DEVELOPMENT OF A GIS SOFTWARE FOR EVALUATING NETWORK RELIBILITY OF LIFELINES UNDER SEISMIC HAZARD

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

BY

LÜTFİ ODUNCUOĞLU

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
GEODETIC AND GEOGRAPHICAL INFORMATION TECHNOLOGIES

DECEMBER 2010

Approval of Thesis:

DEVELOPMENT OF A GIS SOFTWARE FOR EVALUATING NETWORK RELIABILITY OF LIFELINES UNDER SEISMIC HAZARD

submitted by **LÜTFİ ODUNCUOĞLU** in partial fulfillment of requirements for degree of **Master of Science In Geodetic and Geographical Information Technologies, Middle East Technical University** by,

Prof Dr.Canan Özgen
Dean, Graduate School of Natural and Applied Sciences _____

Prof Dr. Vedat Toprak _____
Head of Department, **Geodetic and Geographical Information Technologies**

Prof. Dr. H. Şebnem Düzgün _____
Supervisor, **Mining Engineering Department, METU**

Assoc. Prof. Dr. Sevtap Kestel _____
Co-Supervisor, **Graduate School of Applied Mathematics. METU**

Examining Committee Memebers:

Prof Dr.Vedat Toprak _____
Geological Engineering Department, METU

Prof. Dr. H. Şebnem Düzgün _____
Mining Engineering Department, METU

Assoc Prof. Dr. A. Sevtap Kestel _____
Graduate School of Applied Mathmetics, METU

Assoc. Prof. Dr, Ahmet COŞAR _____
Computer Engineering Department, METU

Assoc. Prof. Dr. Nurünnisa Usul _____
Civil Engineering Department, METU

Date: 27/12/2010

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 cited and referenced all material and results that are not original to this work.

Name, Last Name : Lütfi Oduncuoğlu

Signature :

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to Prof. Dr. Şebnem Duzgun for her supervision and guidance.

My deepest appreciation to Assoc. Prof. Dr. Sevtap Kestel in helping me to broaden my view and knowledge.

I am heartily thankful to my friends, Gokcen Guner and Onur Yurtsever, whose encouragement, guidance and support from the initial to the final level enabled me to develop an understanding of the subject.

My heartiest thanks to Tugce Senturk in editing my thesis.

And my deepest gratitude to my parents in supporting me.

Lastly, I offer my regards and blessings to all of those who supported me in any respect during the completion of the thesis.

ABSTRACT

DEVELOPMENT OF A GIS SOFTWARE FOR EVALUATING NETWORK RELIBILITY OF LIFELINES UNDER SEISMIC HAZARD

Oduncuoglu, Lutfi
M. Sc. Department of Geodetic and Geographical Information Technologies
Supervisor : Prof. Dr. H. Sebnem Duzgun
Co-Supervisor: Assoc. Prof. Dr. Sevtap Kestel

December, 2010, 111 pages

Lifelines are vital networks and it is important that those networks are still be functional after major natural disasters such as earthquakes. The goal of this study is to develop a GIS software for evaluating network reliability of lifelines under seismic hazard. In this study, GIS, statistics and facility management is used together and a GIS software module, which constructs GIS based reliability maps of lifeline networks, is developed by using geoTools. Developed GIS module imports seismic hazard and lifeline network layers in GIS formats using geoTools libraries and after creating a gridded network structure it uses a network reliability algorithm, initially developed by Yoo and Deo (1988), to calculate the upper and lower bounds of lifeline network reliability under seismic hazard. Also it can show the results in graphical form and save as shape file format. In order to validate the developed application, results are compared with

a former case study of Selcuk (2000) and the results are satisfactorily close to previous study. As a result of this study, an easy to use, GIS based software module that creates GIS based reliability map of lifelines under seismic hazard was developed.

Keywords: GIS, Seismic Hazard, Facility Management, Network Reliability

ÖZ

SİSMİK RİSK ALTINDAKİ CANDAMARLARININ AĞ GÜVENİLİRLİĞİNİN DEĞERLENDİRİLMESİ İÇİN BİR CBS YAZILIMI GELİŞTİRİLMESİ

Oduncuoğlu, Lütfi
Yüksek Lisans, Jeodezi ve Coğrafi Bilgi Teknolojileri Bölümü
Tez Yöneticisi : Prof. Dr. Şebnem Düzgün
Ortak Tez Yöneticisi : Doç. Dr. Sevtap Kestel

Aralık, 2010, 111 Sayfa

Candamarları, hayati öneme sahip ağlardır. Bu ağların deprem gibi büyük doğal afetler sonrasında işlevlerini yerine getirmeye devam etmeleri önemlidir. Bu çalışmanın amacı; sismik tehlike altındaki candamarlarının ağ güvenilirliği için bir CBS yazılımı geliştirilmesidir. Çalışmada, CBS, istatistik ve tesis yönetimi birlikte kullanılmış ve geoTools kullanılarak candamarı ağları için CBS tabanlı güvenilirlik haritaları oluşturan CBS yazılım modülü geliştirilmiştir. Geliştirilen CBS modülü, geoTools kütüphanelerini kullanarak sismik tehlike ve candamarı ağ tabakalarını CBS formatında almaktadır. Hücreli (Gridli) ağ yapısı oluşturulduktan sonra, ilk olarak Yoo ve Deo (1988) tarafından geliştirilen ve sismik tehlike altındaki can damarı ağ güvenilirliğinin alt ve üst sınırlarını hesaplamakta kullanılan ağ güvenilirliği algoritması kullanılmaktadır. Ayrıca sonuçlar grafik formatta gösterilip, shape dosya biçiminde kaydedilir. Geliştirilen uygulamanın test edilebilmesi için, sonuçlar Selcuk'un (2000) önceki

bir çalışması ile karşılaştırılmış; elde edilen değerlerin önceki çalışmayla uyumlu olduğu görülmüştür. Yapılan bu çalışmanın sonunda, sismik tehlike altındaki candamarlarının CBS tabanlı güvenilirlik haritalarını meydana getiren, kullanımı kolay, CBS tabanlı yazılım modülü geliştirilmiştir.

Anahtar Kelimeler: CBS, Sismik Tehlike, Tesis Yönetimi, Ağ Güvenilirliği

TABLE OF CONTENTS

LIST OF FIGURES

FIGURES

LIST OF TABLES

TABLES

CHAPTER 1

INTRODUCTION

1.1 Importance of the Problem

Seismic hazard studies have mostly studied over the safety of individually built facilities such as hospitals, dams, power plants, rather than the network infrastructures coming out from the "central" facility. Indeed, any seismic hazard to those central facilities mostly causes system failure or power down. However, evaluating the overall responses to seismic loads of such spatially distributed network infrastructures require considering the performance of all components of such a system. Likewise, such systems spatially lie over long distances and thus, for the same earthquake, various components of the same system may be subjected to different seismic loading.

During last twenty years, more attention has been given to application of probabilistic simulation models and procedures for the calculation of quantitative evaluation of reliability of lifelines under earthquake loads. Transportation systems, pipelines, communication and power transmission systems are examples of lifelines. During such destructive earthquakes or afterwards, it is important to forecast secondary disasters such as fires. The damage to lifelines during Kobe earthquake in Japan is one of the best examples of this fact. In Kobe earthquake, major highways are damaged at 1257 different points, water of one million, natural gas of 857000, electricity of 916000 households were cut and telephone of 410000 subscribers were disconnected (Aydinoglu and Erdik, 1995). After the earthquake, it has been declared that at 234 different places, fire

started mainly because of natural gas release and electricity sparks. Total number of fires were 531 and the area that was burned was about 1 km^2 (Aydinoglu and Erdik, 1995). While those examples taken under consideration, network reliability is an important tool for forecasting the damage after such natural disasters.

Network reliability analysis is mainly the assessment of the performance of a network in case of its ability to resist the failure of its components. A network modeled as GIS based vector data, contains a set of nodes and vertices representing either directed or undirected communication links and information about those nodes and vertices. In this study lifelines are idealized as equivalent networks where the link and node capacities can be selected either deterministic or random. Indeed, those network infrastructures are exposed to various manmade and natural hazards treating their safety and performance.

In today's technology Geographic Information System (GIS) tools can be used for doing analysis of network reliability under seismic hazard. GIS is a cross-discipline involving information science, spatial and earth science. It combines geospatial data with computer technology. By means of the setup and operation of systems and model analysis, it provides useful information for resource environment, region planning, decision-making, and management. In recent years, GIS has been widely used in many fields such as engineering, disasters prevention, natural resources management, simulation, and prediction of natural disasters and emergency response system of disasters, etc. Moreover, GIS can be used together with facility management. Some of these earliest applications of GIS in facility management were related to pavement management at airports, municipal water and wastewater infrastructure, and electric utility distribution. For example facility managers of the US Air Force have developed a standardized set of GIS layers to support the management of Air Force bases. In such system GIS brings considerable ease and support to management of the relevant facility.

The possible use of GIS is not limited to management of these lifeline systems, but GIS can also be used with facility management for other analysis. Although, many applications have been developed for facility management in GIS, there aren't any considerable tool in GIS for reliability of those structures under seismic hazard. Since GIS can be defined as a system of hardware, software and procedures developed for sustaining the acquisition, management, analysis, modeling and display of spatially referenced data for solving complex planning and management problems; it is a powerful tool for dealing with the spatial data. Therefore, GIS can be used to decrease damage on structure such as lifeline networks, caused by natural disasters. Indeed, GIS has many roles in such an analysis, at first glance, it is a vital tool for encircling the spatial characteristics of network systems and the network topology can be visualized. The last but not the least beneficial role of the GIS is its usage for visualization of outcomes of the analysis in various file formats which allow user to analyze spatial characteristics of network systems.

GIS is an ideal tool for network reliability analyses under seismic hazard, since it is suited for dealing with the spatial data. Moreover, visualizing eases to understand the results and users can make queries over results by using just only one tool because of database integration of GIS. Therefore, GIS can be used to evaluate network reliability under seismic hazard by using the spatial data management. In scope of this study, a GIS based computational tool is developed that can assess reliability of lifelines under seismic hazard based on former study done by Selcuk (2000).

Developed software combines GIS, statistics and facility management. For this purpose, seismic hazard, network and seismic hazard map layers are taken as inputs. By overlapping layers a square mash grid is constructed depending on the resolution of the seismic hazard layer. Network layer is partitioned based on square mash and line segments are obtained which are used for reiliability estimation of network components. Knowing the component reliabilities, the network reliability can be calculated and reliability map of lifeline is presented.

1.2 Aim of the study

There are many applications and studies about lifeline reliability under seismic hazard. However, there exists no application that combines GIS, lifeline reliability and facility management. Nowadays, GIS is popular and widely used concept for earth dependent data. Moreover, GIS keeps not only schema of a network, but stores position on earth in a specific projection such as WGS 84 and attributes of those structures. This makes it easier to join two layers such as seismic hazard layer and network layer for this study depending on earth positioning. Furthermore, since applications like facility management mostly use GIS based approaches nowadays, GIS based risk analysis tools like the one in this study are easy to interoperate with facility management tools.

The basic aim of this study is to develop a software module for generating GIS based reliability maps of lifelines. For this purpose, a GIS software module is developed by using geoTools. Although a few former studies were conducted on this subject, these did not truly use GIS and their inputs and results were text based, even the software in those studies were not easy to understand and use. For this aim, former study of Selcuk (2000) is implemented to GIS environment for constructing the reliability maps of lifeline networks.

CHAPTER 2

LITERATURE REVIEW

2.1 GIS Related Literature

In recent years due to the new developments in GIS applications and platforms, several researchers have studied GIS based applications on reliability of structures under seismic hazard, facility management, and developing software modules within GIS.

Toprak et al. (1999), used GIS to visualize damages on water pipeline systems after earthquakes in San Francisco. For this purpose they stated the damage on network as repair rate. By gridding the study area, they developed the repair rate contours within GIS, thus they determined the number of repairs in each grid. They developed a hyperbolic relationship model between visual display of high damage areas and grid size. This relationship helps the GIS users to obtain sufficiently refined and easily visualized map of damage patterns.

Ertugay and Duzgun (2006) concentrated on the emergency accessibility which was the most vital and important component of disaster preparedness. For this purpose, 3 different accessibility measurement techniques (Zone Based, Isochronal Based, and Raster Based Techniques) were analyzed within GIS environment. Eskisehir urban area was used as a case study and accessibility vulnerability index was created.

Kemec and Duzgun (2006) created effective 3D visualization tools and 3D city models, which have improved the viewer's understanding of the information

contents, for earthquake vulnerability level of each building. To achieve this goal, 3D visualization methodologies of GIS, Remote Sensing (RS) and Computer Aided Design (CAD) systems were used. Spatial decision support systems (SDSS) for earthquake risk assessment were developed for the Cumhuriye Quarter of Eskisehir City.

In contrast to the studies that hitherto done (earthquake risk estimations had been solely performed by evaluating the seismic hazard and prediction of buildings to be damaged. Hence, seismic hazard consequences had been considered based on only building damage), Duzgun and Yucemen (2007) took the other elements of the urban area, such as inhabitants, urban economy, infrastructure, cultural and historical heritage into account with using integrated risk models. They constituted a spatial urban disaster risk model and implemented it to the Odunpazari Municipality in Eskisehir and risk maps are found.

Duzgun et al. (2009) indicated that, to be able to do urban 3D visualization, interoperable data, and approaches should be combined. For this purpose, they presented a rule based approach to find a link that was between the types of potential hazards and the relevant urban features, and this link could be described by a set of criteria which depended on them. They claimed that the urban model features they used were compatible with detailed definitions specified in CityGML.

Lembo et al. (2009) offered a general idea about migration of a traditional graphic based application to a modern GIS application, which is a spatially enabled database application to assess possible damage of a water network system after an earthquake. The application developed in this study was based on a former one developed by authors (Graphical Iterative Response Analysis for Flow Following Earthquakes (GIRAFFE)). Object oriented programming, storage of spatial data within a database management system and performance of analytical processing in computer memory was used for this new software. As a result the developed application showed better performance by using modern

approaches of GIS processing, in terms of ease-of-use, speed, system modification and maintenance.

O'Rourke et al. (1999), studied characteristics of Los Angeles water system about earthquake effects and pipeline damage based on GIS. One of the main aims of this study was to identify components and facilities which are under high seismic hazard. This information was used for damage assessment and deployment of emergency services and system restoration resources. The study resulted in a comprehensive GIS characterization of Los Angeles water pipeline network including earthquake damage patterns and distributions of seismic parameters, permanent ground deformation. This characterization would be used for forthcoming system analysis and reliability assessment.

Yamazaki (2001), studied over seismic monitoring and early damage assessment in Japan. He used GIS as a platform for damage assessment and loss estimation both real and simulated data. Seismic information gathering and network alert (SIGNAL) an early damage assessment system, was combined with GIS. This combined system was used for natural gas network. Together with the actual data, the result of GIS damage estimation was utilized for whether or not to shut off the gas supply to avoid secondary disasters in a seismic event.

Chen and Cherng (1997), designed a new model for network load management. For this purpose they developed new software that combines facility management, automated mapping, and geographical information systems. This program provides system information to engineers about the distribution system itself and load of the distribution system. This developed application is used for preventing distribution system from damage depending on overloading or inefficient operations result from very low loads.

2.2 Reliability Related Literature

In literature there exist many studies on seismic reliability of structures depending on seismic hazard. Those studies are mainly over single structures

such as hospitals. On the other hand, there are a few studies over structures like pipeline systems.

Yamazaki et al. (1998), studied early damage assessment and monitoring systems in Japan. They discussed several recent vintage and future directions of real-time earthquake hazard assessment. They claimed, damage assessment means calculating damage statistics using empirical formulas; in fact those estimations have wide variability scale, estimating range of damage actually better than predicting a single number. Furthermore some researchers have done much more detailed advanced studies using methods from other disciplines.

Bendimerad (2001), discussed that functionality of loss estimation models improved due to advances in information and computer technology such as GIS technologies. GIS makes it possible to display of input and output. Moreover, loss estimation provides key information on potential damages and losses of buildings, water, power and gas systems and other vital systems. He adds that, with GIS systems make it possible to build developed loss estimation functions. But those estimations do not work well for megacities, because of lack of inventory and engineering data, even lack of expertise. Moreover, cooperating use of damage and loss estimation with expertise it is possible to standardize their relationship with hazard and vulnerability.

Padgett et al. (2007) studied relationships between bridge damage and the resulting loss of functionality of the bridge. They claimed that these are critical for assessing the impact of an earthquake event on the performance of the transportation network. They used these data to assess the probability of meeting various damage states and they expressed in terms of restoration of functionality, and subsequently facilitate the refinement of component limit-state capacities in order to develop analytical fragility curve. Hence they improved the assessment of transportation network performance. For modeling the potential decision model an expert opinion was used since the subjective nature of the problem. Hence, they calculated the damage state exceedance probabilities for various kinds of bridge component damage, where the damage

states were given in terms of maximum allowable level of traffic carrying capacity. They claimed exceedance probabilities may be represented by fragility curves, which were derived from functionally consistent limit states.

Ellingwood (2001) studied over special considerations on structural design and evaluation of buildings and other facilities with regard to their ability to withstand the effects of earthquakes, specifically related to probability-based codes design and reliability-based state assessment of existing buildings. Moreover, he compared his method with current deterministic approaches for post earthquake building condition assessment. He also examined importance of inherent randomness and uncertainty modeling in forecasting the building performance through a fragility assessment of a steel building with welded connections. Moreover, his comparisons on predicted and observed building damage indicates that despite advances in non-linear dynamic analysis and structural modeling, still there were limitations for using deterministic approaches for post-earthquake building condition assessment. Hence, a probabilistic analysis of building response is critical in providing ideas of building behavior. He also claimed that, reliability tools are quite sophisticated but they are not economically feasible for condition assessment of vast majority of building.

Yoo and Deo (1988) compared a number of network reliability algorithms and commented that the best one was Dotson and Gobien (1979) algorithm. This algorithm gave the results for upper and lower bound in a short time. This technique for finding out the terminal reliability or probabilistic networks is obtained. This technique utilized set-the-oretic concepts division of space of graph realizations in way that allows fast evaluation for the source to terminal probability (Dotson, et al, 1979).

Selcuk and Yucemen (1998), idealized a lifeline network as an equivalent network with capacity of its elements being random and spatially correlated and a comprehensive probabilistic model for the assessment of the reliability of lifeline under earthquake loads was developed. By using the past earthquake

occurrence data, a probability distribution of seismic hazard, that the network was exposed was derived. The seismic hazard method they used, was a modified version of “classical” seismic hazard analysis model. In case studies they had satisfactory results by using their new approach.

Selcuk and Yucemen (1998) studied lifeline reliability under seismic hazard with multiple sources. For this purpose they combined three aspects which were hazard methodology, capacity determination techniques, and network reliability assessment methods. Combining these three aspects in one probabilistic model, made it possible to evaluate the reliability of any lifeline network under seismic hazard. Therefore, the main aim of study was to present probabilistic model for evaluation of the seismic reliability of a lifeline having multiple seismic sources.

CHAPTER 3

THEORETICAL BACKGROUND

3.1 GIS Platforms

Open source solutions are available for many types of commercial applications, in almost every area of use, such as databases, web applications, GIS and more. Using open source applications has a number of benefits, which can be listed as price, security and developing source-code based applications. Price is an advantage since open source software is totally free of charge. In security issues, because the source code of those applications are open to everyone, security updates are released quicker than commercial applications. Another advantage of using open source software is, users can develop source-code based applications. Moreover, developing applications on open source platforms has other benefits for developing on enterprise applications. One of the most important advantages of developing on open source platforms is the large amount of documents and support available. Such support can be taken from developers' community via e-mail groups. As a result, considering all these factors, open source was selected for this study.

In order to develop a GIS based application, a GIS platform should be selected firstly. Many documents or resources comparing these GIS applications are available on the internet and Cascadoss Project is one of them. This project evaluates desktop GIS applications according to marketing, technical and economical potentials. For each potential area related criteria are defined and evaluation is done on basis with total grade of 60. When the first criterion

marketing potential is considered, GRASS and QuantumGIS are the applications with highest marketing potential. uDIG, gvSIG and OpenJUMP are other desktop GIS applications with significant growth potential. On the other hand, as the technical potential is considered GRASS, QGIS and uDig come to the forefront, while gvSIG and OpenJUMP are promising ones. Concerning the economical potential QGIS and OpenJUMP are significant ones and again the GRASS, gvSIG and uDIG are promising applications. Under the enlightening of this information the potential GIS desktop application that can be used in this study are GRASS, Quantum GIS (QGIS), uDig, gvSIG and OpenJUMP. However, other then these desktop applications there exists some tool packages for developing GIS applications, such as geoTools. Using geoTools, plug-ins can be developed for uDIG, OpenJUMP and other Java based GIS applications. Therefore, one of these GIS applications can be used in this study.

The earliest open source GIS application is GRASS (Geographic Resources Analysis Support System) which was developed by U.S. Army Corps of Engineer in 1982. This application supports both raster and vector layers. On the other hand it uses topological vector data model, in which areas are expressed with boundaries and centroids. This topological structure does not allow overlapping boundaries within a single layer.

Quantum GIS (QGIS) is an open source software based on KDE Linux desktop environment, which has GPL (General Public Licensed). QGIS project was first started by Gary Sherman in 2002. QGIS can run on different operating systems such as Mac OS X, UNIX variants and Microsoft Windows. Moreover, it requires less system resources than other GIS applications. The disadvantage of this application is lack of python coding documentation, which arises from changing plug-in coding language from C++ to python by developers.

uDig (User-friendly Desktop GIS) is an LPGL (Lesser General Public Licensed) GIS application. This application is based on Eclipse java development environment. Developers of uDig aim to build a user friendly, desktop located, internet oriented, and a GIS ready platform. The disadvantages of this

application result from JRE (Java Runtime Environment), depending on the slight differences between open JDK (Java Development Kit) and Sun Java JDK, java sources.

gvSIG is another open source GIS platform, this application is developed to handle complex planning problems. This application is developed by a small community, indeed this is the disadvantage of this GIS platforms. Updates and bug fixes are not released regularly, moreover building plug-ins for this application is not very easy depending on lack of documentation.

OpenJUMP is a recently developed open source GIS platform based on Java. This application is still under development. This situation is the basic disadvantage of this GIS platform, bug fixes and updates are not regularly released. Therefore, it is not possible to build complex plug-ins for OpenJUMP for the moment.

