
    Dim sonyaz As String
    sonyaz = Str(son)

    Set pOutRasLayer = New RasterLayer
    pOutRasLayer.CreateFromRaster pOutRaster
    pMap.AddLayer pOutRasLayer



    'Print " Son " + yaz

    Open "TESTFILE.txt" For Output As #1    ' Open file for output.
    ' The second word prints at column 20.
    Print #1, "bas"; Tab(20); basyaz
    Print #1, "son"; Tab(20); sonyaz
    Close #1    ' Close file.


End Sub
```