GeoTools is an open source GIS toolkit for developing standards compliant solutions. It provides an implementation of Open Geospatial Consortium (OGC) specifications as they are developed. GeoTools is a contributor of the GeoAPI project -a vendor-neutral set of Java interfaces derived from OGC specifications- and implements a subset of those. It is written in Java and currently is under active development. It is used by GeoServer and UDig projects.

Between the GIS platforms listed above, two of them stand out due to their advantages, QGIS and geoTools. The main of this study is to develop a plug-in that provides seismic analyses results of lifelines within GIS. To that end, these two GIS platforms are compared with respect to their features.

Since, the proposed solution starts with gaining the required data from the input files, the code built for this purpose should be capable of interacting with widely used GIS software and able to read common GIS data formats; QuantumGIS and geoTools commonly give users opportunity to build custom plug-ins for different needs. Furthermore, such tools have both extension capability for custom computation on different data sources. On the other side, whatever the GIS

platform used, building extensions require perfect understanding of the platform itself and delicate documentations on both the system itself and how to write extensions. Despite they give users to understand, record, visualize, manage and compute GIS data, when QuantumGIS and geoTools are compared, geoTools has better and more user friendly development interface and better documentation and support on not only how to use but also on how to extend. Moreover, QuantumGIS has some disadvantages that have potential for difficulty or problems in writing extensions. In the first instance, in recent versions the system allows plug-ins only in Python language, although the platform itself coded in C++. This brings the potential problem of ability to embed the code parts of the platform and reuse them in the plug-ins. Moreover, the architecture with two different coding languages C++ and Python makes debugging of the whole system more difficult in case of problems, thus makes solution of the possible problems harder. Also use of different languages in the same platform comes up with potential problem of contradiction in libraries. Furthermore, uses of different languages make documentation and understanding of the whole system more complicated. This causes potential difficulty for further advances of proposed system and for future studies. Unlike QuantumGIS, geoTools is written in java and also has extension capacity in the same language. Besides, geoTools have more detailed documentation on how to write plug-ins. The last but not the least benefit is that, since Java is a platform independent high level programming language both the GIS platform itself and the written extension modules, whether in source code or complied form, can run on any operating system and processer architecture.

As a result, due to detailed documentations, unified code language and the platform independent nature, considering also the well documented advanced support and wide capabilities of Java language itself, geoTools is chosen as the main development platform for the solution of the problem. In coding as expected and required Java language is used. Moreover, one of the chief developers of QGIS, Gary Sherman was gotten in contact with, about the issue

and he suggested using of geoTools, because of the complexity of the plug-in developed in this study.

As the main platform to use in solution is decided, the next step is to define the data format to be used as input and output data file formats. Since the study mainly considers use of available hazard data and the already built lifeline structure, use of already defined GIS data formats are preferred. The common GIS platforms mostly consider a general map format called geotiff in order to save data with respect to position in a defined space. In this data format the properties are visualized as colors at pixels of the image where the pixels represent the position. Then geotiff format is a suitable choice for storing hazard map. In addition, geotiff is a widely used common GIS data format and as a result in this study the input data format is used in geotiff format. On the other side, lifelines also should be represented in proper formats. Because the data for lifelines are usually designed in terms of geometrical shapes, shape file format which is designed to store vector data, is a more suitable choice. Since this format is also a widely used one, it is used as the second input file type to be used for lifeline data network.

In order to develop such software, the first step is to build tools that can read data from the input files load them into running program with usable structure. On purpose, the developed system uses libraries written in Java available for use under GIS platform. Then as the data are obtained from the files the next step is to combine the shapes from the shape file with the characteristic data. In fact, this part is the real place where the product of this research starts. The built Java code first should input the hazard map and vector data from the shape file, and then compile these two data. In this study, since the aim is to get the reliability profile of the lifeline under seismic hazard, the seismic hazard layer is converted into lifeline vector data.

As it becomes possible to get and process input data, functions are written in order to combine these. Then the first step is to take an ordinary shape file such

as a map and geo referenced seismic hazard layer file, and to visualize the property gained from the seismic hazard layer file on the shape file.

3.2 Seismic Hazard Analysis

Seismic hazard analysis gives numerical information that can be used for designing and checking safety of structures face with seismic activity. In general, in earthquake protection measures, the vital factor is how frequent the occurrence of the damaging earthquake.

Since the former earthquakes are known from databases and from past seismicity records, expected rate of seismic activity at a specific region based on probabilistic models can be calculated. There are a number of probabilistic methods in past to estimate the probability of future occurrences of earthquakes.

The numeric values of seismic hazard is mostly given as probability, that the site experiences ground motion intensities (given as intensity or peak ground acceleration of earthquake severity) larger than a given value. For calculation of seismic hazard, firstly it is necessary to analyze the available records on former earthquakes. For doing this, earthquake catalogs of instrumental records and historical earthquakes are needed to be examined. By the help these data, a database is created that includes location, occurrence time, and size of former seismic activity. Consequently, it can be used as an input for the stochastic model that gives an estimation of the future seismic activity, expected at a specified region over a specified time interval, as an output.

A future earthquake is unpredictable, i.e. the occurrence time, location, size and some other characteristics cannot be known. Random variables of probabilistic seismic hazard model are the seismicity parameters such as magnitude of a future earthquake, its location, occurrence time, and space. Calculation of probability distribution of those random variables and estimation of the required parameters are based on former seismic activities data, which are used in the procedure of probabilistic seismic hazard analysis. Attenuation equation, which

describes the decay of ground motion from the earthquake origin to the site associated with the uncertainty relationships, is also considered in the analysis.

The attenuation of energy from the earthquake source to the site is described through an empirical model which is referred as the attenuation function. In last phase, seismic hazard curves are estimated expressing the probability of exceeding various levels of specified earthquake intensity or ground motion parameter.

As stated, the probability of exceedance of a specified peak ground acceleration level at a given site during a specified time interval is the output of a probabilistic seismic hazard analysis. With utilizing the total probability theorem, this probability can be calculated from

$$P(Y > y) = \int P(Y > y \mid \underline{x}) f_{\underline{X}}(\underline{x}) d\underline{x} \qquad (\mathbf{3.1})$$

where Y is earthquake intensity as a random variable, y is the intensity level and $\underline{x}$ expresses the vector of random variables that affects the outcome of Y. The vector $\underline{x}$ usually contains the random variables M and R, which express magnitude of the earthquake and distance to the site of earthquake respectively. In the case of those variables are assumed to be independent, the equation 3.1 can be written as:

$$P(Y > y) = \iint P(Y > y \mid m, r) f_M(m) f_R(r) dm dr \qquad (\mathbf{3.2})$$

Firstly, attenuation is the energy with which an earthquake affects a location depends on the running distance. The attenuation in the signal of ground motion intensity plays an important role in the assessment of possible strong ground shaking. A seismic wave loses energy as it propagates through the earth (attenuation). This phenomenon is tied in to the dispersion of the seismic energy with the distance as shown in Figure 3.1.

It is necessary to select a ground motion parameter of severity of the earthquakes while estimating the seismic hazard at a site. Once having parameter, an attenuation relationship has to be set. This relationship defines an estimate of ground motion parameter in terms of magnitude and distance from site to epicenter of earthquake.

The peak ground acceleration or PGA is defined as the absolute maximum value of the representative temporary series of the ground acceleration. It is useful to define lateral forces and shear stresses in procedures that use equivalent static forces like the specified ones in the seismic codes. Hence, attenuation relationships defines the peak ground acceleration in terms of magnitude and epicentral or hypocentral distance. The intensity of ground motion at a site changes with distance from the zone of energy release (fault) during an earthquake.

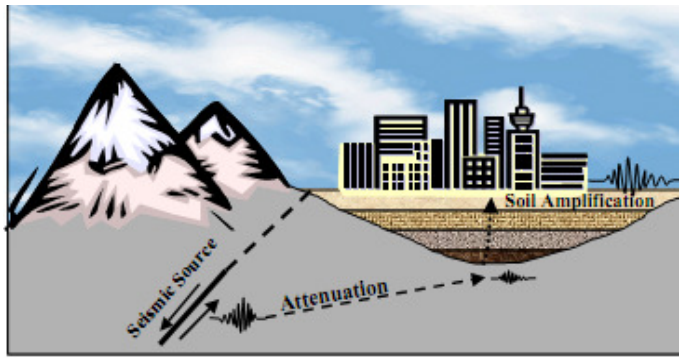

Figure 3.1: Representation of attenuation relationship (MIT, 2004)

3.3 Element Reliability

Seismic reliability of network elements gives reliability of a lifeline under an earthquake. Seismic capacity, which means ability to stand to earthquake loads, is used to determine the seismic reliability of an element. Since there are various random factors and uncertainties, that are utilized for calculating seismic capacity, seismic capacity should be defined as a probabilistic model. Uncertainties in seismic capacity of an element are based on uncertainties in the

material properties, dimensions, and models that are utilized for calculating the capacity.

3.3.1 Seismic Capacity Model

Seismic capacity model uses term capacity or strength of an element and the demand loading on the element by two random variables C and D respectively. The case demand exceeds capacity means failure, the failure probability becomes which is symbolized as P_f is a function like:

$$P_f = P(C \le D) = P(C - D \le 0) \quad \textbf{(3.3)}$$

This equation can be written as:

$$P_f = \int_{-\infty}^{\infty} \left(1 - F_D(c)\right) f_C(c)\, dc \quad \textbf{(3.4)}$$

Another important term for probability of survival state or the reliability of the element symbolized as P_S is just the opposite of P_f that is given below.

$$P_S = P(C > D) \quad \textbf{(3.5)}$$

$$P_S = R = 1 - P_f \quad \textbf{(3.6)}$$

In this respect, R, which means reliability, is a synonym for survival probability. With the same manner as P_f, reliability can be written as:

$$P_f = \iint_{w_f} f_{CD}(c,d)\, dcdd \quad \textbf{(3.7)}$$

$f_{CD}(c,d)$ is joint density function of C and D and w_f is the failure domain, which can be defined as $w_f = \{(c,d) \in R^2 \mid c - d < 0\}$.

In a simplified case of independence, distributions of f_c and f_d are still required to be known for estimating the probability of failure. However, if normality and independency of distributions are assumed both for demand and capacity with

means μ_C and μ_D, and standard deviations σ_C and σ_D, then probability of failure is expressed as below:

$$P_f = 1 - \Phi\left(\frac{\mu_c - \mu_D}{\sqrt{\sigma_C^2 - \sigma_D^2}}\right) \quad (\mathbf{3.8})$$

where $\Phi(\bullet)$ refers to the standardized normal distribution.

Reliability for a single site, point, which is given in equation 3.8 needs to be modified for spatially extended components. For this purpose, multi-site model which is based on separating the lifeline into smaller segments is developed by Selcuk (2000). This is done by the formula:

$$\prod_{j=1}^{n_j} P_{S_i} \leq P_{S_i} \leq min_{j=1,n_i} P_{S_i} \quad (\mathbf{3.9})$$

In terms of reliability, this equation is given as:

$$\prod_{j=1}^{n_j} R_{S_i} \leq R_{S_i} \leq min_{j=1,n_i} R_{S_i} \quad (\mathbf{3.10})$$

Since, reliability evaluation for elements is done under normality and independence assumptions, probabilities can be calculated by using two methods:

i. In first method, population mean, which is parameter for average value of peak ground acceleration values, is used. Therefore, z values are calculated as:

$$z = \frac{p(x_i - \mu_{pga})}{\sqrt{\sigma^2}} \quad (\mathbf{3.11})$$

Here, x_i values are the average of pga values of a grid, and μ_{pga} is the average of all the pga.

ii. In second method, sample estimate is used. If this method is chosen, the estimation of μ_{pga} is used. Therefore, for having a unbiased estimation for reliability the equation 3.17 is changed to:

$$z = \frac{p(x_i - \bar{x}_{pga})}{\sqrt{\sigma^2 / n}} \quad \textbf{(3.12)}$$

Here, $\bar{x}_{pga}$ is the average of pga values on a link.

3.4 Network Reliability

In a network every component is connected. The main issue in network reliability is to find operational path of nodes for analyzing. Since there would be more than one course of those nodes from terminal to end point, to achieve this aim, a network is divided into nodes and lines. Usually, the reliability of a system is probability that while operating under given environmental conditions, the system will do its job properly for certain time period (Kapur et al., 1977).

There are many methods for calculating the network reliability. These are

i. State Enumeration Method
ii. Factoring Method
iii. Series and Parallel Reduction Method
iv. Cut-Set Enumeration Method
v. Path Enumeration Method
vi. Network Reliability Model

Yoo and Deo (1988) compared all algorithms and commented that the best one was Dotson and Gobien (1979) algorithm which was a path enumeration algorithm. This algorithm gives reliabilities within an upper and lower bound in

a short time. This technique for finding out the terminal reliability or probabilistic networks is obtained. This technique utilizes set-the-oretic concepts division of space of graph realizations in a way that allows fast evaluation for the source to terminal probability (Dotson, et al, 1979).

This algorithm aims to find the probability of survival for each communication between specific pair of nodes in a network. However survival probability of each link has to be known. Presuming there is a network having n nodes and m links, two terminal nodes are identified as source and sink, which are denoted as s and t respectively. Therefore, an element must be either a node or link. Two adjacent nodes denoted by j and i. As a result this graph can be shown as an nxn matrix, which is namely adjacency matrix denoted as G=[g_{ij}](Dotson, et al, 1979).

$$g_{ij} = \begin{Bmatrix} 1, & \text{if there is a link} \\ 0, & \text{if there is no link} \end{Bmatrix} \quad \textbf{(3.13)}$$

In this study the modified Dotson Algorithm, which is developed by Yoo and Deo (1988), is used.

The flow of the algorithm is given below in Figure 3.2:

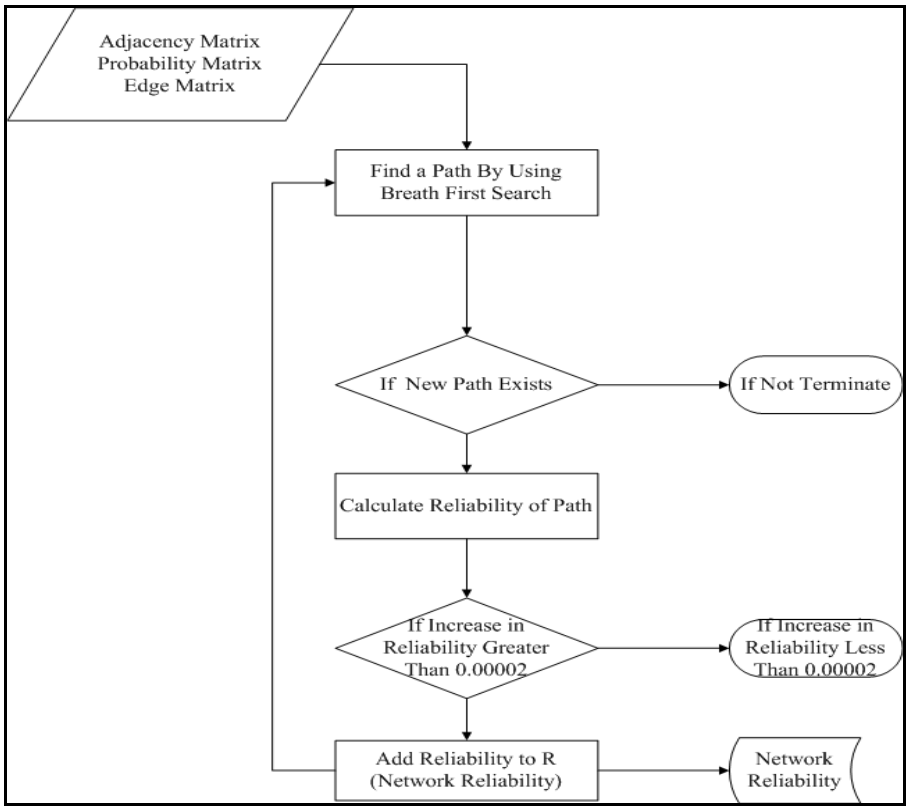

Figure 3.2: Flow Chart of Yoo and Deo Algortihm

Algorithm starts with reading the inputs which are adjacency, probability and edge matrices. Neighbourhood data between nodes and direction information are kept by adjaceny matrix. Element reliabilities are contained by probability matrix and link data are kept by edge matrix. After reading the inputs, a path is found by breath first search method and reliability of the path is calculated and added to R. Iteration continues until no new path can be found or the increase in R is smaller than 0.00002.

An illustrative example for the use of proposed algorithm to calculate network reliability is presented below. This example is taken from the earlier study of Yoo and Deo for comparison reasons. Given a network with four nodes and five edges (Figure 3.2), source and sink nodes are taken to be 1 and 4, respectively.

The figure below presents a network with four edges and 5 nodes. In this example source edge is 1 and sink edge is 4.

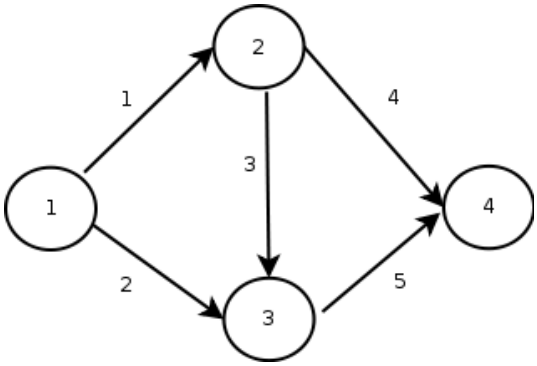

Figure 3.3: An Example to a Network

Based on the algorithm, the formulation of matrices needed as follows: Let G denotes the adjacency matrix and EDGE be the node matrix, and then the algorithm results in a vector of PROB yielding reliabilities.

$$G = \begin{pmatrix} 0\ 1\ 1\ 0 \\ 0\ 0\ 1\ 1 \\ 0\ 0\ 0\ 1 \\ 0\ 0\ 0\ 0 \end{pmatrix} \qquad EDGE = \begin{pmatrix} 1\ 3 \\ 1\ 2 \\ 2\ 3 \\ 2\ 4 \\ 3\ 4 \end{pmatrix} \qquad PROB = \begin{pmatrix} 0.9 \\ 0.9 \\ 0.9 \\ 0.9 \\ 0.9 \end{pmatrix}$$

Solution algorithm developed by Yoo and Deo is as follows:

INITILIAZATION: Set R = 0. Put event E_0 = [0 0 0 0 0] into the queue.

ITERATION 1: The E_0 = [0 0 0 0 0] deleted from the event queue. New E_0 is created which comes from the original network. Breath first shortest paths are found with source and sink edges 1 and 4, respectively. E_1 = [1 0 0 1 0] corresponds to first event and it is a success. Therefore 0.9 x 0.9 = 0.81 is added to R. After that, complement events of E_1 which are obtained and put into the queue. These are the

E_2 = [-1 0 0 0] and E_3 = [1 0 0 -1 0]

ITERATION 2: The next event deleted from the queue is E_2. The adjacency matrix of the network corresponding E_2 is given below:

A breath first search method finds the new shortest path source to sink. In this case this event contains edges 5 and 2 which are defined as E_4= [-1 1 0 0 1]. As it leads us to the success, its probability values become (0.1 x 0.9 x 0.9 = 0.081). This will increase the value of R to 0.891. Indeed complement events of E_4 which are sub-events of E_2 are given as

E_5 = [-1 -1 0 0 0] and E_6 = [-1 1 0 0 -1] are inserted into the queue.

This iteration continues until the queue is empty so that the algorithm terminates. The value of R as a result of these i iterations is 0.97119. (Yoo et al. 1988)

The figure below shows the event tree structure created by this algorithm. Each node in this tree means to a success in rectangular shape or a failure in oval shape. Algorithm generates four success and five failures. There are 2^5=32 elementary events (Yoo et al, 1988).

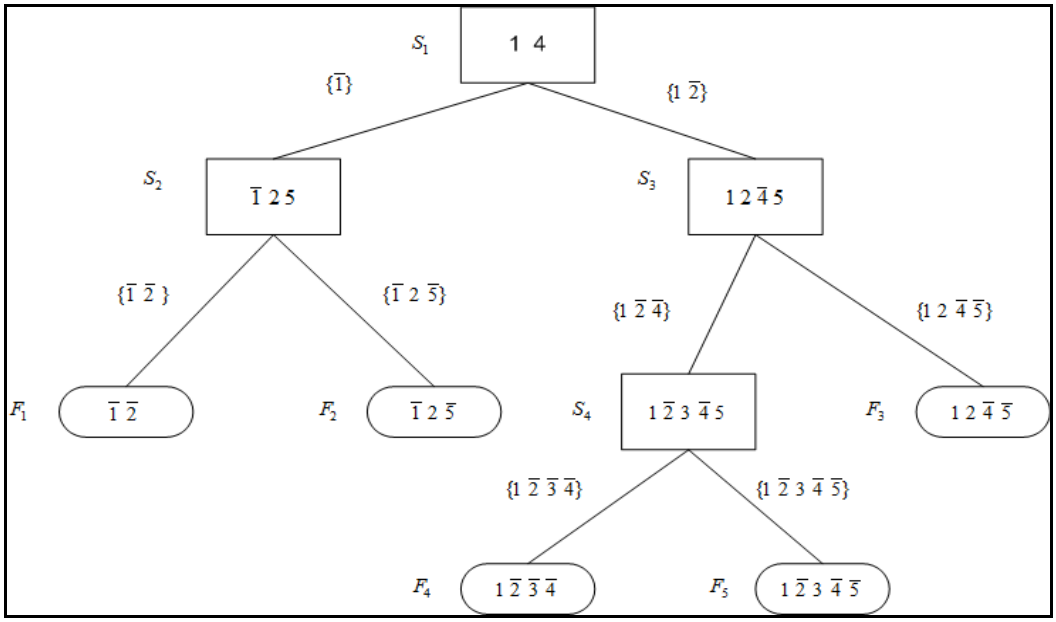

Figure 3.4: Event tree for Yoo and Deo algorithm (Yoo and Deo, 1988).

CHAPTER 4

METHODOLOGY

In order to build a GIS based reliability map of a lifeline under seismic hazard, a GIS based application was developed in Java programming language. For achieving this goal, developed application must take several inputs and after some conversions process, these inputs are used to plot the reliability map and to calculate the upper and lower bounds for network reliability of the system under seismic hazard. Figure 4.1, presents the overall flow chart of this process.

The application consists of a main.java class and several others that are invoked by the main.java class. After the main.java reads the inputs from the files, it passes them to util.java class which mainly does the gridding of the network, in other words it creates the line segment structure for reliability analysis. The util.java class takes these input data and grids the network layer in accordance with pixel size of the seismic hazard layer and also calculates the hazard values (pga values) of the grid elements. Then the main.java class passes the results from the util.java class to Gaussian.java class which calculates the survival probabilities for gridded network elements using standard normal distribution. After that main.java class passes the results from Gaussian.java class to netrel.java class, which calculates the network reliability. Finally reliability map is created by main.java.

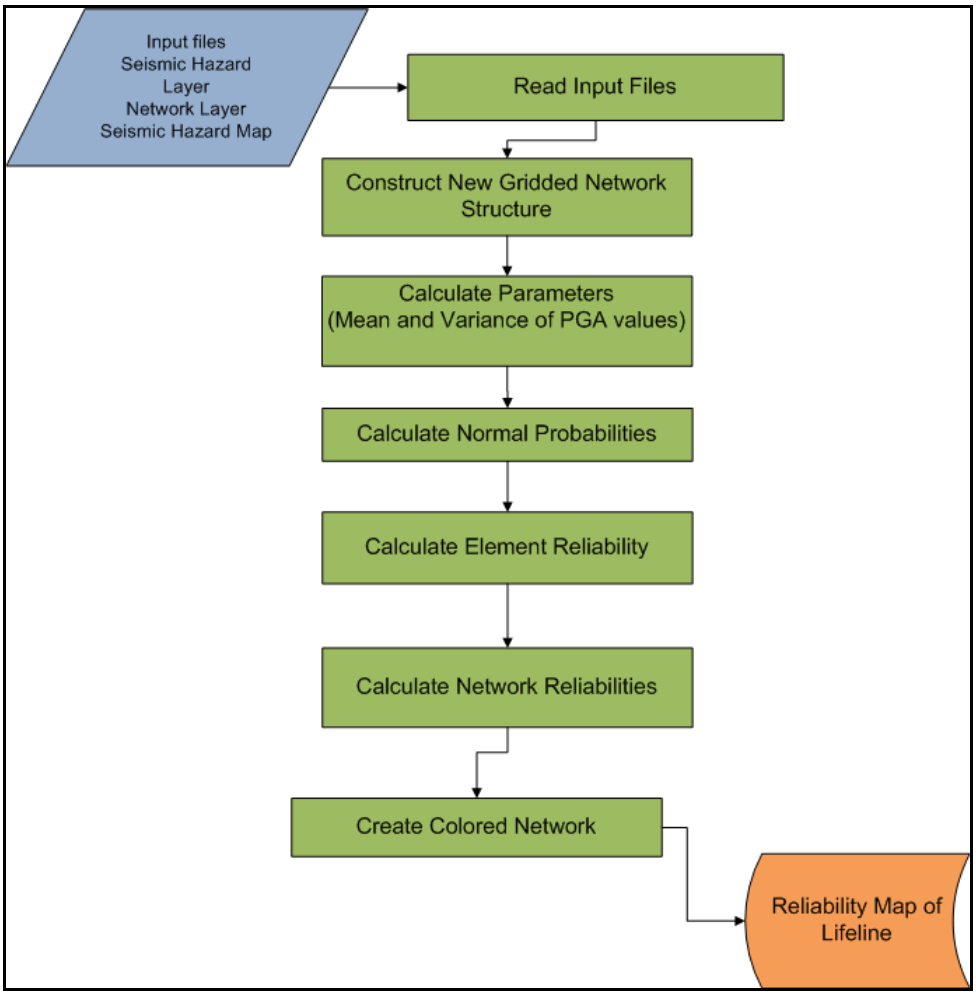

Figure 4.1: Flow chart of the developed application

The developed application has a sequential structure as mentioned in figure 4.1. In first step, input files are read by using the “FileDataStore” library network and seismic hazard layers can be represented. This library is also used for working with the “shp”, “dbf” and “prj” files as a group. Moreover, FeatureSource library is used for restoring the attributes from seismic hazard layer. Seismic hazard map layer is read by rasterFileChooser function (Appendix A).

Once the input files are read, application constructs a gridded network for reliability calculation. These operations are conducted by util.java. In this step, firstly nodes have to split into smaller line segments. In geoTools platform nodes are stored as MULTILINESTRING format, which allows creating smaller line

segments in LINESTRING format, indeed those two are the same in usage, but MULTILINESTRING format is a coalescence of line parts in LINESTRING format. The algorithm developed in this study for creating the line segments are given below:

1. Find coordinates of start and end points of link under consideration.
2. Using start and end coordinates, find the grids under interest.
3. Intercepts grids and node.
4. Create line parts of node depending on grid size.
5. Set average pga values of pixels to new multi-line strings.

The figure 4.2 illustrates the layers before the algorithm applied. Here the pga and network layers are not processed.

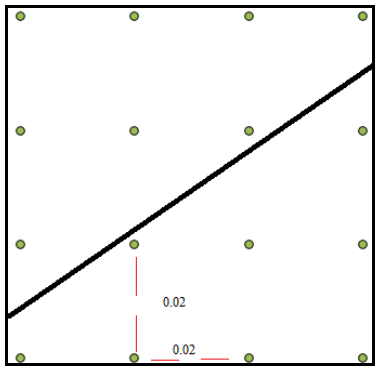

Figure 4.2: A node example over pga layer

Table 4.1 illustrates the attribute table of the network layer before the process.

Table 4.1: Attribute Table of Network Layer Before Process

Link No	FIRST_X	FIRST_Y	LAST_X	LAST_Y
1	X_1	Y_1	X_2	Y_2

After algorithm applied to those two layers a new vector layer is created, Figure 4.3 shows the new layer after application.

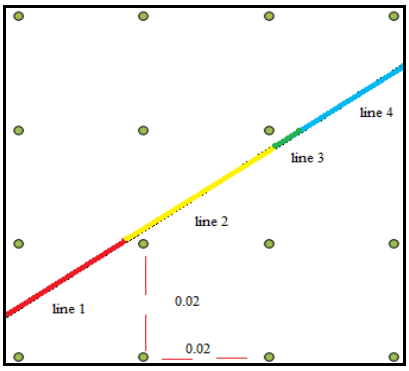

Figure 4.3: Line parts after the algorithm

Table 4.2 below illustrates the attribute table of the network layer after application of algorithm.

Table 4.2: Attribute Table of Network Layer Before Process

Link No	PGA	FIRST_X	FIRST_Y	LAST_X	LAST_Y	Probs
1	H_1	X_1	Y_1	X_2	Y_2	P_1
2	H_2	X_2	Y_2	X_3	Y_3	P_2
3	H_3	X_3	Y_3	X_4	Y_4	P_3
4	H_4	X_4	Y_4	X_5	Y_5	P_4

After creating line segments as a new layer, reliabilities have to be calculated for those line parts. For doing this, a pga value is assigned to every line segment. The pga values are taken to be average of pga values of the grid elements. This step requires the calculation of parameters the mean and the variance are calculated.

As mentioned the processes above are done by Util.java class. While performing these operations util.java class uses “getintersectionPoints” and

“getEndPointPositions” functions mainly and generate a new vector layer of line-segments (Appendix A). In first phase, the end points of the network element are obtained by “getEndPointPositions”. Once having the end points of the elements, the grids under interest are found by “getIntersectionLinearRings” function. Finally intersection point of network and seismic hazard layers are obtained by “getintersectionPoints” and new line segments are created by using those intersection points. Here, functions such as “Main.selectFeatures” are used for controls of the developed application. “Main.selectFeatures” functions is for choosing source and sink points by mouse, “Main.displaylayers” function is used for displaying the new network layer, the “Main.generateTempShapeFile” is used for generating a temporary file for analysis. The boxes such as “invoked by” or “Instantiated by” are used to display the running sequence of functions in developed modules. Detailed structure of the util.java class can be seen in figure 4.4. Appendix B represents the output of this class in Table 6.1.

Once the gridded network structure is constructed by util.java, this new network structure is sent through main.java and z values are calculated based on pga values by using the equations (3.11) or (3.12) and Gaussian.class is imported by main.java. Those z values are processed by gaussian.java class, where the normal probabilities are calculated which can be seen in Table 6.1 (Appendix B). Gaussian class has two main functions, those are “phi” and “Phi” here “phi” used for normal probabilities of standard Gaussian function and “Phi” is for calculating the cumulative probabilities of standard normal values by using Taylor series expansion. Gaussian.class uses “Phi” for obtaining the tabulated normal probabilities. Those probabilities are set as an attribute to generated temporary shape file for analysis. The detailed structure of the Gaussian.java class can be seen in figure 4.5.

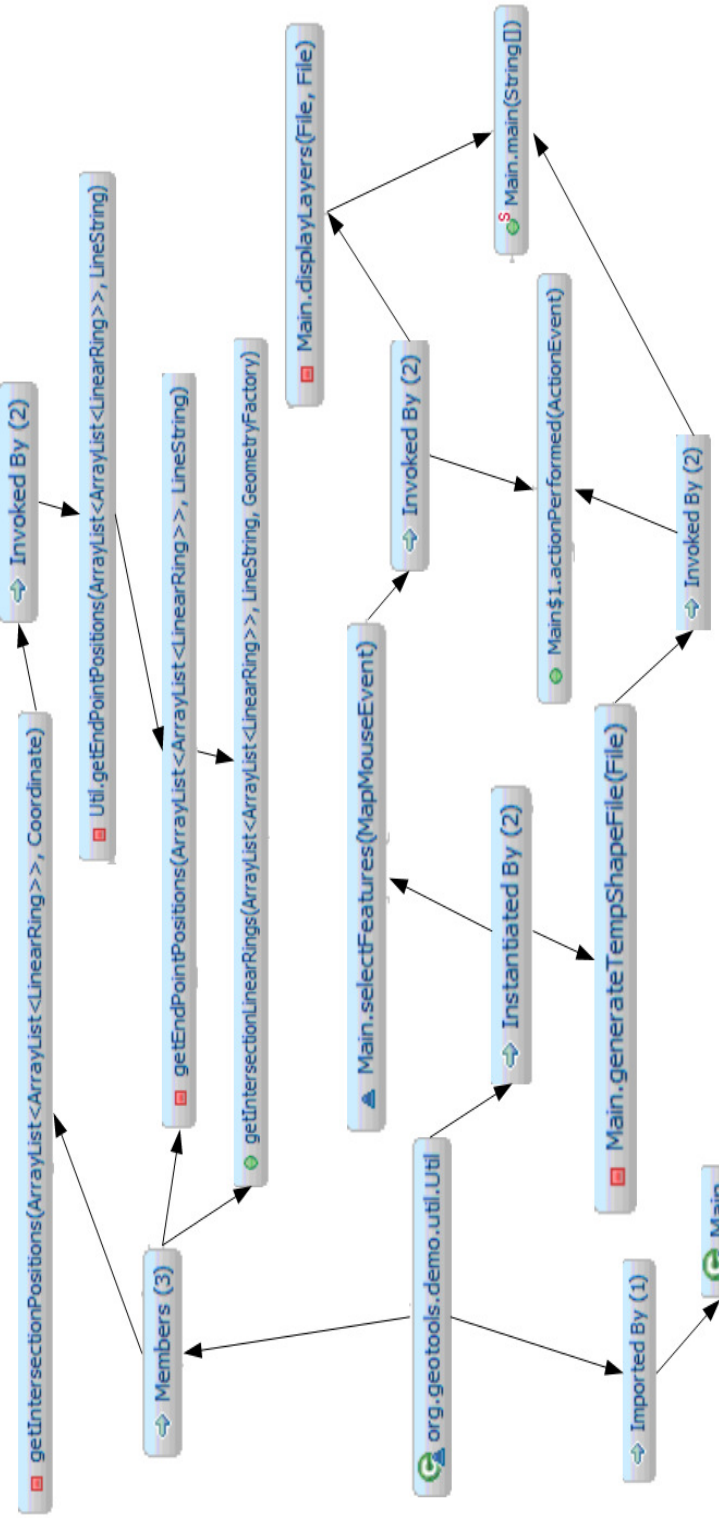

Figure 4.4: Detailed structure of util.java class

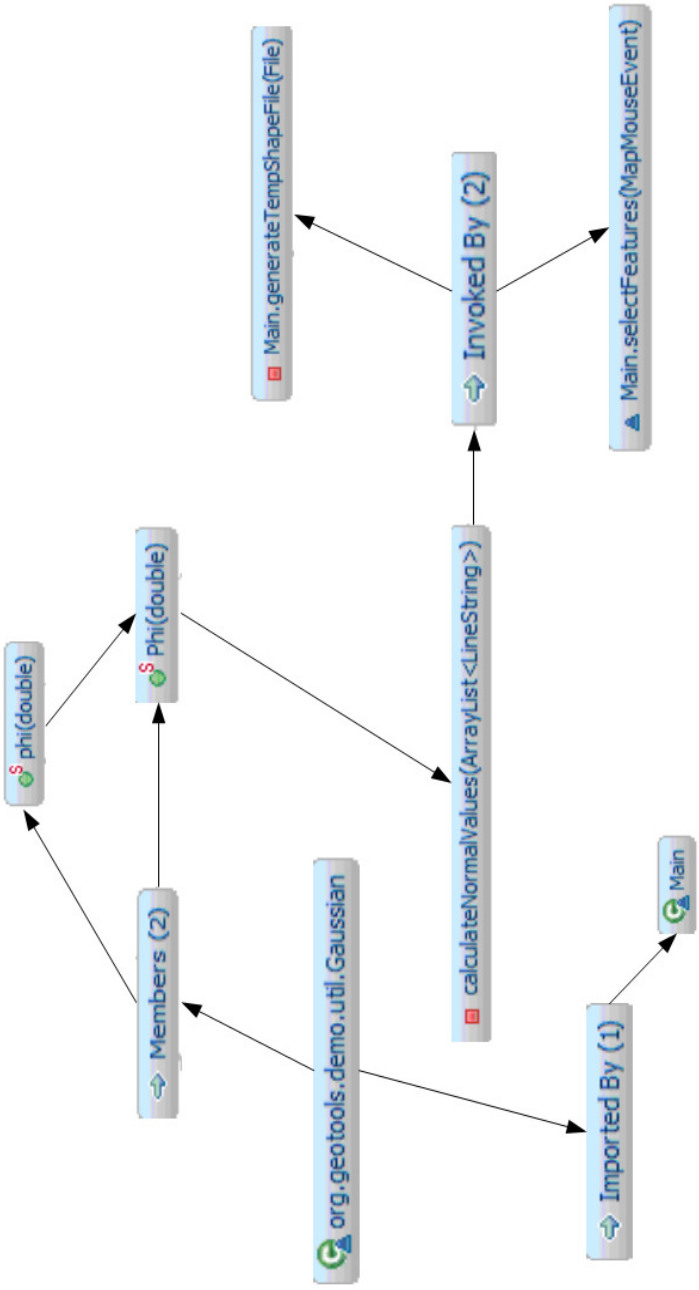

Figure 4.5: Structure of gaussian.java

Having the line segments survival probabilities, the upper and lower bounds of element reliabilities are calculated in main.java by using equations (3.9) and (3.10). Those element reliability values are set to temporary network layer as an attribute, by using the AttributeType and AttributeDescriptor libraries. After having element reliabilities those values are passed through netrel.java class.

At this stage, element reliability values are used in netrel.java class for calculating network reliability. Other required parameters for network reliability calculation such as adjacency matrix and edge matrix are calculated and passed through netrel.java class by main.java. In main.java, calculateAdjacencyMatrix reads the node list and creates an adjacency matrix, and using hashmap libraries of java, edge matrix is created by egdeIndex function. Having reliability, adjacency, and edge matrices netrel.java class calculated the network reliability. This class includes two main parameters and four functions. Those parameters are “qsize” which is used for queue size of the iteration and set to 2500 and “psize” is used for number of paths which is set to 500. The functions in netrel.java are “bfs”, “findInterval”, “probab”, and “comple”. bfs function is used for making a breath first search of paths and after first path is found by this function, probab function calculates the reliability of that path. Then comple function removes the used path from queue for next iterations, finally findInterval function is calculates the reliability interval of the network under interest. The iteration ends in two ways, when there is no new path or the increase in reliability values less than 0.00002. The detailed structure of netrel.java class can be seen in Figure 4.6.

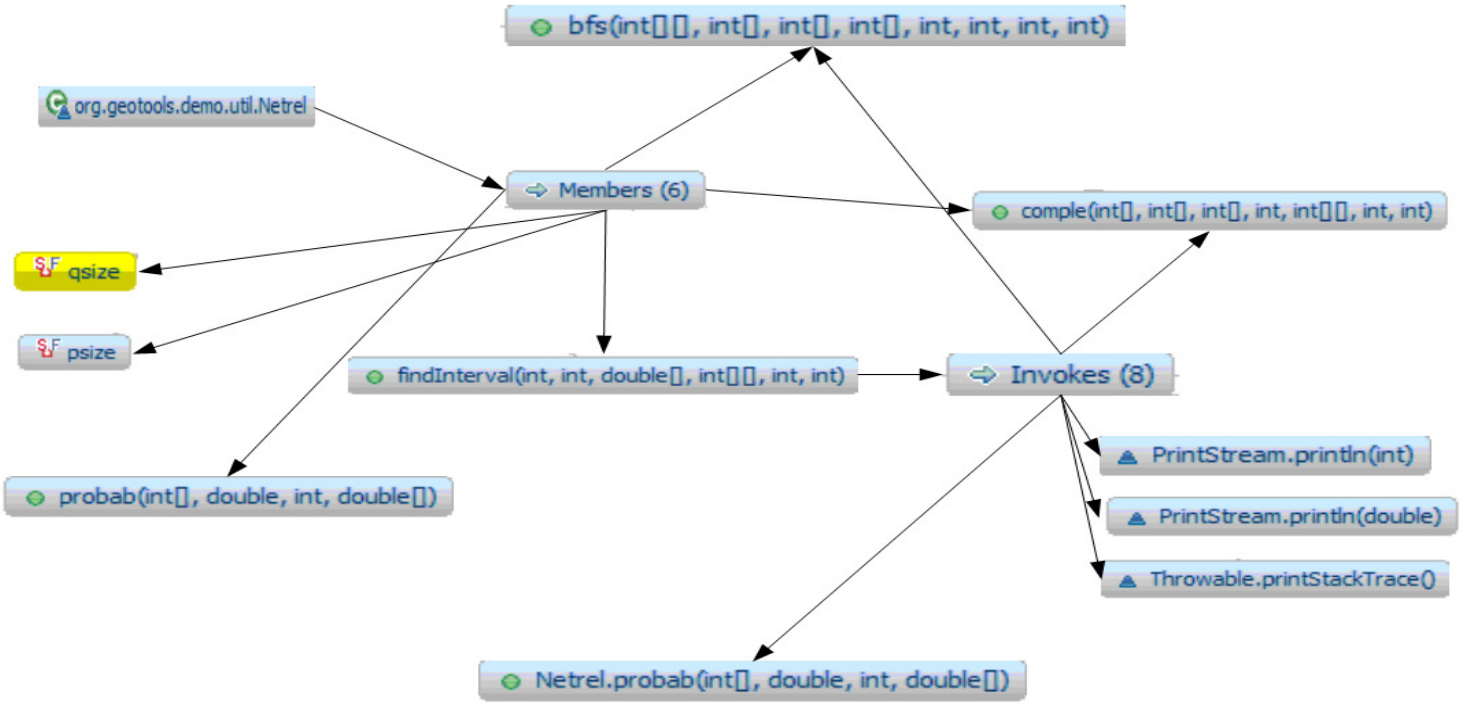

Figure 4.6: Detailed Sturcture of netrel.java class.

4.1 Implementation of Software

In order to give the overall reliability of the lifeline network, the tool combines the seismic hazard and lifeline network layer initially by using util.class. For each element of the lifeline network, element reliabilities are evaluated afterwards. Once having those reliabilities, possible routes between source and sink are found and adjacency matrix is created which is used for network reliability assessment by netrel.java class later on.

There are many methods for generating a seismic hazard map. However, in this study formerly calculated seismic hazard values are used as peak ground acceleration (pga) values which are given to the application as seismic hazard layer. These values are taken from the study of Yilmaz (2008). In order to give these values as an input to software, a point shape file is created from the given pga values. This shape file keeps coordinates of the point in latitude, longitude format with WGS84 datum and also keeps the average pga values of these points. Figure 4.7 shows a magnified part of shape file of pga values.

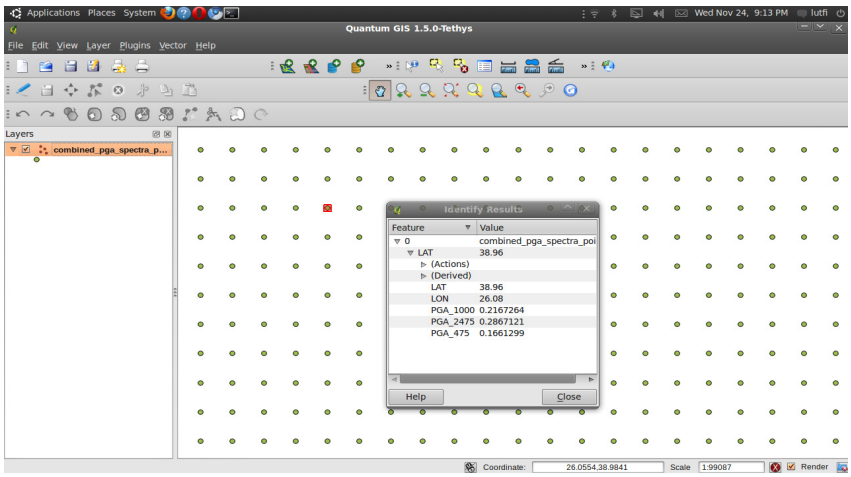

Figure 4.7: Magnified part of pga layer

This seismic hazard layer has grid structure with dimensions of 0.02 degree, which are approximately equal to 250 meters resolution. Figure 4.8 represents a

seismic hazard map generated in accordance with 475 years average pga values by using Mapinfo and this raster layer is just used for the visualization of the seismic hazard, which can be replaced with any georeferenced raster file.

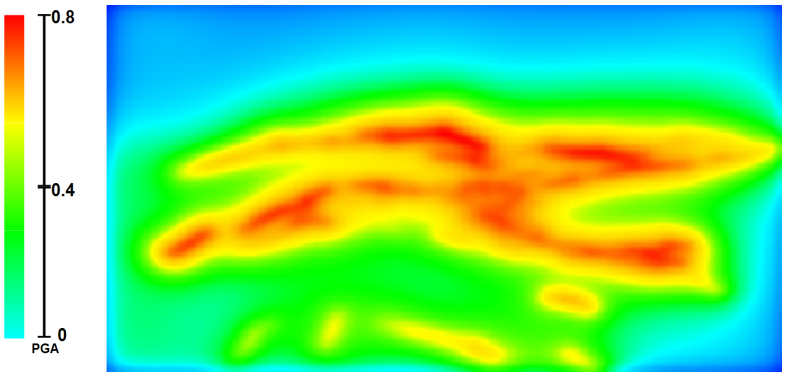

Figure 4.8: Seismic hazard map of Bursa based on 475 years average values

The network layer representing water pipeline system of Bursa is taken from study of Sevuk and Altinbilek (1977). Figure 4.9 shows the considered network. The water pipeline includes a pump station and two reservoirs to be chosen as the source points while a certain point in network chosen as sink point. In such networks, the existence of more than one source and several pipelines results in multiple routes between source and sink points, thus the developed application gives an overall reliability of sum of that routes by using the algorithm given in chapter 4.

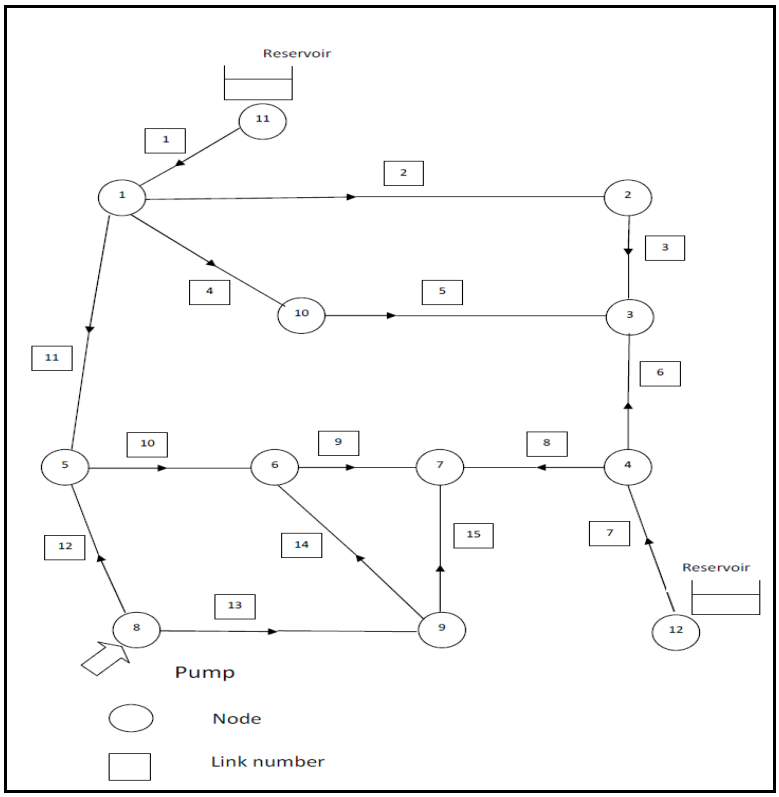

Figure 4.9: A schematic drawing of the water distribution system (Sevuk and Altinbilek, 1977)

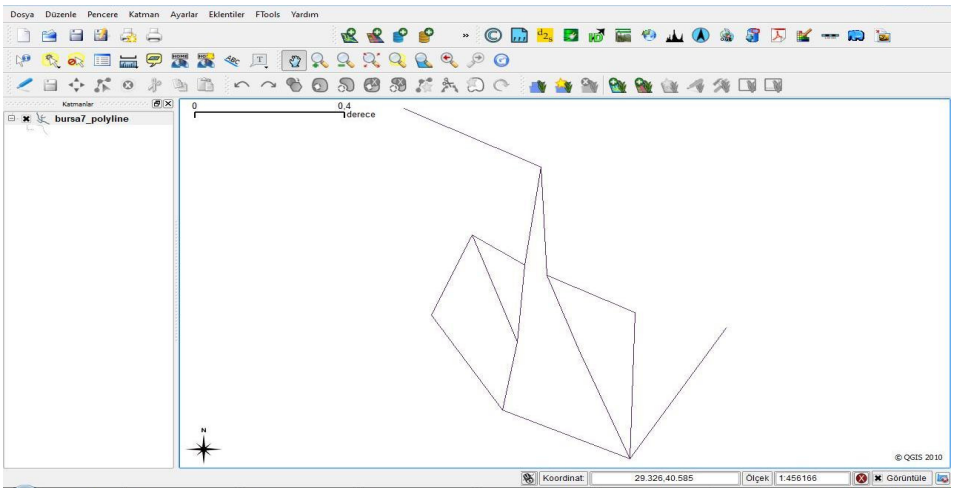

Figure 4.10: Network Layer

Figure 4.10 illustrates the network layer in original coordinates generated by Sevuk and Altinbilek, 1977.

In figure 4.11, the proposed lifeline network is placed on a georeferenced satellite image of the part of Bursa under interest.

Figure 4.11: Lifeline layer over a satellite image

The developed tool takes seismic hazard and lifeline network layers in shape file format. Once the application is executed it starts with file selector window, which is for seismic hazard layer, shown in Figure 4.12;

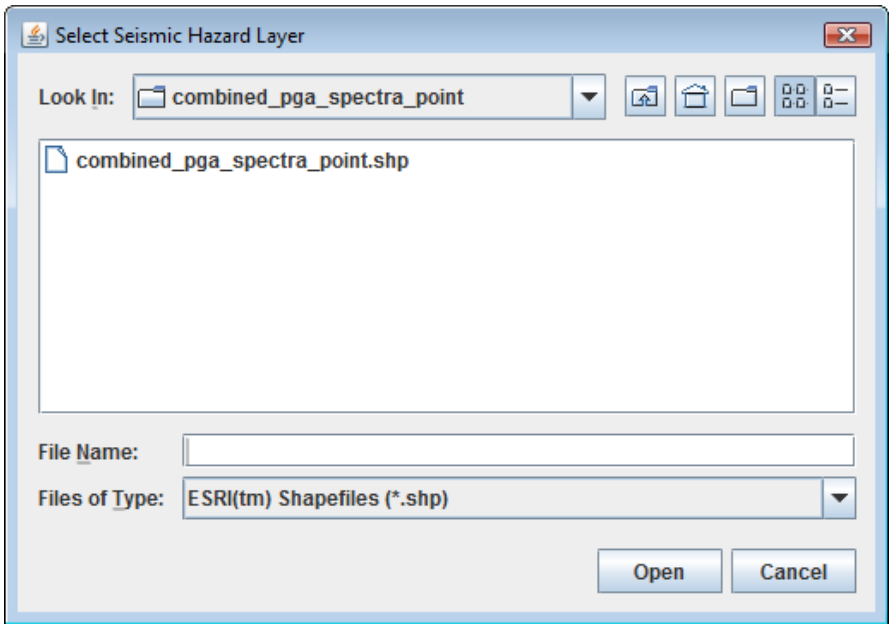

Figure 4.12: File selector window for seismic hazard layer.

After selecting the seismic hazard layer, another file chooser window appears for network layer which can be seen in Figure 4.13.

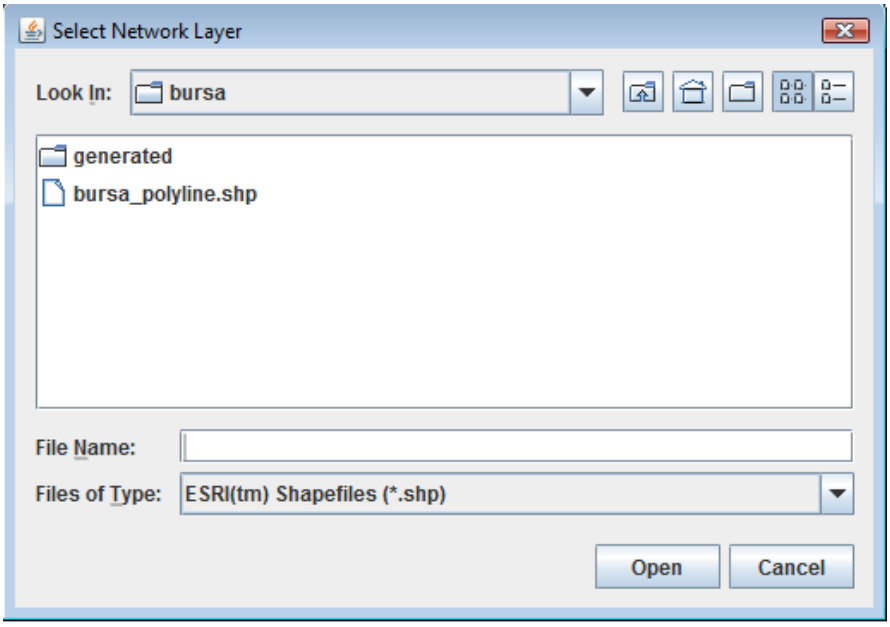

Figure 4.13: File selector window for network layer.

Finally, another file chooser window appears for raster layer which can be seen in Figure 4.14. This layer is optional and by choosing “cancel”, the user can continue without this layer.

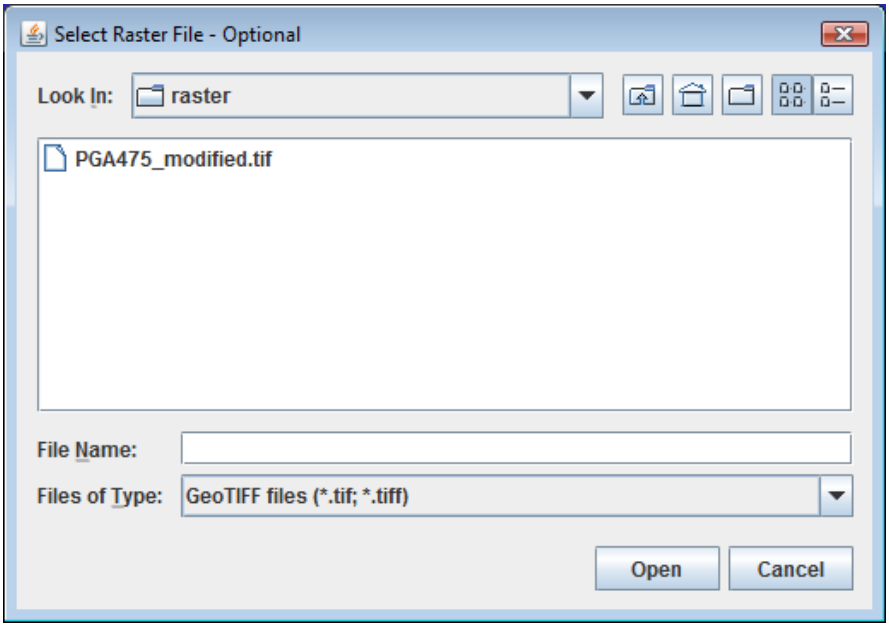

Figure 4.14: File selector window for raster layer.

Once the input files are chosen, another file chooser for the output file appears which allows user to save output file in a certain directory. Figure 4.15 below illustrates this file chooser window for output file.

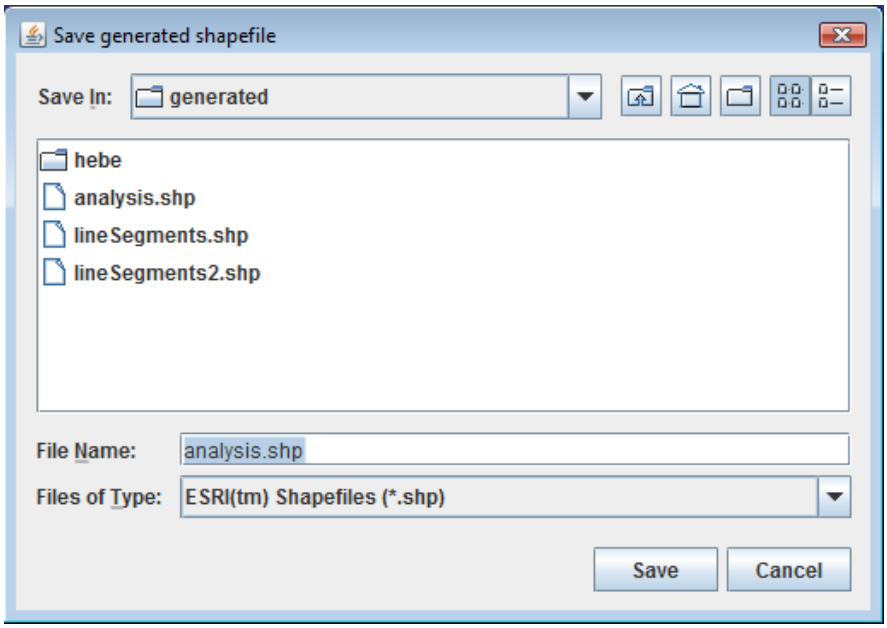

Figure 4.15: File selector window for output file.

After input and output files specified, the main window of the develop application is opened. Figure 4.16 illustrates the main window of the application.

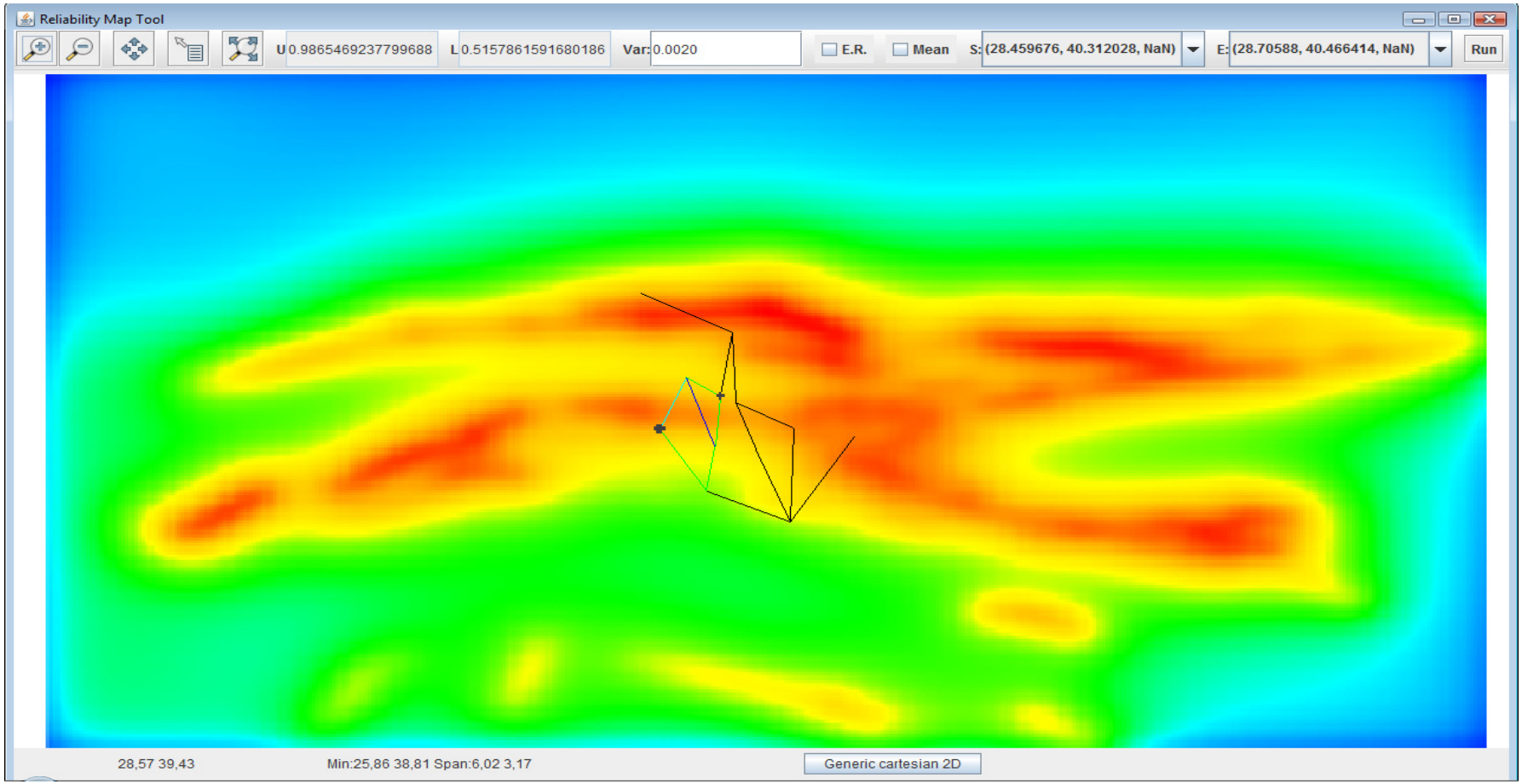

Figure 4.16: Main window after application started.

On main window two points from the list are selected as source and sink, and clicking the “Run” button application is executed. On the main frame of the program, user can choose the “Mean” option, “E.R.” (Element Reliability) and variance option which are explained in detail in Chapter 3. The Figure 4.17 below indicate results when the mean option is selected on toolbar.

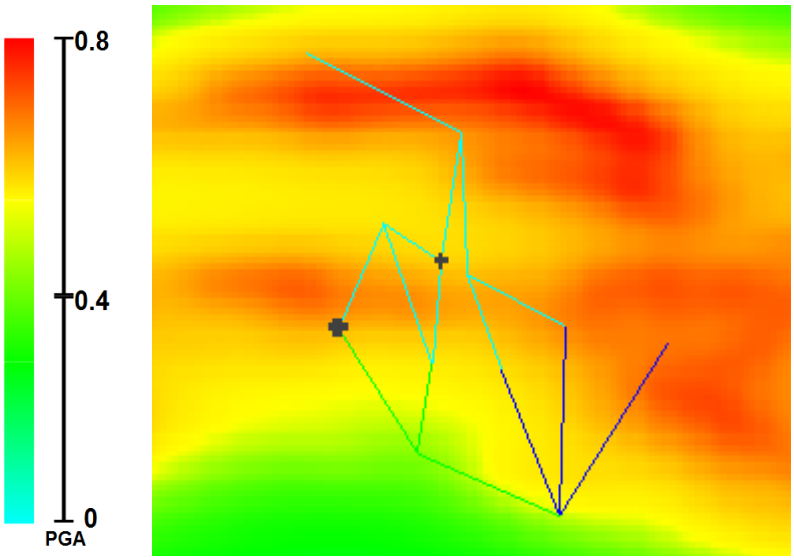

Figure 4.17: Reliabilities for elements by selecting“Mean” option

Based on the component reliability coded in the program Table 4.3 and Table 4.4 present the values of survival probability for with and without selecting mean option, respectively.

Table 4.3: Element reliabilities by selected “Mean” option

Link Number	Reliability
1	0,558384137
2	0,686912141
3	0,686912141
4	0,686912141
5	0,971471511
6	0,918452944
7	0,919727409
8	0,994207004
9	0,551945781
10	0,992773483
11	0,939943514
12	0,94964042
13	0,525365759
14	0,963004678
15	0,919727409

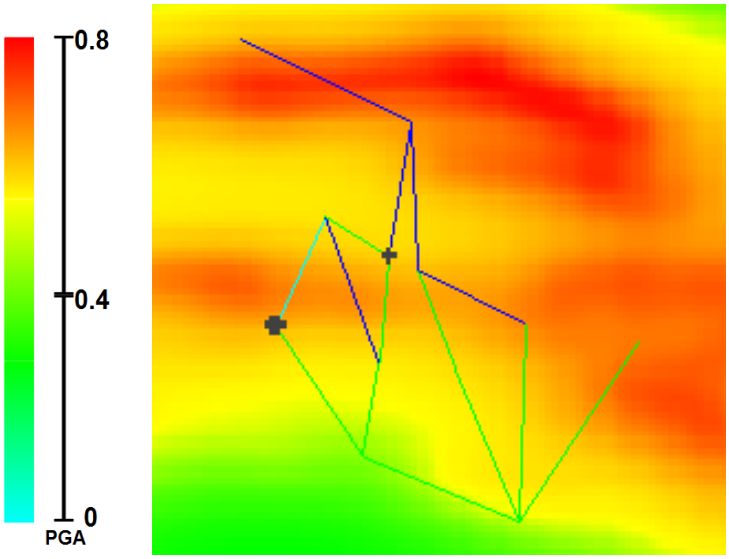

Figure 4.18: Reliabilities for elements without “Mean” option

Table 4.4: Element reliabilities without “Mean” option

Link Number	Reliability
1	0,89862064
2	0,69862064
3	0,76862064
4	0,99862064
5	0,99862064
6	0,99862064
7	0,99862064
8	0,99862064
9	0,99862064
10	0,99862064
11	0,99862064
12	0,99862064
13	0,79862064
14	0,99862064
15	0,99862064

“E.R” option is for presenting the element wise reliabilities, instead of network reliability. The output with “E.R” option is shown in figure 4.19.

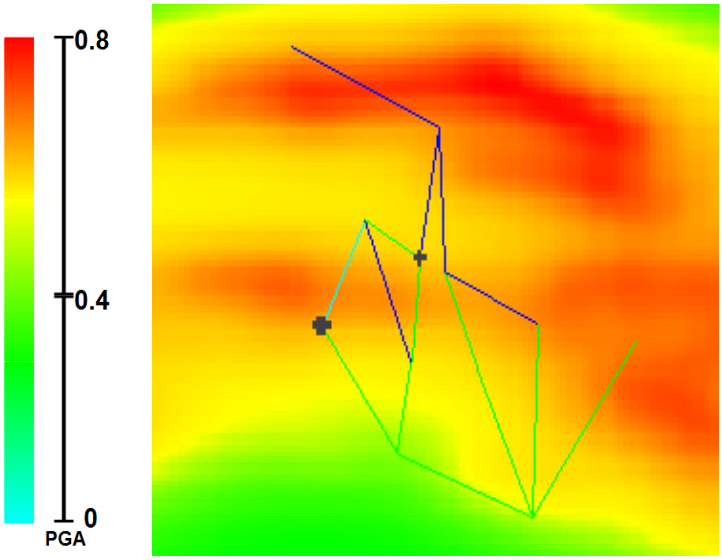

Figure 4.19: Reliability map for elements

The main aim of the developed application is to calculate the network reliability between two given points. Network reliability is calculated using the element reliabilities, as mentioned in Chapter 3. In this study, lower bound of the element reliabilities which are given in Table 4.5 are used.

Table 4.5: Lower Bounds for Element Reliability

Link Number	Lower Bound
1	0,5548
2	0,5420
3	0,5909
4	0,5022
5	0,5256
6	0,8880
7	0,6098
8	0,6974
9	0,5299
10	0,6951
11	0,5987
12	0,6588
13	0,5164
14	0,7822
15	0,5255

As the element reliabilities are available the data is passed to netrel.java class and network reliability analyses are conducted. The algorithm of Yoo and Deo (1988) algorithm is implemented in netrel.java class (Appendix A). Within this class “bfs”, “probab”, and “comple” functions are used. A breath first search is done finding the alternative path from source to sink. After that, those paths are analyzed by comple function one by one and passed through the “probab” function. Network reliabilities are calculated by “probab” function.

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{pmatrix}$$

Figure 4.20: Adjaceny Matrix Created by netrel.java

By using the AttributeType and AttributeDescriptor libraries, the reliability values are set to network layer as an attribute. Resulting attribute table of the network, given in table 4.6. Here, “reliability” field is the element reliability, that is used for network reliability analysis and “upper” and “lower” fields refer to lower and upper bounds on the network respectively.

Table 4.6: Example of attribute table of Resulting Network Layer

Reliability	Lower	Upper
0,554823899	0,515786159	0,986546924
0,541959357	0,515786159	0,986546924
0,59088161	0,515786159	0,986546924
0,502193882	0,515786159	0,986546924
0,525604927	0,515786159	0,986546924
0,888024551	0,515786159	0,986546924
0,609808289	0,515786159	0,986546924
0,697357123	0,515786159	0,986546924
0,529895028	0,515786159	0,986546924
0,695090484	0,515786159	0,986546924
0,598713428	0,515786159	0,986546924
0,658800089	0,515786159	0,986546924
0,516377549	0,515786159	0,986546924
0,78217528	0,515786159	0,986546924
0,52549179	0,515786159	0,986546924

4.2 Validation of Software

In order to check whether the developed software functions properly, as a case study, an earlier research of Selcuk (2000) over Bursa water pipeline was used. The original study of Selcuk uses the pipeline plans from the study of Sevuk and Altinbilek (1977). In Figure 4.9 the network is given.

Results of this case study are obtained, by using different options of application. As source points reservoirs and pump station are used. The same point with Selcuk's study is used as sink, which is node 7. Since application can study one source and sink points at a time, the application was executed for three sources separately.

In the first run, the pump station 8 was selected as the source and point 7 as the sink, the result of which can be seen in Figure 4.21. As can be seen from Figure 4.21, the nodes between these two points are coloured according to different values of reliabilities.

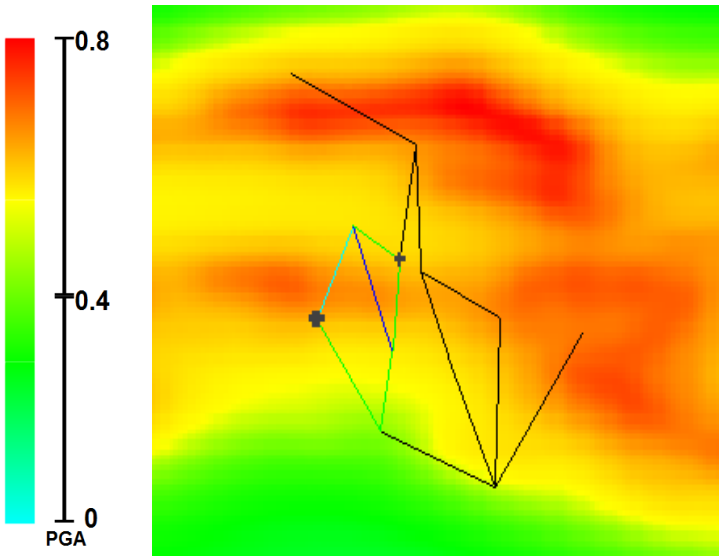

Figure 4.21: Output of application for source point as 8 to sink point as 7.

In former study of Selcuk (2000), the reliability interval of this network had the results 0.50397654 as lower bound and 0.92255221 as upper bound, while in this study the upper and lower reliability bounds are found as 0.51578615, 0.98654692, respectively.

In the next run, the reservoir 11 was selected as the source and point 7 as the sink. The result can be seen in figure 4.22. As can be seen from Figure 4.22, the nodes between these two points are coloured according to reliabilities.

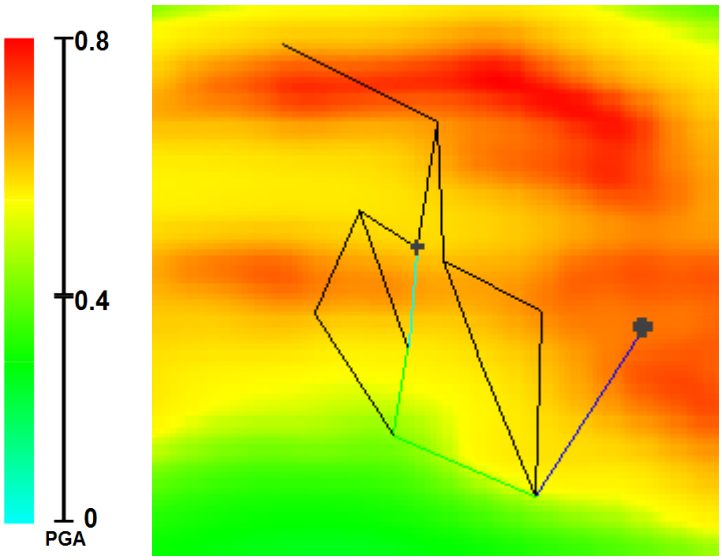

Figure 4.22: Output of application for source point as 11 to sink point as 7.

In this run the results are 0.242322972 as lower bound and 0.97885037 as upper bound, while in former study of Selcuk (2000) the reliability interval of this network had 0.05824558 as lower bound and 0.531142095 as upper bound.

In the last run, the reservoir (12) was selected as the source and point 7 as the sink, the result of which is presented in Figure 4.23. As can be seen from Figure 4.23 the nodes between these two points are coloured according to reliabilities.

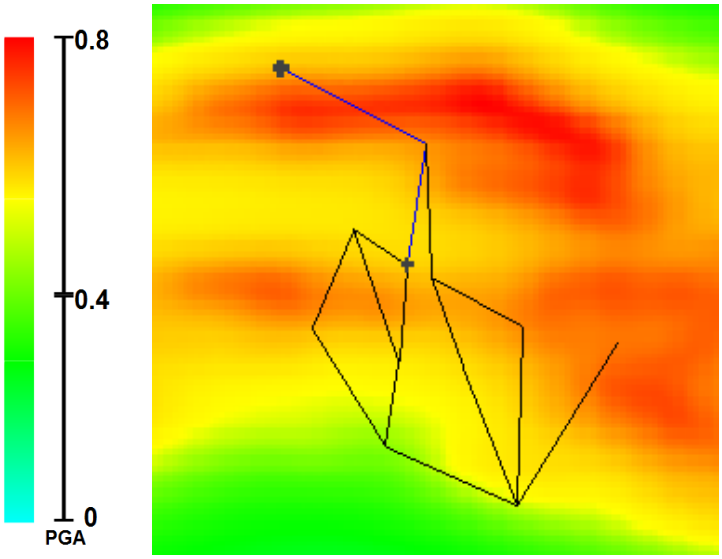

Figure 4.23: Output of application for source point as 12 to sink point as 7.

In this final run the results are 0.27135216 as lower bound and 0.75497461 as upper bound. In former study of Selcuk (2000) the reliability interval of this network was 0.23964787 as lower bound and 0.72729676 as upper bound.

4.3 Discussion

The developed application was tested with the earlier study of Selcuk (2000) and the results were satisfactorily close to this study. Differences between the results arouse from two reasons;

i. The seismic hazard values for two studies are different. In the former study of Selcuk (2000), the seismic hazard values were calculated by the developed software, while in this study the seismic hazard values for Bursa are taken as input to application. The seismic hazard values used in

this study are taken from a recent study which was done by Yilmaz (2008). Therefore, seismic hazard values are up to date and seismic resolution is more precise.

ii. Since the computation power of computers today are very much higher than those at the time of study of Selcuk (2000), element reliabilities are calculated for each network element which are divided into smaller line segments than former study of Selcuk.

As these conditions are considered, the results are taken to be satisfactorily close the former study done by Selcuk.

Since the reliability calculation algorithm used in this study does not consider the material or inner structure of the lifeline system and soil properties, it can be used for networks such as water, waste water, and natural gas pipeline systems. As explained in the methodology the solution algorithm is limited to use unidirectional systems. Since lifelines are unidirectional networks, such an algorithm is applicable to most of the lifeline system. The application developed in this study is essential for providing survival probability of lifeline structures under earthquake loads. That is especially important for lifelines like natural gas pipeline systems, which can cause cascading disaster effects after major earthquakes.

Due to the limited scope and time of the research the application was developed as a standalone GIS tool. However it is suitable to convert a plug-in for Udig. Furthermore, use of java which is a platform independent language, gives extra flexibility about the architecture and operating system of the environment, in which the application can run.

This software module was tested on a hypothetical network, previously used by Sevuk and Altinbilek (1977). Therefore the network structure does not fit the city plan correctly and this application should be applied to a real life network.

As performed in this study the reliability analysis of lifelines is important to estimate the results of a possible earthquake. However, for such systems, it is important to prevent failures under earthquake loads rather than to estimate the risk. Consequently, the output of the developed GIS application can be used for strengthening structure of lifeline networks.

CHAPTER 5

CONCLUSIONS and RECOMMENDATIONS

This thesis is about developing a GIS tool that can generate reliability map for lifelines under seismic hazard. To achieve this goal, an application that combines GIS, statistics, and facility management is developed. In order to combine these three, point shape file seismic hazard layer, network layer and seismic map layer are taken as input to application while a reliability map is the output of application. Using these input layers, the application first overlaps the network and seismic hazard layers. Then, seismic hazard layer is considered as square mash grids. Depending on the coordinates of those points and network layer is divided into line segments such that each line segment lies in a mash grid. After these proeesses, pga values, which are the average of that mash grid, are calculated for every line segment and given as an attribute to that line segments which are in line string format. Probabilities for those line segments are evaluated and used for the assessment of reliabilities of network elements. Knowing the reliabilities for elements of network, reliability is calculated between sources and sink points.

This application was developed on geoTools, which is GIS solutions for Java language. Moreover, the GIS application called Udig is developed using Java language on Eclipse development environment. Because geoTools is coded in Java, this application can be used as a plug in for Udig GIS software with some modifications. Document of Udig provides information about those modifications that have to be applied. As explained in these documents, in order

to add developed application to Udig as a plugin, relevant classes of Udig are downloaded and added as a project. Then developed classes are injected as classes and recompiled under downloaded project.

Developed software was tested with benchmark tests in previous papers and former study of Selcuk (2000) as a case study for a part of Bursa water pipeline system. The result are slightly different then Selcuk's former study. Those differences aroused from the difference of seismic hazard values and precision of the calculation. Therefore, the results were satisfactorily in accordance with former study.

As explained above, the developed GIS application can be used for different purposes, and can be interoperated with different platforms and other applications. In order to make such uses easier and more efficient, for further researches a database connection functionality can be added to this application. This database connection allows users make queries over the output results.

The developed tool was tested by using the former study of Selcuk, which considers a hypothetical network. In order to, further check the reliability of the developed GIS application several case studies and comparisons can be done using real lifeline networks which were studied before.

Lifeline networks are constructed using different materials, such as concrete, plastic, etc. which affect their strength against the earthquake loads. Also soil structure is another important parameter for resistance against earthquake loads. Therefore, another research subject may be to include material type and soil properties to the model used in this study, for more precise calculations of reliability under seismic hazard.

Resulting application can be used any GIS user, visualizing the results ease the understanding and managing the lifelines under earthquake loads. The outcomes of this application would be used to improve purposes of network components against seismic hazard which have low survival probability.

Therefore, after major earthquakes other disaster like fires depending of lifeline networks can be prevented. Moreover, this application can be used to design new lifeline networks, designers can build the networks on different routes or more resistant materials to seismic hazard can be used.

REFERENCES

Aydinoglu, M. N., Erdik, M., 1995. January 1995 Hyogo-ken Nanbu (Kobe) earthquake reconnaissance and assessment report. Bogazici Universitesi, Kandilli Rasathanesi ve Deprem Arastirma Enstitusu, Bogazici University publications, 1995, vi,118p.

Bendimerad. F., 2001. Loss estimation: a powerful tool for risk assessment and mitigation. Soil Dynamics and Earthquake Engineering 21 467-472.

Chen, T.-H., Cherng, J.-T., 1997. Design of a TLM application program based on an AM/FM/GIS system. Power Industry Computer Applications. 20th International Conference Book 346 - 351.

Duzgun, H.S.B., Yücemen, M.S., 2007. Kentsel alanlarda bütünleşik deprem riski modeli: Eskişehir örneği. Afet Sempozyumu Bildiriler Kitabı, pp. 201-211.

Ellingwood, B.R., 2001. Earthquake risk assessment of building structures Reliability. Engineering and System Safety 74, 251-262.

Ertugay, K., Duzgun, S., 2006. Integrating physical accessibility of emergency establishments into earthquake risk assessment. ECI Conference on Geohazards, Lillehammer, Norway, Paper 45.

Faccioli, E., 2006. Seismic hazard assessment for derivation of earthquake scenarios in Risk-UE. Bull Earthquake Eng 4, 341–364.

Kemec, S., Duzgun, S., 2006. Use of 3D visualization in natural disaster risk assessment for urban areas. Innovations in 3D Geoinformation Systems Lecture Notes in Geoinformation and Cartography, Part 8, Part 2, 557-566.

Kemec, S., Zlatanova S., and Duzgun, S., 2009. Selecting 3D urban visualisation models for disaster management: a rule-based approach. Proceedings of TIEMS Annual Conference, Istanbul, Turkey.

Lembo, A. J., O'Rourke Jr, T. D., and Bonneau, A. L., 2009. Advances in GIS for Lifeline. Visualization and Management. American Society of Civil Engineers, Conf. Proc. 357, 51.

O'Rourke, T. D., Toprak, S., and J., Sang-Soo, 1999. GIS Characterization of the Los Angeles Water Supply, Earthquake Effects, and Pipeline Damage Research Progress and Accomplishments 1997-1999. Multidisciplinary Center for Earthquake Engineering Research, University at Buffalo, July, pp. 45-54.

Padgett, J. E., and DesRoches, R., 2007. Bridge Functionality Relationships for Improved Seismic Risk Assessment of Transportation Networks, Earthquake Spectra Volume 23, Issue 1, pp. 115-130

Selcuk, A.S., Yücemen, M.S., 1999. Reliability of lifeline networks under seismic hazard. Reliability Engineering and System Safety 65, 213–227.

Selcuk, A.S., Yücemen, M.S., 2000. Reliability of lifeline networks with multiple sources under seismic hazard. Natural Hazards 21 1–18.

Sevuk, S., Altinbilek, D., 1977. Su Dagitim Sebekeleri Projelendirme ve Bilgisayarla Cozum Esaslari, Orta Dogu Teknik Universitesi Muhendislik Fakultesi Yayinlari No 56.

Toprak, S., O'Rourke, T. D.,Tutuncu, I., 1999. GIS characterization of spatially distributed lifeline damage. Tech Council Lifeline Earthquake Eng Monogr , no. 16, pp. 110-119.

Yamazaki, F., 2001. Seismic monitoring and early damage assessment systems in Japan. Progress in Structural Engineering and Materials Volume 3, pp 66 – 75.

Yamazaki, F., Megiro, K., Noda, S., 1998. Developments of early earthquake damage assessment in Japan. Structural Safety and Reliability, 1570-1580.

Yilmaz Ozturk, N. and Yucemen M. 2008. Probabilistic seismic hazard analysis : a sensitivity study with respect to different models, Phd. Thesis

Yoo, Y. B., Deo N., 1988. A Comparison of algorithms for terminal-pair reliability. IEEE Transactionson Reliability, 37(2).

Bursa Satellite Image, Google. Retrieved December 2010, from http://maps.google.com/

Attenuation Relationship Image, Seismic Risk Assestment and Loss Estimation 2004, Retrieved June 2010, from http://web.mit.edu/istgroup/ist/documents/earthquake/Part1.pdf

APPENDICES

A. PROGRAM CODES

A.1 Main.java Class

```
package org.geotools.demo;
import java.awt.Color;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.Serializable;
import java.util.ArrayList;
import java.util.Date;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.Vector;

import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JComboBox;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JTextField;
import javax.swing.JToolBar;

import org.geotools.coverage.GridSampleDimension;
import org.geotools.coverage.grid.GridCoverage2D;
import
org.geotools.coverage.grid.io.AbstractGridCoverage2DReader;
import org.geotools.coverage.grid.io.AbstractGridFormat;
import org.geotools.coverage.grid.io.GridFormatFinder;
import org.geotools.data.DefaultTransaction;
import org.geotools.data.FeatureSource;
import org.geotools.data.FileDataStore;
import org.geotools.data.FileDataStoreFinder;
```

```
import org.geotools.data.Transaction;
import org.geotools.data.shapefile.ShapefileDataStore;
import org.geotools.data.shapefile.ShapefileDataStoreFactory;
import org.geotools.data.simple.SimpleFeatureCollection;
import org.geotools.data.simple.SimpleFeatureSource;
import org.geotools.data.simple.SimpleFeatureStore;
import org.geotools.demo.node.Node;
import org.geotools.demo.util.Gaussian;
import org.geotools.demo.util.Netrel;
import org.geotools.demo.util.Util;
import org.geotools.factory.CommonFactoryFinder;
import org.geotools.feature.AttributeTypeBuilder;
import org.geotools.feature.FeatureCollections;
import org.geotools.feature.FeatureIterator;
import org.geotools.feature.simple.SimpleFeatureBuilder;
import org.geotools.feature.simple.SimpleFeatureTypeBuilder;
import org.geotools.geometry.DirectPosition2D;
import org.geotools.geometry.jts.ReferencedEnvelope;
import org.geotools.map.DefaultMapContext;
import org.geotools.map.MapContext;
import org.geotools.map.MapLayer;
import org.geotools.referencing.crs.DefaultGeographicCRS;
import org.geotools.styling.ChannelSelection;
import org.geotools.styling.ContrastEnhancement;
import org.geotools.styling.FeatureTypeStyle;
import org.geotools.styling.Fill;
import org.geotools.styling.Graphic;
import org.geotools.styling.Mark;
import org.geotools.styling.RasterSymbolizer;
import org.geotools.styling.Rule;
import org.geotools.styling.SLD;
import org.geotools.styling.SelectedChannelType;
import org.geotools.styling.Stroke;
import org.geotools.styling.Style;
import org.geotools.styling.StyleFactory;
import org.geotools.styling.Symbolizer;
import org.geotools.swing.JMapFrame;
import org.geotools.swing.JMapPane;
import org.geotools.swing.data.JFileDataStoreChooser;
import org.geotools.swing.event.MapMouseEvent;
import org.geotools.swing.event.MapMouseListener;
import org.opengis.feature.simple.SimpleFeature;
import org.opengis.feature.simple.SimpleFeatureType;
import org.opengis.feature.type.AttributeDescriptor;
import org.opengis.feature.type.AttributeType;
import org.opengis.feature.type.GeometryDescriptor;
import org.opengis.filter.Filter;
import org.opengis.filter.FilterFactory2;
import org.opengis.filter.identity.FeatureId;
import org.opengis.style.ContrastMethod;

import com.vividsolutions.jts.geom.Coordinate;
import com.vividsolutions.jts.geom.GeometryFactory;
import com.vividsolutions.jts.geom.LineString;
import com.vividsolutions.jts.geom.LinearRing;
import com.vividsolutions.jts.geom.MultiLineString;
import com.vividsolutions.jts.geom.MultiPolygon;
```

```
import com.vividsolutions.jts.geom.Point;
import com.vividsolutions.jts.geom.Polygon;

/**
 * In this example we create a map tool to select a feature
clicked with the
 * mouse. The selected feature will be painted yellow.
 *
 * @source $URL$
 */
/**
 * SIRA: >loadShapeRaster >selectOtherFeatures >displayLayers
>selectFeatures
 * >displaySelectedFeatures
 */
public class Main
{

      /*
       * Factories that we will use to create style and filter
objects
       */
      private StyleFactory sf =
CommonFactoryFinder.getStyleFactory(null);
      private FilterFactory2 ff =
CommonFactoryFinder.getFilterFactory2(null);
      private GeometryFactory gf = new GeometryFactory();

      // private static final double demand = 0.24;
      private static double stdDev = 0.002;

      private AbstractGridCoverage2DReader reader;
      private Netrel netrel = new Netrel();

      public File logFile;
      public FileWriter logWriter;

      /*
       * Convenient constants for the type of feature geometry in
the shapefile
       */
      private enum GeomType
      {
            POINT, LINE, POLYGON
      };

      /*
       * Some default style variables
       */
      private static final Color LINE_COLOUR = Color.BLACK;
      private static final Color FILL_COLOUR = Color.DARK_GRAY;

      private static final Color MAX_DANGEROUS = Color.RED;
      private static final Color MORE_DANGEROUS = Color.ORANGE;
      private static final Color MODERATE_DANGEROUS =
Color.GREEN;
      private static final Color LESS_DANGEROUS = Color.BLUE;
```

```
	private static final Color MIN_DANGEROUS = Color.CYAN;
	private static final Color OTHER_LINE_COLOR = Color.BLACK;

	private static final float OPACITY = 0.5f;
	private static final float LINE_WIDTH = 1.0f;
	private static final float THICK_LINE_WIDTH = 5.0f;
	private static final float POINT_SIZE = 10.0f;

	private static final double proximity = 0.2;
	private static int counter = 0;

	private static File rasterFile;// = null;// new
                                                        //
File("MAPS/raster/PGA475_modified.tif");
	private static File shapeRasterFile;// = new File(
	//
"MAPS/shape/combined_pga_spectra_point/combined_pga_spectra_point
.shp");
	private static File shapeFile;// = new
                                                        //
File("MAPS/shape/bursa/bursa_polyline.shp");
	private static File tempFile;

	private JMapFrame mapFrame;
	private FeatureSource<SimpleFeatureType, SimpleFeature>
featureSource;
	private SimpleFeatureType originalFeatureType;
	private SimpleFeatureType generatedFeatureType;
	private FeatureSource<SimpleFeatureType, SimpleFeature>
otherFeatureSource;

	private String geometryAttributeName;
	private GeomType geometryType;

	private Set<FeatureId> MAX_IDs = new HashSet<FeatureId>();
	private Set<FeatureId> MORE_IDs = new HashSet<FeatureId>();
	private Set<FeatureId> MODERATE_IDs = new
HashSet<FeatureId>();
	private Set<FeatureId> LESS_IDs = new HashSet<FeatureId>();
	private Set<FeatureId> MIN_IDs = new HashSet<FeatureId>();
	private Set<FeatureId> OTHER_IDs = new
HashSet<FeatureId>();
	private ArrayList<Set<FeatureId>> sets = new
ArrayList<Set<FeatureId>>();

	private ArrayList<MultiLineString> inputMultiLineStrings =
new ArrayList<MultiLineString>();
	private ArrayList<Long> inputMultiLineStringsDirections =
new ArrayList<Long>();
	private Vector<Coordinate> nodeList = new
Vector<Coordinate>();
	private ArrayList<FeatureId> featureIds = new
ArrayList<FeatureId>();
	private double[] reliabilities;
	private int[][] adjacencyMatrix;

	private double generalPgaValAvg;
```

```
	private int totalNumberOfPoints;
	private double totalPgaVal;

	private Coordinate sourcePoint;
	private Coordinate sinkPoint;
	/**
	 * 1 means UPPER 0 means LOWER
	 */
	// private final static int upperOrLower = 1;

	private ArrayList<ArrayList<LinearRing>> gridList;

	boolean fullNetworkColorize = false;
	boolean fullNetworkAverage = false;

	boolean isItFirstRun = true;

	private JComboBox sourcePointCheckBox;
	private JComboBox sinkPointCheckBox;
	private JTextField lowerValueField;
	private JTextField upperValueField;

	private double lastClickTime;
	private double lastX;
	private double lastY;
	private final double doubleClickTime = 300;
	private double networkReliabilityLower;
	private double networkReliabilityUpper;

	private boolean isRasterUsed = true;

	/**
	 * orijinal dosyayı okuyup gerekli düzenlemeleri yapıp
yeniden çağrılıp
	 * çağrılmadıgını tutar
	 */

	/*
	 * The application method
	 */
	public static void main(String[] args)
	{
		Main me = new Main();
		try
		{
			me.logFile = new File("MAPS/log.txt");
			me.logFile.delete();
			me.logFile.createNewFile();
			me.logWriter = new FileWriter(me.logFile);
		}
		catch (IOException e)
		{
			e.printStackTrace();
		}
		JFileDataStoreChooser shapeRasterFileChooser = new
JFileDataStoreChooser("shp");
		shapeRasterFileChooser.setCurrentDirectory(new
```

```
File("MAPS/shape/combined_pga_spectra_point"));
            shapeRasterFileChooser.setDialogTitle("Select Seismic
Hazard Layer");
            shapeRasterFileChooser.showOpenDialog(null);
            shapeRasterFile =
shapeRasterFileChooser.getSelectedFile();
            if (shapeRasterFile == null)
            {
                  return;
            }
            JFileDataStoreChooser shapeFileChooser = new
JFileDataStoreChooser("shp");
            shapeFileChooser.setCurrentDirectory(new
File("MAPS/shape/bursa"));
            shapeFileChooser.setDialogTitle("Select Network
Layer");
            shapeFileChooser.showOpenDialog(null);
            shapeFile = shapeFileChooser.getSelectedFile();

            if (shapeFile == null)
            {
                  return;
            }
            JFileDataStoreChooser rasterFileChooser = new
JFileDataStoreChooser("tif");
            rasterFileChooser.setCurrentDirectory(new
File("MAPS/raster"));
            rasterFileChooser.setDialogTitle("Select Raster File
- Optional");
            rasterFileChooser.showOpenDialog(null);
            rasterFile = rasterFileChooser.getSelectedFile();
            if (rasterFile == null)
            {
                  me.isRasterUsed = false;
            }
            me.sets.add(me.MIN_IDs);
            me.sets.add(me.LESS_IDs);
            me.sets.add(me.MODERATE_IDs);
            me.sets.add(me.MORE_IDs);
            me.sets.add(me.MAX_IDs);
            me.sets.add(me.OTHER_IDs);

            long startTime = new Date().getTime();
            me.loadShapeRaster(shapeRasterFile);
            tempFile = me.generateTempShapeFile(shapeFile);
            me.displayLayers(rasterFile, tempFile);
            long endTime = new Date().getTime();
            long elapsedTime = endTime - startTime;
            try
            {
                  me.logWriter.write("Elapsed Time :-> " +
elapsedTime + " ms" + '\n');
                  me.logWriter.flush();
            }
            catch (IOException e)
            {
                  e.printStackTrace();
```

```
		}
	}

	private File generateTempShapeFile(File shapeFile)
	{
		File f = null;
		FileDataStore store;
		try
		{
			store =
FileDataStoreFinder.getDataStore(shapeFile);
			FeatureSource<SimpleFeatureType, SimpleFeature>
featureSource = store.getFeatureSource();
			originalFeatureType = store.getSchema();
			setGeometry(featureSource);
			ArrayList<MultiLineString>
inputMultiLineStrings = new ArrayList<MultiLineString>();
			Vector<Coordinate> nodeList = new
Vector<Coordinate>();
			ArrayList<Long> inputMultiLineStringsDirections
= new ArrayList<Long>();
			ArrayList<FeatureId> featureIds = new
ArrayList<FeatureId>();
			int[][] pointsAndEdges = new int[][] {};
			ArrayList<Long> directions = new
ArrayList<Long>();
			FeatureIterator<SimpleFeature> featureIterator
= null;
			try
			{
				featureIterator =
featureSource.getFeatures().features();
				Util util = new Util();
				double[] reliabilities = new
double[featureSource.getFeatures().size()];
				int index = 0;
				while (featureIterator.hasNext())
				{
					if (index == reliabilities.length)
						break;
					SimpleFeature feature =
featureIterator.next();
					List<Object> objects =
feature.getAttributes();
					for (Iterator<Object> iterator =
objects.iterator(); iterator.hasNext();)
					{
						// iterates once
						MultiLineString
multiLineString;
						Object object =
iterator.next();
						if (object instanceof
MultiLineString)
							multiLineString =
(MultiLineString) object;
						else if
```

```
(!object.toString().equals(""))
                                    {
                                          directions.add((Long)
object);
                                          continue;
                                    }
                                    else
                                          continue;

      inputMultiLineStrings.add(multiLineString);
                                    Coordinate firstEndPoint =
multiLineString.getCoordinates()[0];
                                    Coordinate secondEndPoint =
multiLineString.getCoordinates()[1];
                                    if
(!nodeList.contains(firstEndPoint))

      nodeList.add(firstEndPoint);
                                    if
(!nodeList.contains(secondEndPoint))

      nodeList.add(secondEndPoint);

                                    LineString lineString =
gf.createLineString(multiLineString.getCoordinates());
                                    ArrayList<LineString>
lineStrings = util.getIntersectionLinearRings(gridList,
lineString, gf,
                                                      null);
                                    reliabilities[index] =
calculateNormalValues(lineStrings, false);

      featureIds.add(feature.getIdentifier());
                                    index++;
                              }
                        }
                        int[][] adjacencyMatrix =
calculateAdjacencyMatrix(inputMultiLineStrings,

      inputMultiLineStringsDirections, nodeList);
                        if (isItFirstRun)
                        {
                              sourcePoint = nodeList.get(8);
                              sinkPoint = nodeList.get(7);
                        }
                        Object[] result =
netrel.findInterval(inputMultiLineStrings.size(),
nodeList.size(), reliabilities,
                                    adjacencyMatrix,
nodeList.indexOf(sourcePoint) + 1, nodeList.indexOf(sinkPoint) +
1);
                        pointsAndEdges = (int[][]) result[0];
                        double lower = (Double) result[1];
                        double upper = (Double) result[2];
                        f = createAndLoadShapeFile(reliabilities,
inputMultiLineStrings, pointsAndEdges, directions, lower,
                                    upper);
```

```
                }
                catch (Exception e)
                {
                        e.printStackTrace();
                }
                finally
                {
                        featureIterator.close();
                }
        }
        catch (IOException e)
        {
                e.printStackTrace();
        }
        return f;
}

private void loadShapeRaster(File shapeRasterFile)
{
        FileDataStore store;
        try
        {
                store =
FileDataStoreFinder.getDataStore(shapeRasterFile);
                otherFeatureSource = store.getFeatureSource();
        }
        catch (IOException e)
        {
                e.printStackTrace();
        }
        selectOtherFeatures(null);
}

/**
 * This method connects to the shapefile; retrieves
information about its
 * features; creates a map frame to display the shapefile
and adds a custom
 * feature selection tool to the toolbar of the map frame.
 */
private void displayLayers(File rasterFile, File shpFile)
{
        Style rasterStyle = null;
        if (isRasterUsed)
        {
                AbstractGridFormat format =
GridFormatFinder.findFormat(rasterFile);
                reader = format.getReader(rasterFile);
                rasterStyle = createRGBStyle();
        }

        FileDataStore store;
        try
        {
                store =
FileDataStoreFinder.getDataStore(shpFile);
                featureSource = store.getFeatureSource();
```

```
            }
            catch (IOException e1)
            {
                  e1.printStackTrace();
            }

            setGeometry(featureSource);

            MapContext map = new DefaultMapContext();
            map.setTitle("Reliability Map Tool");
            Style style = createDefaultStyle();
            if (isRasterUsed)
                  map.addLayer(reader, rasterStyle);
            map.addLayer(featureSource, style);
            mapFrame = new JMapFrame(map);
            mapFrame.enableToolBar(true);
            mapFrame.enableStatusBar(true);
            JMapPane mapPane = mapFrame.getMapPane();

            /*
             * Before making the map frame visible we add a new
button to its
             * toolbar for our custom feature selection tool
             */
            JToolBar toolBar = mapFrame.getToolBar();
            toolBar.setSize(toolBar.getWidth(), 150);

            final JLabel upperValue = new JLabel("U");
            upperValue.setToolTipText("Upper Bound");
            upperValueField = new
JTextField(String.valueOf(networkReliabilityUpper), 8);
            upperValueField.setEditable(false);
            toolBar.addSeparator();
            toolBar.add(upperValue);
            toolBar.add(upperValueField);

            final JLabel lowerValue = new JLabel("L");
            lowerValue.setToolTipText("Lower Bound");
            lowerValueField = new
JTextField(String.valueOf(networkReliabilityLower), 8);
            lowerValueField.setEditable(false);
            toolBar.addSeparator();
            toolBar.add(lowerValue);
            toolBar.add(lowerValueField);

            final JTextField stdDevTextField = new
JTextField(String.valueOf(stdDev), 8);
            JLabel stdDevLabel = new JLabel("Var:");
            stdDevLabel.setToolTipText("Variance");
            toolBar.addSeparator();
            toolBar.add(stdDevLabel);
            toolBar.add(stdDevTextField);

            final JCheckBox fullNetworkColorizeCheckBox = new
JCheckBox("E.R.");
            fullNetworkColorizeCheckBox.setToolTipText("Show
element reliability");
```

```
            // fullNetworkColorizeCheckBox.setSelected(true);
            toolBar.addSeparator();
            toolBar.add(fullNetworkColorizeCheckBox);

            final JCheckBox fullNetworkAverageCheckBox = new
JCheckBox("Mean");
            fullNetworkAverageCheckBox.setToolTipText("Use
Population Mean");
            toolBar.addSeparator();
            toolBar.add(fullNetworkAverageCheckBox);

            // final JCheckBox upperValueCheckBox = new
JCheckBox("U.B.");
            // upperValueCheckBox.setToolTipText("Use Upper
Bounds For Network Reliability Calculation");
            // toolBar.addSeparator();
            // toolBar.add(upperValueCheckBox);

            JLabel sourcePointLabel = new JLabel("S:");
            sourcePointLabel.setToolTipText("Source Point");
            sourcePointCheckBox = new JComboBox(nodeList);
            sourcePointCheckBox.setToolTipText("Select source
point from the list");
            sourcePointCheckBox.setIgnoreRepaint(true);
            sourcePointCheckBox.addActionListener(new
ActionListener()
            {
                  @Override
                  public void actionPerformed(ActionEvent e)
                  {
                        if (!isItFirstRun)
                        {
                              reCalculate(stdDevTextField,
fullNetworkColorizeCheckBox, fullNetworkAverageCheckBox);
                        }
                  }
            });
            toolBar.addSeparator();
            toolBar.add(sourcePointLabel);
            toolBar.add(sourcePointCheckBox);

            JLabel sinkPointLabel = new JLabel("E:");
            sinkPointLabel.setToolTipText("Sink Point");
            sinkPointCheckBox = new JComboBox(nodeList);
            sinkPointCheckBox.setToolTipText("Select sink point
from the list");
            sinkPointCheckBox.setIgnoreRepaint(true);
            sinkPointCheckBox.addActionListener(new
ActionListener()
            {
                  @Override
                  public void actionPerformed(ActionEvent e)
                  {
                        if (!isItFirstRun)
                        {
                              reCalculate(stdDevTextField,
fullNetworkColorizeCheckBox, fullNetworkAverageCheckBox);
```

```
                        }
                    }
                });
                toolBar.addSeparator();
                toolBar.add(sinkPointLabel);
                toolBar.add(sinkPointCheckBox);

                JButton reRunButton = new JButton("Run");
                toolBar.addSeparator();
                toolBar.add(reRunButton);

                reRunButton.addActionListener(new ActionListener()
                {
                    @Override
                    public void actionPerformed(ActionEvent arg0)
                    {
                        reCalculate(stdDevTextField,
fullNetworkColorizeCheckBox, fullNetworkAverageCheckBox);
                    }
                });

                mapPane.addMouseListener(new MapMouseListener()
                {
                    @Override
                    public void onMouseWheelMoved(MapMouseEvent ev)
                    {
                    }

                    @Override
                    public void onMouseReleased(MapMouseEvent ev)
                    {
                    }

                    @Override
                    public void onMousePressed(MapMouseEvent ev)
                    {
                    }

                    @Override
                    public void onMouseMoved(MapMouseEvent ev)
                    {
                    }

                    @Override
                    public void onMouseExited(MapMouseEvent ev)
                    {
                    }

                    @Override
                    public void onMouseEntered(MapMouseEvent ev)
                    {
                    }

                    @Override
                    public void onMouseDragged(MapMouseEvent ev)
                    {
                    }
```

```
                  @Override
                  public void onMouseClicked(MapMouseEvent ev)
                  {
                        long clickTime = ev.getWhen();
                        if ((lastX == ev.getX()) && (lastY ==
ev.getY()) && ((clickTime - lastClickTime) < doubleClickTime))
                        {
                              DirectPosition2D directPosition2D =
ev.getMapPosition();
                              Coordinate c = new
Coordinate(directPosition2D.x, directPosition2D.y);
                              Coordinate found =
findClickedNode(c, nodeList, proximity);
                              if (counter == 0 && found != null)
                              {
                                    sourcePoint = found;

      sourcePointCheckBox.setSelectedItem(sourcePoint);
                                    reCalculate(stdDevTextField,
fullNetworkColorizeCheckBox, fullNetworkAverageCheckBox);
                                    counter++;
                              }
                              else if (counter == 1 && found !=
null)
                              {
                                    sinkPoint = found;

      sinkPointCheckBox.setSelectedItem(sinkPoint);
                                    reCalculate(stdDevTextField,
fullNetworkColorizeCheckBox, fullNetworkAverageCheckBox);
                                    counter--;
                              }
                        }
                        else
                        {
                              lastClickTime = clickTime;
                              lastX = ev.getX();
                              lastY = ev.getY();
                        }
                  }
            });

            selectFeatures(null);

            /**
             * Finally, we display the map frame. When it is
closed this application
             * will exit.
             */
            mapFrame.setSize(1280, 780);
            mapFrame.setVisible(true);
      }

      private Coordinate findClickedNode(Coordinate c,
Vector<Coordinate> nodeList, double proximity)
      {
```

```
            double distance = 1000;
            int index = -1;
            for (int i = 0; i < nodeList.size(); i++)
            {
                  Coordinate node = nodeList.elementAt(i);
                  if (c.distance(node) < distance)
                  {
                        distance = c.distance(node);
                        index = i;
                  }
            }
            if (distance < proximity && index != -1)
                  return nodeList.get(index);
            return null;
      }

      private void reCalculate(JTextField stdDevTextField,
JCheckBox fullNetworkColorizeCheckBox,
                  JCheckBox fullNetworkAverageCheckBox)
      {
            try
            {
                  stdDev =
Double.valueOf(stdDevTextField.getText()).doubleValue();
                  if (stdDev >= 1 || stdDev <= 0)
                        throw (new NumberFormatException());
                  fullNetworkColorize =
fullNetworkColorizeCheckBox.isSelected();
                  fullNetworkAverage =
fullNetworkAverageCheckBox.isSelected();
                  // upperOrLower =
upperValueCheckBox.isSelected() ? 1 : 0;
                  sourcePoint = (Coordinate)
sourcePointCheckBox.getSelectedItem();
                  sinkPoint = (Coordinate)
sinkPointCheckBox.getSelectedItem();
                  tempFile = generateTempShapeFile(shapeFile);
                  selectFeatures(null);
            }
            catch (NumberFormatException exception)
            {
                  if (exception.getMessage() == null)
                        JOptionPane.showMessageDialog(mapFrame,
"Enter a decimal number between 0 and 1", "Warning",
                                    JOptionPane.WARNING_MESSAGE);
                  else
                        JOptionPane.showMessageDialog(mapFrame,
"Enter valid decimal number", "Warning",
                                    JOptionPane.WARNING_MESSAGE);
                  stdDev = 0.002;
                  stdDevTextField.setText("0.002");
            }
      }

      /**
       * This method is called by our feature selection tool when
the user has
```

```
	 * clicked on the map.
	 *
	 * @param pos
	 *            map (world) coordinates of the mouse cursor
	 * @throws IOException
	 */
	void selectFeatures(MapMouseEvent ev)
	{
		MAX_IDs.clear();
		MORE_IDs.clear();
		MODERATE_IDs.clear();
		LESS_IDs.clear();
		MIN_IDs.clear();
		inputMultiLineStrings.clear();
		nodeList.clear();
		inputMultiLineStringsDirections.clear();
		featureIds.clear();
		int[][] pointsAndEdges = new int[][] {};
		FeatureIterator<SimpleFeature> featureIterator =
null;
		ArrayList<ArrayList<LineString>> smallLines = new
ArrayList<ArrayList<LineString>>();
		try
		{
			featureIterator =
featureSource.getFeatures().features();
			Util util = new Util();
			if (isItFirstRun)
				reliabilities = new
double[featureSource.getFeatures().size()];
			int index = 0;
			while (featureIterator.hasNext())
			{
				if (index == reliabilities.length)
					break;
				SimpleFeature feature =
featureIterator.next();
				List<Object> objects =
feature.getAttributes();
				for (Iterator<Object> iterator =
objects.iterator(); iterator.hasNext();)
				{
					// iterates once
					MultiLineString multiLineString;
					Object object = iterator.next();
					if (object instanceof
MultiLineString)
						multiLineString =
(MultiLineString) object;
					else
						continue;
					logWriter.write("MultiLineString :-
> " + multiLineString + "\n");
					logWriter.flush();

	inputMultiLineStrings.add(multiLineString);
					//
```

```
inputMultiLineStringsDirections.add(direction);
                                Coordinate firstEndPoint =
multiLineString.getCoordinates()[0];
                                Coordinate secondEndPoint =
multiLineString.getCoordinates()[1];
                                if
(!nodeList.contains(firstEndPoint))
                                      nodeList.add(firstEndPoint);
                                if
(!nodeList.contains(secondEndPoint))
                                      nodeList.add(secondEndPoint);

                                LineString lineString =
gf.createLineString(multiLineString.getCoordinates());
                                ArrayList<LineString> lineStrings =
util.getIntersectionLinearRings(gridList, lineString, gf,
                                            logWriter);
                                smallLines.add(lineStrings);
                                double reliability =
calculateNormalValues(lineStrings, true);
                                logWriter.write("Reliability :-> "
+ reliability + '\n');
                                logWriter.append('\n');
                                logWriter.flush();
                                reliabilities[index] = reliability;

      featureIds.add(feature.getIdentifier());
                                index++;
                          }
                  }
                  adjacencyMatrix =
calculateAdjacencyMatrix(inputMultiLineStrings,
inputMultiLineStringsDirections, nodeList);
                  if (isItFirstRun)
                  {

      sourcePointCheckBox.setSelectedItem(sourcePoint);

      sinkPointCheckBox.setSelectedItem(sinkPoint);
                  }
                  Object[] result =
netrel.findInterval(inputMultiLineStrings.size(),
nodeList.size(), reliabilities,
                                adjacencyMatrix,
nodeList.indexOf(sourcePoint) + 1, nodeList.indexOf(sinkPoint) +
1);
                  pointsAndEdges = (int[][]) result[0];
                  networkReliabilityUpper = (Double) result[2];
                  networkReliabilityLower = (Double) result[1];

      upperValueField.setText(String.valueOf(networkReliabilityUp
per));

      lowerValueField.setText(String.valueOf(networkReliabilityLo
wer));
                  logWriter.write("Lower Value: " +
networkReliabilityLower + '\n');
```

```
                  logWriter.write("Upper Value: " +
networkReliabilityUpper + '\n');
                  logWriter.flush();
                  SimpleFeatureStore featureStore =
(SimpleFeatureStore) featureSource;
                  featureStore.modifyFeatures("Upper",
networkReliabilityUpper, Filter.INCLUDE);
                  featureStore.modifyFeatures("Lower",
networkReliabilityLower, Filter.INCLUDE);
                  ArrayList<Coordinate> coordinates = new
ArrayList<Coordinate>();
                  coordinates.add(sinkPoint);
                  coordinates.add(sourcePoint);
                  addCustomShapes(coordinates,
(SimpleFeatureStore) featureSource);
                  isItFirstRun = false;
            }
            catch (Exception e)
            {
                  e.printStackTrace();
            }
            finally
            {
                  featureIterator.close();
            }
            createShapeMapForSmallLines(smallLines);
            colorizeFullNetwork(pointsAndEdges,
fullNetworkColorize, reliabilities, featureIds);
      }

      private void
createShapeMapForSmallLines(ArrayList<ArrayList<LineString>>
smallLines)
      {
            File newFile = null;
            try
            {
                  newFile = getNewShapeFile(shapeFile,
"lineSegments.shp");
                  newFile.delete();
                  newFile.createNewFile();
                  SimpleFeatureCollection collection =
FeatureCollections.newCollection();

                  final SimpleFeatureType TYPE =
createFeatureType();
                  SimpleFeatureBuilder featureBuilder = new
SimpleFeatureBuilder(TYPE);
                  for (ArrayList<LineString> lineStrings :
smallLines)
                  {
                        for (LineString lineString : lineStrings)
                        {
                              featureBuilder.add(lineString);
                              featureBuilder.add((Double)
lineString.getUserData());
                              SimpleFeature feature =
```

```
featureBuilder.buildFeature(null);
                              collection.add(feature);
                        }
                  }
                  ShapefileDataStoreFactory dataStoreFactory =
new ShapefileDataStoreFactory();

                  Map<String, Serializable> params = new
HashMap<String, Serializable>();
                  params.put("url", newFile.toURI().toURL());
                  params.put("create spatial index",
Boolean.TRUE);

                  ShapefileDataStore newDataStore =
(ShapefileDataStore) dataStoreFactory.createNewDataStore(params);
                  newDataStore.createSchema(TYPE);

                  /*
                   * You can comment out this line if you are
using the
                   * createFeatureType method (at end of class
file) rather than
                   * DataUtilities.createType
                   */

      newDataStore.forceSchemaCRS(DefaultGeographicCRS.WGS84);
                  Transaction transaction = new
DefaultTransaction("segmentHandle");

                  String typeName =
newDataStore.getTypeNames()[0];
                  SimpleFeatureSource featureSource =
newDataStore.getFeatureSource(typeName);

                  if (featureSource instanceof
SimpleFeatureStore)
                  {
                        SimpleFeatureStore featureStore =
(SimpleFeatureStore) featureSource;

                        featureStore.setTransaction(transaction);
                        try
                        {

      featureStore.addFeatures(collection);
                              transaction.commit();

                        }
                        catch (Exception problem)
                        {
                              problem.printStackTrace();
                              transaction.rollback();
                        }
                        finally
                        {
                              transaction.close();
                        }
```

```
                }
                else
                {
                        System.out.println(typeName + " does not
support read/write access");
                        System.exit(1);
                }
            }
            catch (Exception e)
            {
                e.printStackTrace();
            }
      }

      private static SimpleFeatureType createFeatureType()
      {
            SimpleFeatureTypeBuilder builder = new
SimpleFeatureTypeBuilder();
            builder.setName("Segment");
            builder.setCRS(DefaultGeographicCRS.WGS84); // <-
Coordinate reference

            // system
            // add attributes in order
            builder.add("Segment", LineString.class);
            builder.length(15).add("PGA", Double.class); // <- 15
chars width for

                  // name field
            // build the type
            final SimpleFeatureType SEGMENT =
builder.buildFeatureType();

            return SEGMENT;
      }

      private File createAndLoadShapeFile(double[] reliabilities,
ArrayList<MultiLineString> inputMultiLineStrings,
                  int[][] pointsAndEdges, ArrayList<Long>
directions, double lower, double upper)
      {
            File newFile = null;
            try
            {
                  newFile = getNewShapeFile(shapeFile,
"analysis.shp");
                  newFile.delete();
                  newFile.createNewFile();
                  SimpleFeatureCollection collection =
FeatureCollections.newCollection();

                  /*
                   * GeometryFactory will be used to create the
geometry attribute of
                   * each feature (a Point object for the
location)
                   */
```

```
                    AttributeTypeBuilder attributeTypeBuilder1 =
new AttributeTypeBuilder();
                    attributeTypeBuilder1.setNillable(true);
                    attributeTypeBuilder1.setBinding(Double.class);
                    attributeTypeBuilder1.setName("Reliabilit");
                    AttributeType reliabilityType1 =
attributeTypeBuilder1.buildType();
                    AttributeDescriptor attributeDescriptor1 =
attributeTypeBuilder1.buildDescriptor("Reliabilit",
                                reliabilityType1);

                    AttributeTypeBuilder attributeTypeBuilder2 =
new AttributeTypeBuilder();
                    attributeTypeBuilder2.setNillable(true);
                    attributeTypeBuilder2.setBinding(Double.class);
                    attributeTypeBuilder2.setName("Lower");
                    AttributeType reliabilityType2 =
attributeTypeBuilder2.buildType();
                    AttributeDescriptor attributeDescriptor2 =
attributeTypeBuilder2.buildDescriptor("Lower", reliabilityType2);

                    AttributeTypeBuilder attributeTypeBuilder3 =
new AttributeTypeBuilder();
                    attributeTypeBuilder3.setNillable(true);
                    attributeTypeBuilder3.setBinding(Double.class);
                    attributeTypeBuilder3.setName("Upper");
                    AttributeType reliabilityType3 =
attributeTypeBuilder3.buildType();
                    AttributeDescriptor attributeDescriptor3 =
attributeTypeBuilder3.buildDescriptor("Upper", reliabilityType3);

                    SimpleFeatureTypeBuilder featureTypeBuilder =
new SimpleFeatureTypeBuilder();
                    featureTypeBuilder.init(originalFeatureType);
                    featureTypeBuilder.remove("BURSA8");
                    featureTypeBuilder.add(attributeDescriptor1);
                    featureTypeBuilder.add(attributeDescriptor2);
                    featureTypeBuilder.add(attributeDescriptor3);
                    generatedFeatureType =
featureTypeBuilder.buildFeatureType();
                    SimpleFeatureBuilder featureBuilder = new
SimpleFeatureBuilder(generatedFeatureType);
                    for (int i = 0; i <
inputMultiLineStrings.size(); i++)
                    {

      featureBuilder.add(inputMultiLineStrings.get(i));
                          featureBuilder.add(reliabilities[i]);
                          featureBuilder.add(lower);
                          featureBuilder.add(upper);

                          SimpleFeature feature =
featureBuilder.buildFeature(null);
                          collection.add(feature);
                    }
```

```
                  ShapefileDataStoreFactory dataStoreFactory =
new ShapefileDataStoreFactory();

                  Map<String, Serializable> params = new
HashMap<String, Serializable>();
                  params.put("url", newFile.toURI().toURL());
                  params.put("create spatial index",
Boolean.TRUE);

                  ShapefileDataStore newDataStore =
(ShapefileDataStore) dataStoreFactory.createNewDataStore(params);

      newDataStore.createSchema(generatedFeatureType);

                  /*
                   * You can comment out this line if you are
using the
                   * createFeatureType method (at end of class
file) rather than
                   * DataUtilities.createType
                   */

      newDataStore.forceSchemaCRS(DefaultGeographicCRS.WGS84);

                  // docs break transaction
                  /*
                   * Write the features to the shapefile
                   */
                  Transaction transaction = new
DefaultTransaction("create");

                  String typeName =
newDataStore.getTypeNames()[0];
                  SimpleFeatureSource featureSource =
newDataStore.getFeatureSource(typeName);

                  if (featureSource instanceof
SimpleFeatureStore)
                  {
                        SimpleFeatureStore featureStore =
(SimpleFeatureStore) featureSource;

                        featureStore.setTransaction(transaction);
                        try
                        {

      featureStore.addFeatures(collection);
                              transaction.commit();
                        }
                        catch (Exception problem)
                        {
                              problem.printStackTrace();
                              transaction.rollback();

                        }
                        finally
                        {
```

```
                              transaction.close();
                        }
                  }
                  else
                  {
                        System.out.println(typeName + " does not
support read/write access");
                        System.exit(1);
                  }
            }
            catch (Exception e)
            {
                  e.printStackTrace();
            }
            return newFile;
      }

      private static File getNewShapeFile(File shapeFile, String
fileName)
      {
            String path = shapeFile.getAbsolutePath();
            String folderPath = path.substring(0, path.length() -
shapeFile.getName().length()) + "generated";
            boolean folderExist = new File(folderPath).exists();
            String newPath = null;
            if (!folderExist)
            {
                  new File(folderPath).mkdir();
            }

            newPath = folderPath + "/" + fileName;
            JFileDataStoreChooser chooser = new
JFileDataStoreChooser("shp");
            chooser.setDialogTitle("Save generated shapefile");
            chooser.setSelectedFile(new File(newPath));

            int returnVal = chooser.showSaveDialog(null);

            if (returnVal !=
JFileDataStoreChooser.APPROVE_OPTION)
            {
                  // the user cancelled the dialog
                  System.exit(0);
            }

            File newFile = chooser.getSelectedFile();
            if (newFile.equals(shapeFile))
            {
                  System.out.println("Error: cannot replace " +
shapeFile);
                  System.exit(0);
            }

            return newFile;
      }

      private void colorizeFullNetwork(final int[][]
```

```
pointsAndEdges, final boolean colorizeFullMode,
                  final double[] reliabilities, final
ArrayList<FeatureId> featureIds)
      {
            for (Set<FeatureId> set : sets)
                  set.clear();
            if (!colorizeFullMode)
            {
                  int renklendirilenSayisi = 0;
                  int renklendirilmeyenSayisi = 0;
                  Set<FeatureId> colorableFeatureIds = new
HashSet<FeatureId>();
                  for (int edgeIndex = 0; edgeIndex <
pointsAndEdges.length; edgeIndex++)
                  {
                        int[] edgeArray =
pointsAndEdges[edgeIndex];
                        int lineStringIndex =
findLine(edgeArray[0], edgeArray[1]);
                        if (reliabilities[lineStringIndex] < 0.2)

      sets.get(0).add(featureIds.get(lineStringIndex));
                        else if (reliabilities[lineStringIndex] <
0.4)

      sets.get(1).add(featureIds.get(lineStringIndex));
                        else if (reliabilities[lineStringIndex] <
0.6)

      sets.get(2).add(featureIds.get(lineStringIndex));
                        else if (reliabilities[lineStringIndex] <
0.8)

      sets.get(3).add(featureIds.get(lineStringIndex));
                        else

      sets.get(4).add(featureIds.get(lineStringIndex));

      colorableFeatureIds.add(featureIds.get(lineStringIndex));
                        renklendirilenSayisi++;
                  }
                  for (int edgeIndex = 0; edgeIndex <
featureIds.size(); edgeIndex++)
                  {
                        FeatureId featureId =
featureIds.get(edgeIndex);
                        if
(!colorableFeatureIds.contains(featureId))
                        {
                              sets.get(5).add(featureId);
                              renklendirilmeyenSayisi++;
                        }
                  }
            }
            else
            {
                  for (int edgeIndex = 0; edgeIndex <
```

```
featureIds.size(); edgeIndex++)
                  {
                        if (reliabilities[edgeIndex] < 0.2)

      sets.get(0).add(featureIds.get(edgeIndex));
                        else if (reliabilities[edgeIndex] < 0.4)

      sets.get(1).add(featureIds.get(edgeIndex));
                        else if (reliabilities[edgeIndex] < 0.6)

      sets.get(2).add(featureIds.get(edgeIndex));
                        else if (reliabilities[edgeIndex] < 0.8)

      sets.get(3).add(featureIds.get(edgeIndex));
                        else

      sets.get(4).add(featureIds.get(edgeIndex));
                  }
            }
            displaySelectedFeatures(sets);
      }

      private int findLine(int i, int j)
      {
            Coordinate start = nodeList.get(i);
            Coordinate end = nodeList.get(j);
            for (MultiLineString multiLineString :
inputMultiLineStrings)
            {
                  Coordinate[] endPoints =
multiLineString.getCoordinates();
                  if ((endPoints[0].equals(start) &&
endPoints[1].equals(end))
                              || (endPoints[1].equals(start) &&
endPoints[0].equals(end)))
                  {
                        return
inputMultiLineStrings.indexOf(multiLineString);
                  }
            }
            return -1;
      }

      private void addCustomShapes(ArrayList<Coordinate>
nodeList, SimpleFeatureStore featureStore)
      {
            double sourceLineLength = 0.01;
            double sinkLineLength = 0.005;
            Transaction transaction = new
DefaultTransaction("handle");
            SimpleFeatureCollection collection =
FeatureCollections.newCollection();
            featureStore.setTransaction(transaction);

            SimpleFeatureTypeBuilder featureTypeBuilder = new
SimpleFeatureTypeBuilder();
            featureTypeBuilder.init(generatedFeatureType);
```

```
            SimpleFeatureType simpleFeatureType =
featureTypeBuilder.buildFeatureType();
            SimpleFeatureBuilder featureBuilder = new
SimpleFeatureBuilder(simpleFeatureType);
            try
            {
                  Coordinate sink = nodeList.get(0);
                  Coordinate source = nodeList.get(1);

                  MultiLineString grid1 = createCross(source,
sourceLineLength);
                  featureBuilder.add(grid1);
                  SimpleFeature feature =
featureBuilder.buildFeature(null);
                  collection.add(feature);

                  MultiLineString grid2 = createCross(sink,
sinkLineLength);
                  featureBuilder.add(grid2);
                  SimpleFeature feature2 =
featureBuilder.buildFeature(null);
                  collection.add(feature2);

                  featureStore.addFeatures(collection);
            }
            catch (Exception e)
            {
                  e.printStackTrace();
            }
      }

      private MultiLineString createCross(Coordinate point,
double crossLength)
      {
            LineString l1 = gf.createLineString(new Coordinate[]
{
                        new Coordinate(point.x - crossLength,
point.y - crossLength),
                        new Coordinate(point.x - crossLength,
point.y + crossLength) });
            LineString l2 = gf.createLineString(new Coordinate[]
{
                        new Coordinate(point.x - crossLength,
point.y + crossLength),
                        new Coordinate(point.x + crossLength,
point.y + crossLength) });
            LineString l3 = gf.createLineString(new Coordinate[]
{
                        new Coordinate(point.x + crossLength,
point.y + crossLength),
                        new Coordinate(point.x + crossLength,
point.y - crossLength) });
            LineString l4 = gf.createLineString(new Coordinate[]
{
                        new Coordinate(point.x + crossLength,
point.y - crossLength),
                        new Coordinate(point.x - crossLength,
```

```
point.y - crossLength) });
            MultiLineString grid = gf.createMultiLineString(new
LineString[] { l1, l2, l3, l4 });
            return grid;
      }

      private int[][]
calculateAdjacencyMatrix(ArrayList<MultiLineString>
inputMultiLineStrings,
                  ArrayList<Long>
inputMultiLineStringsDirections, Vector<Coordinate> nodeList)
      {
            int[][] adjacency = new
int[nodeList.size()][nodeList.size()];
            for (int i = 0; i < nodeList.size(); i++)
            {
                  Coordinate coordinate = nodeList.get(i);
                  for (int j = 0; j <
inputMultiLineStrings.size(); j++)
                  {
                        Coordinate otherCoordinate = null;
                        MultiLineString multiLineString =
inputMultiLineStrings.get(j);
                        if
(multiLineString.getCoordinates()[0].equals(coordinate))
                        {
                              otherCoordinate =
multiLineString.getCoordinates()[1];

      adjacency[nodeList.indexOf(coordinate)][nodeList.indexOf(ot
herCoordinate)] = 1;
                        }
                  }
            }
            return adjacency;
      }

      private double calculateAvgPgaVal(ArrayList<LineString>
lineStrings)
      {
            double totalPgaValue = 0;
            double avgPgaValue = 0;
            for (LineString lineString : lineStrings)
            {
                  totalPgaValue += (Double)
lineString.getUserData();
            }
            if (lineStrings.size() == 0)
                  try
                  {
                        throw new Exception();
                  }
                  catch (Exception e)
                  {
                        e.printStackTrace();
                  }
            avgPgaValue = totalPgaValue / lineStrings.size();
```

```
            return avgPgaValue;
      }

      /**
       *
       * @param lineStrings
       * @param lowerUpper
       *            0 means lower, 1 means upper
       * @return
       */
      private double calculateNormalValues(ArrayList<LineString>
lineStrings, boolean log)
      {
            double result = 1;
            double avgPgaValue;
            ArrayList<Double> zVals = new ArrayList<Double>();
            int size = lineStrings.size();
            if (fullNetworkAverage)
                  avgPgaValue = generalPgaValAvg;
            else
                  avgPgaValue = calculateAvgPgaVal(lineStrings);
            for (int i = 0; i < size; i++)
            {
                  LineString lineString = lineStrings.get(i);
                  double pgaVal = (Double)
lineString.getUserData();
                  double zVal;
                  if (fullNetworkAverage)
                        zVal = Math.abs((avgPgaValue - pgaVal)) /
Math.sqrt(stdDev);
                  else
                        zVal = Math.abs((avgPgaValue - pgaVal)) /
Math.sqrt(stdDev / size);
                  double zValNormal = Gaussian.Phi(zVal);
                  zVals.add(zValNormal);
                  if (log)
                  {
                        try
                        {
                              logWriter.write("Z-Value :-> " +
zValNormal + '\n');
                              logWriter.flush();
                        }
                        catch (IOException e)
                        {
                              e.printStackTrace();
                        }
                  }
            }
            for (double zValNormal : zVals)
            {
                  // if (upperOrLower == 0)
                  // {
                  // result *= zValNormal;
                  // }
                  // else if (upperOrLower == 1)
                  // {
```

```
                  // if (result > zValNormal)
                  // result = zValNormal;
                  // }
                  if (result > zValNormal)
                        result = zValNormal;
            }
            return result;
      }

      void selectOtherFeatures(MapMouseEvent ev)
      {
            try
            {
                  FeatureIterator<SimpleFeature> featureIterator
= otherFeatureSource.getFeatures().features();
                  ReferencedEnvelope envelope =
otherFeatureSource.getBounds();
                  double maxX = envelope.getMaxX();
                  totalNumberOfPoints = 0;
                  totalPgaVal = 0;
                  try
                  {
                        featureIterator =
otherFeatureSource.getFeatures().features();
                        ArrayList<Node> nodeList = new
ArrayList<Node>();
                        ArrayList<ArrayList<Node>> nodesList =
new ArrayList<ArrayList<Node>>();
                        while (featureIterator.hasNext())
                        {
                              SimpleFeature feature =
featureIterator.next();
                              List<Object> objects =
feature.getAttributes();
                              Iterator<Object> iterator =
objects.iterator();
                              Point point = (Point)
iterator.next();
                              iterator.next();
                              iterator.next();
                              double pgaValue = (Double)
iterator.next();
                              Node node = new Node();
                              node.pgaValue = pgaValue;
                              node.point = point;
                              if (point.getX() < maxX)
                              {
                                    nodeList.add(node);
                              }
                              else
                              {
                                    nodeList.add(node);
                                    nodesList.add(nodeList);
                                    nodeList = new
ArrayList<Node>();
                              }
                              totalNumberOfPoints++;
```

```
                              totalPgaVal += pgaValue;
                        }
                        gridList = constructGrid(nodesList);
                        if (totalNumberOfPoints == 0)
                              throw (new Exception());
                        generalPgaValAvg = totalPgaVal /
totalNumberOfPoints;
                  }
                  catch (Exception ex)
                  {
                        ex.printStackTrace();
                        return;
                  }
                  finally
                  {
                        featureIterator.close();
                  }
            }
            catch (Exception ex)
            {
                  ex.printStackTrace();
                  return;
            }
      }

      private ArrayList<ArrayList<LinearRing>>
constructGrid(ArrayList<ArrayList<Node>> input)
      {
            ArrayList<ArrayList<LinearRing>> outerGridList = new
ArrayList<ArrayList<LinearRing>>();
            ArrayList<Node> firstParam = input.get(0);
            for (int i = 1; i < input.size(); i++)
            {
                  ArrayList<Node> secondParam = input.get(i);
                  ArrayList<LinearRing> innerGridList = new
ArrayList<LinearRing>();
                  for (int j = 0; j < firstParam.size() - 1; j++)
                  {
                        Node solAlt = firstParam.get(j);
                        Node solUst = secondParam.get(j);
                        Node sagAlt = firstParam.get(j + 1);
                        Node sagUst = secondParam.get(j + 1);

                        LinearRing grid =
createGridFromNode(solAlt, solUst, sagAlt, sagUst);
                        innerGridList.add(grid);
                  }
                  outerGridList.add(innerGridList);
                  firstParam = new ArrayList<Node>(secondParam);
            }
            return outerGridList;
      }

      private LinearRing createGridFromNode(Node solAlt, Node
solUst, Node sagAlt, Node sagUst)
      {
            double avgPGA = (solAlt.pgaValue + solUst.pgaValue +
```

```
sagAlt.pgaValue + sagUst.pgaValue) / 4;
            Coordinate c1 = solAlt.point.getCoordinate();
            Coordinate c2 = sagAlt.point.getCoordinate();
            Coordinate c3 = sagUst.point.getCoordinate();
            Coordinate c4 = solUst.point.getCoordinate();
            LinearRing grid = gf.createLinearRing(new
Coordinate[] { c1, c2, c3, c4, c1 });
            grid.setUserData(avgPGA);
            return grid;
      }

      // docs end select features

      // docs start display selected
      /**
       * Sets the display to paint selected features yellow and
unselected
       * features in the default style.
       *
       * @param IDs
       *            identifiers of currently selected features
       */
      public void
displaySelectedFeatures(ArrayList<Set<FeatureId>> sets)
      {
            Style style;

            style = createSelectedStyle(sets);
            MapContext mapContext = mapFrame.getMapContext();
            MapLayer mapLayer;
            if (isRasterUsed)
                  mapLayer = mapContext.getLayer(1);
            else
                  mapLayer = mapContext.getLayer(0);
            mapLayer.setStyle(style);
            mapFrame.getMapPane().repaint();
      }

      /**
       * Create a default Style for feature display
       */
      private Style createDefaultStyle()
      {
            Rule rule = createRule(LINE_COLOUR, FILL_COLOUR,
LINE_WIDTH);

            FeatureTypeStyle fts = sf.createFeatureTypeStyle();
            fts.rules().add(rule);

            Style style = sf.createStyle();
            style.featureTypeStyles().add(fts);
            return style;
      }

      /**
       * Create a Style where features with given IDs are painted
yellow, while
```

```
       * others are painted with the default colors.
       */
      private Style createSelectedStyle(ArrayList<Set<FeatureId>>
sets)
      {
            Rule maxRule = createRule(MAX_DANGEROUS,
MAX_DANGEROUS, LINE_WIDTH);
            maxRule.setFilter(ff.id(sets.get(0)));
            //
            Rule moreRule = createRule(MORE_DANGEROUS,
MORE_DANGEROUS, LINE_WIDTH);
            moreRule.setFilter(ff.id(sets.get(1)));
            //
            Rule moderateRule = createRule(MODERATE_DANGEROUS,
MODERATE_DANGEROUS, LINE_WIDTH);
            moderateRule.setFilter(ff.id(sets.get(2)));
            //
            Rule lessRule = createRule(LESS_DANGEROUS,
LESS_DANGEROUS, LINE_WIDTH);
            lessRule.setFilter(ff.id(sets.get(3)));
            //
            Rule minRule = createRule(MIN_DANGEROUS,
MIN_DANGEROUS, LINE_WIDTH);
            minRule.setFilter(ff.id(sets.get(4)));
            //
            Rule uncoloredLineRule = createRule(OTHER_LINE_COLOR,
OTHER_LINE_COLOR, LINE_WIDTH);
            uncoloredLineRule.setFilter(ff.id(sets.get(5)));
            //
            Rule otherRule = createRule(FILL_COLOUR, FILL_COLOUR,
THICK_LINE_WIDTH);
            otherRule.setElseFilter(true);

            FeatureTypeStyle fts = sf.createFeatureTypeStyle();
            fts.rules().add(otherRule);
            fts.rules().add(maxRule);
            fts.rules().add(moreRule);
            fts.rules().add(moderateRule);
            fts.rules().add(lessRule);
            fts.rules().add(minRule);
            if (!fullNetworkColorize)
                  fts.rules().add(uncoloredLineRule);
            else
                  fts.rules().remove(uncoloredLineRule);
            Style style = sf.createStyle();
            style.featureTypeStyles().add(fts);
            return style;
      }

      /**
       * Helper for createXXXStyle methods. Creates a new Rule
containing a
       * Symbolizer tailored to the geometry type of the features
that we are
       * displaying.
       */
      private Rule createRule(Color outlineColor, Color
```

```
fillColor, double lineWidth)
      {
            Symbolizer symbolizer = null;
            Fill fill = null;
            Stroke stroke =
sf.createStroke(ff.literal(outlineColor), ff.literal(lineWidth));

            switch (geometryType)
            {
            case POLYGON:
                  fill = sf.createFill(ff.literal(fillColor),
ff.literal(OPACITY));
                  symbolizer = sf.createPolygonSymbolizer(stroke,
fill, geometryAttributeName);
                  break;

            case LINE:
                  symbolizer = sf.createLineSymbolizer(stroke,
geometryAttributeName);
                  break;

            case POINT:
                  fill = sf.createFill(ff.literal(fillColor),
ff.literal(OPACITY));

                  Mark mark = sf.getCircleMark();
                  mark.setFill(fill);
                  mark.setStroke(stroke);

                  Graphic graphic = sf.createDefaultGraphic();
                  graphic.graphicalSymbols().clear();
                  graphic.graphicalSymbols().add(mark);
                  graphic.setSize(ff.literal(POINT_SIZE));

                  symbolizer = sf.createPointSymbolizer(graphic,
geometryAttributeName);
            }

            Rule rule = sf.createRule();
            rule.symbolizers().add(symbolizer);
            return rule;
      }

      /**
       * Retrieve information about the feature geometry
       */
      private void setGeometry(FeatureSource<SimpleFeatureType,
SimpleFeature> featureSource)
      {
            GeometryDescriptor geomDesc =
featureSource.getSchema().getGeometryDescriptor();
            geometryAttributeName = geomDesc.getLocalName();

            Class<?> clazz = geomDesc.getType().getBinding();

            if (Polygon.class.isAssignableFrom(clazz) ||
MultiPolygon.class.isAssignableFrom(clazz))
```

```
            {
                  geometryType = GeomType.POLYGON;

            }
            else if (LineString.class.isAssignableFrom(clazz) ||
MultiLineString.class.isAssignableFrom(clazz))
            {

                  geometryType = GeomType.LINE;

            }
            else
            {
                  geometryType = GeomType.POINT;
            }

      }

      private Style createRGBStyle()
      {
            GridCoverage2D cov = null;
            try
            {
                  cov = reader.read(null);
            }
            catch (IOException giveUp)
            {
                  throw new RuntimeException(giveUp);
            }
            // We need at least three bands to create an RGB
style
            int numBands = cov.getNumSampleDimensions();
            if (numBands < 3)
            {
                  return null;
            }
            // Get the names of the bands
            String[] sampleDimensionNames = new String[numBands];
            for (int i = 0; i < numBands; i++)
            {
                  GridSampleDimension dim =
cov.getSampleDimension(i);
                  sampleDimensionNames[i] =
dim.getDescription().toString();
            }
            final int RED = 0, GREEN = 1, BLUE = 2;
            int[] channelNum = { -1, -1, -1 };
            // We examine the band names looking for "red...",
"green...",
            // "blue...".
            // Note that the channel numbers we record are
indexed from 1, not 0.
            for (int i = 0; i < numBands; i++)
            {
                  String name =
sampleDimensionNames[i].toLowerCase();
                  if (name != null)
```

```
                  {
                        if (name.matches("red.*"))
                        {
                              channelNum[RED] = i + 1;
                        }
                        else if (name.matches("green.*"))
                        {
                              channelNum[GREEN] = i + 1;
                        }
                        else if (name.matches("blue.*"))
                        {
                              channelNum[BLUE] = i + 1;
                        }
                  }
            }
            // If we didn't find named bands "red...",
"green...", "blue..."
            // we fall back to using the first three bands in
order
            if (channelNum[RED] < 0 || channelNum[GREEN] < 0 ||
channelNum[BLUE] < 0)
            {
                  channelNum[RED] = 1;
                  channelNum[GREEN] = 2;
                  channelNum[BLUE] = 3;
            }
            // Now we create a RasterSymbolizer using the
selected channels
            SelectedChannelType[] sct = new
SelectedChannelType[cov.getNumSampleDimensions()];
            ContrastEnhancement ce =
sf.contrastEnhancement(ff.literal(1.0),
ContrastMethod.NORMALIZE);
            for (int i = 0; i < 3; i++)
            {
                  sct[i] =
sf.createSelectedChannelType(String.valueOf(channelNum[i]), ce);
            }
            RasterSymbolizer sym =
sf.getDefaultRasterSymbolizer();
            ChannelSelection sel = sf.channelSelection(sct[RED],
sct[GREEN], sct[BLUE]);
            sym.setChannelSelection(sel);

            return SLD.wrapSymbolizers(sym);
      }
}
```

A.2 Node.Java Class

```
package org.geotools.demo.node;
```

```
import com.vividsolutions.jts.geom.Point;

public class Node
{
      public Point point;
      public double pgaValue;
}
```

A.3 Gaussian.java Class

```
package org.geotools.demo.util;

/*************************************************************
*********
 * Compilation: javac Gaussian.java Execution: java Gaussian x mu
sigma
 *
 * Function to compute the Gaussian pdf (probability density
function) and the
 * Gaussian cdf (cumulative density function)
 *
 * % java Gaussian 820 1019 209 0.17050966869132111
 *
 * % java Gaussian 1500 1019 209 0.9893164837383883
 *
 * % java Gaussian 1500 1025 231 0.9801220907365489
 *
 * The approximation is accurate to absolute error less than 8 *
10^(-16).
 * Reference: Evaluating the Normal Distribution by George
Marsaglia.
 * http://www.jstatsoft.org/v11/a04/paper
 *

*************************************************************
********/

public class Gaussian
{

      /**
       *
       * @param x
       * @return phi(x) = standard Gaussian pdf
       */
      public static double phi(double x)
      {
            return Math.exp(-x * x / 2) / Math.sqrt(2 * Math.PI);
      }

      /**
       *
       * @param z
```

```
       * @return Phi(z) = standard Gaussian cdf using Taylor
approximation
       */
      public static double Phi(double z)
      {
            if (z < -8.0)
                  return 0.0;
            if (z > 8.0)
                  return 1.0;
            double sum = 0.0, term = z;
            for (int i = 3; sum + term != sum; i += 2)
            {
                  sum = sum + term;
                  term = term * z * z / i;
            }
            return 0.5 + sum * phi(z);
      }
      //
      // // test client
      // public static void main(String[] args)
      // {
      // double z = 820; // Double.parseDouble(args[0]);
      // double mu = 1019; // Double.parseDouble(args[1]);
      // double sigma = 209; // Double.parseDouble(args[2]);
      // System.out.println(Phi(z, mu, sigma));
      // double y = Phi(z);
      // System.out.println(PhiInverse(1));
      //
      // System.out.println(Phi(11));
      // System.out.println(phi(1));
      // }

}
```

A.4 Netrel.java Class

```
package org.geotools.demo.util;

public class Netrel
{
      private static final int qsize = 2500;
      private static final int psize = 200;

      public Object[] findInterval(final int edgeNumber, final
int nodeNumber, final double[] probabilityMatrice, int[][]
adjacencyMatrice,
                  final int sourceNode, final int sinkNode)
      {
            int l;
            int front, rear;
            boolean bool = false, cont_bool;
            int[][] g, q;
            int[][] edge = new int[edgeNumber][2];
```

```
int[] dist, pred, pp;
int[] event, event1, event2;
int[][] paths;

int[] scannedEdgeNumbers = new int[edgeNumber];
int scannedEdgeNumber = 0;

int inf = 99999;
double lower = 0, upper = 0, delta = 0.000002;
int p = 0, mm = 0;
try
{
      g = new int[nodeNumber][nodeNumber];

      edge = new int[edgeNumber][2];

      q = new int[qsize][edgeNumber];

      event = new int[edgeNumber];
      event1 = new int[edgeNumber];
      event2 = new int[edgeNumber];

      dist = new int[nodeNumber];
      pred = new int[nodeNumber];

      pp = new int[edgeNumber];

      paths = new int[psize][edgeNumber];
      for (int i = 0; i < nodeNumber; i++)
      {
            for (int j = 0; j < nodeNumber; j++)
            {
                  if (adjacencyMatrice[i][j] == 0)
                        adjacencyMatrice[i][j] = inf;
                  if (adjacencyMatrice[i][j] == inf)
                        continue;
                  mm++;
                  adjacencyMatrice[i][j] = mm;
                  edge[mm - 1][0] = i + 1;
                  edge[mm - 1][1] = j + 1;
            }
      }

      front = 0;

      for (int i = 0; i < edgeNumber; i++)
            q[front][i] = 0;

      rear = 1;

      while (true)
      {
            if (rear != 1 && rear == front)
            {
                  break;
            }
            for (int i = 0; i < edgeNumber; i++)
```

```
                              event[i] = q[front][i];
                        front++;
                        if (front > qsize)
                              front = 0;
                        for (int i = 0; i < nodeNumber; i++)
                        {
                              for (int j = 0; j < nodeNumber;
j++)
                              {
                                    g[i][j] =
adjacencyMatrice[i][j];
                              }
                        }
                        for (int k = 0; k < edgeNumber; k++)
                        {
                              if (event[k] == -1)
                              {
                                    int i = edge[k][0];
                                    int j = edge[k][1];
                                    g[i - 1][j - 1] = inf;
                              }
                        }
                        Object[] result1 = bfs(g, dist, pred, pp,
nodeNumber, edgeNumber, sourceNode, sinkNode);
                        dist = (int[]) result1[0];
                        pred = (int[]) result1[1];
                        pp = (int[]) result1[2];
                        for(int xx=0;xx<pp.length;xx++)
                        {
                              scannedEdgeNumbers[xx] =
(scannedEdgeNumbers[xx]==1||pp[xx]==1)?1:0;
                        }
                        bool = (Boolean) result1[3];
                        if (!bool)
                        {
                              p++;
                              upper = probab(event, upper,
edgeNumber, probabilityMatrice);
                              if ((1 - (lower + upper)) <= delta)
                                    break;
                              else
                                    continue;
                        }
                        cont_bool = true;
                        if (p != 0)
                        {
                              cont_bool = false;
                              for (int i = 0; i < p; i++)
                                    for (int j = 0; j <
edgeNumber; j++)
                                          if (paths[i][j] ==
pp[j])
                                                cont_bool = true;
                        }
                        if (!cont_bool)
                        {
                              for (int i = 0; i < edgeNumber;
```

```
i++)
                                    paths[p - 1][i] = pp[i];
                              p++;
                        }
                        l = 0;
                        for (int i = 0; i < edgeNumber; i++)
                        {
                              event1[i] = pp[i];
                              if (event[i] != 0)
                                    event1[i] = event[i];
                              if (event[i] == 0 && event1[i] ==
1)
                                    l++;
                        }
                        lower = probab(event1, lower, edgeNumber,
probabilityMatrice);
                        Object[] result2 = comple(event, event1,
event2, l, q, rear, edgeNumber);
                        event2 = (int[]) result2[0];
                        q = (int[][]) result2[1];
                        rear = (Integer) result2[2];
                        rear++;
                  }
            }
            catch (Exception e)
            {
                  e.printStackTrace();
            }
            for(int i:scannedEdgeNumbers)
            {
                  if(i==1)
                        scannedEdgeNumber++;
            }
            int[][] resultingScannedEdges = new
int[scannedEdgeNumber][2];
            int j = 0;
            for(int i=0;i<scannedEdgeNumbers.length;i++)
            {
                  if(scannedEdgeNumbers[i]==1)
                  {
                        resultingScannedEdges[j][0] = edge[i][0]-
1;
                        resultingScannedEdges[j][1] = edge[i][1]-
1;
                        j++;
                  }
            }
            Object[] result = new Object[3];
            result[0] = resultingScannedEdges;
            result[1] = lower;
            result[2] = 1 - upper;
            return result;
      }

      // dist, pred, pp,
      public Object[] bfs(int[][] g, int[] dist, int[] pred,
int[] pp, int pointNumber, int lineNumber, int source,
```

```
                  int sink)
      {
            boolean bool = false;
            Object[] result = new Object[4];
            int inf = 99999;
            for (int i = 0; i < pointNumber; i++)
            {
                  dist[i] = inf;
                  pred[i] = -1;
            }
            dist[source - 1] = 0;
            int d = -1;
            int dd = 0;
            while (pred[sink - 1] == -1)
            {
                  d++;
                  bool = false;
                  for (int i = 0; i < pointNumber; i++)
                  {
                        if (dist[i] != d)
                              continue;
                        dd = d + 1;
                        for (int j = 0; j < pointNumber; j++)
                        {
                              if (g[i][j] == inf || dist[j] <=
dd)
                                    continue;
                              dist[j] = dd;
                              pred[j] = i + 1;
                              bool = true;
                        }
                  }
                  if (!bool)
                  {
                        result = new Object[4];
                        result[0] = dist;
                        result[1] = pred;
                        result[2] = pp;
                        result[3] = bool;
                        return result;
                  }
            }
            for (int i = 0; i < lineNumber; i++)
                  pp[i] = 0;
            int j = sink;
            while (j != source)
            {
                  int i = pred[j - 1];
                  int e = g[i - 1][j - 1];
                  pp[e - 1] = 1;
                  j = i;
            }
            result[0] = dist;
            result[1] = pred;
            result[2] = pp;
            result[3] = bool;
            return result;
```

```
	}

	public double probab(final int[] event, double r, final int
lineNumber, final double[] probabilities)
	{
		double pr = 1;
		for (int i = 0; i < lineNumber; i++)
		{
			if (event[i] == -1)
				pr = pr * (1 - probabilities[i]);
			else if (event[i] == 1)
				pr = pr * probabilities[i];
		}
		r += pr;
		return r;
	}

	// event2, q, rear degisti
	public Object[] comple(int[] event, int[] event1, int[]
event2, int l, int[][] q, int rear, int lineNumber)
	{
		for (int k = 0; k < l; k++)
		{
			int ii = -1;
			for (int i = 0; i < lineNumber; i++)
				event2[i] = event[i];
			for (int i = 0; i < lineNumber; i++)
			{
				if (event[i] != 0 || event1[i] != 1)
					continue;
				event2[i] = event1[i];
				ii++;
				if (ii != k)
					continue;
				event2[i] = -1;
				for (int j = 0; j < lineNumber; j++)
				{
					q[rear][j] = event2[j];
				}
				if (rear >= qsize)
					rear = 0;
				break;
			}
		}
		Object[] result = new Object[3];
		result[0] = event2;
		result[1] = q;
		result[2] = rear;
		return result;
	}
}
```

A.5 Util.java Class

```
package org.geotools.demo.util;

import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;

import com.vividsolutions.jts.geom.Coordinate;
import com.vividsolutions.jts.geom.Geometry;
import com.vividsolutions.jts.geom.GeometryFactory;
import com.vividsolutions.jts.geom.LineString;
import com.vividsolutions.jts.geom.LinearRing;
import com.vividsolutions.jts.geom.MultiPoint;
import com.vividsolutions.jts.geom.Point;

/**
 * Basic utility class that is used by main application
 *
 * @author gokceng
 *
 */
public class Util
{
      /**
       *
       * @param linearRingLists
       *            Whole Grid List to be scanned
       * @param coordinate
       *            a value that is to be searched in
@linearRingLists
       * @return an Object[] that includes (x,y) indexes of
linearRingLists which
       *         includes @coordinate in it
       */
      private Object[]
getIntersectionPositions(ArrayList<ArrayList<LinearRing>>
linearRingLists, Coordinate coordinate)
      {
            Object[] resultList = new Object[2];
            ArrayList<LinearRing> intersectionRow = new
ArrayList<LinearRing>();
            for (int i = 0; i < linearRingLists.size(); i++)
            {
                  ArrayList<LinearRing> linearRings =
linearRingLists.get(i);
                  double yValueOfStartingPoint =
linearRings.get(0).getCoordinateN(0).y;
                  double yValueOfEndingPoint =
linearRings.get(0).getCoordinateN(3).y;
                  if (yValueOfStartingPoint < coordinate.y &&
yValueOfEndingPoint > coordinate.y)
                  {
                        intersectionRow = linearRings;
                        resultList[1] = i;
                        break;
                  }
```

```
            }
            for (int j = 0; j < intersectionRow.size(); j++)
            {
                  LinearRing linearRing = intersectionRow.get(j);
                  double xValueOfStartingPoint =
linearRing.getCoordinateN(0).x;
                  double xValueOfEndingPoint =
linearRing.getCoordinateN(1).x;
                  if (xValueOfStartingPoint < coordinate.x &&
xValueOfEndingPoint > coordinate.x)
                  {
                        resultList[0] = j;
                        break;
                  }
            }
            return resultList;
      }

      /**
       *
       * @param linearRingLists
       *            Whole Grid List to be scanned
       * @param lineString
       *            the parameter which is used for getting end
point coordinates
       * @return an Object[][] that includes both (x,y) indexes
of linearRingLists
       *         which includes end points of @lineString in it
       */
      private Object[][]
getEndPointPositions(ArrayList<ArrayList<LinearRing>>
linearRingLists, LineString lineString)
      {
            Object[][] intersectionLinearRings = new Object[2][];
            Object[] startLinearRing =
getIntersectionPositions(linearRingLists,
lineString.getCoordinates()[0]);
            Object[] endLinearRing =
getIntersectionPositions(linearRingLists,
lineString.getCoordinates()[1]);
            intersectionLinearRings[0] = startLinearRing;
            intersectionLinearRings[1] = endLinearRing;
            return intersectionLinearRings;
      }

      /**
       *
       * @param linearRingLists
       *            Whole Grid List to be scanned
       * @param lineString
       *            The linestring object that is used for
scanning
       * @return smallest rectangular grid list that includes
@lineString
       *
       *         su an sadece cevreleyen listeyi donduruyor, bu
fonsiyonu
```

```
       *         degistirip, kesisme noktalarini donduren bir
yapi olusturulmali.
       */
      public ArrayList<LineString>
getIntersectionLinearRings(ArrayList<ArrayList<LinearRing>>
linearRingLists,
                  LineString lineString, GeometryFactory
geometryFactory, FileWriter logWriter)
      {
            Coordinate startingCoordLineString =
lineString.getCoordinates()[0];
            Coordinate endingCoordLineString =
lineString.getCoordinates()[1];

            Object[][] intersectionLinearRings =
getEndPointPositions(linearRingLists, lineString);
            int firstEndPointX = (Integer)
intersectionLinearRings[0][0];
            int firstEndPointY = (Integer)
intersectionLinearRings[0][1];

            int secondEndPointX = (Integer)
intersectionLinearRings[1][0];
            int secondEndPointY = (Integer)
intersectionLinearRings[1][1];

            int startingX = (firstEndPointX < secondEndPointX) ?
firstEndPointX : secondEndPointX;
            int endingX = (firstEndPointX > secondEndPointX) ?
firstEndPointX : secondEndPointX;
            int startingY = (firstEndPointY < secondEndPointY) ?
firstEndPointY : secondEndPointY;
            int endingY = (firstEndPointY > secondEndPointY) ?
firstEndPointY : secondEndPointY;
            ArrayList<LineString> lineStrings = new
ArrayList<LineString>();
            for (int i = startingY; i <= endingY; i++)
            {
                  for (int j = startingX; j <= endingX; j++)
                  {
                        LinearRing linearRing =
linearRingLists.get(i).get(j);
                        LineString string;
                        Coordinate firstEnd;
                        Coordinate secondEnd;
                        if (linearRing.intersects(lineString))
                        {
                              Geometry intersection =
linearRing.intersection(lineString);
                              if (intersection instanceof
MultiPoint)
                              {
                                    firstEnd = ((MultiPoint)
intersection).getCoordinates()[0];
                                    secondEnd = ((MultiPoint)
intersection).getCoordinates()[1];
                              }
```

```
                              // it means it is just a point
                              else
                              {
                                    firstEnd = ((Point)
intersection).getCoordinate();
                                    if
(firstEnd.distance(startingCoordLineString) <
firstEnd.distance(endingCoordLineString))
                                          secondEnd =
startingCoordLineString;
                                    else
                                          secondEnd =
endingCoordLineString;
                              }
                              if (secondEnd.compareTo(firstEnd)
== -1)
                                    string =
geometryFactory.createLineString(new Coordinate[] { secondEnd,
firstEnd });
                              else
                                    string =
geometryFactory.createLineString(new Coordinate[] { firstEnd,
secondEnd });

      string.setUserData(linearRing.getUserData());
                              if (logWriter != null)
                              {
                                    try
                                    {

      logWriter.write("LineSegment :-> " + string + "\n");
                                          logWriter.flush();
                                    }
                                    catch (IOException e)
                                    {
                                          e.printStackTrace();
                                    }
                              }
                              lineStrings.add(string);
                        }
                  }
            }
            return lineStrings;
      }
}
```

B. ATTRIBUTES OF CONSTRUCTED NETWORK LAYER

B.1 Attirubute table of gridded network structure

Table 5.1: Attribute table of line segments

PGA, N,15,13	FIRST_X, N,15,13	FIRST_Y, N,15,13	LAST_X ,N,15,13	LAST_Y, N,15,13	probs, N.15.13
0,35693	28,64786	40,02065	28,65110	40,04000	0,98550
0,29114	28,65110	40,04000	28,65444	40,06000	0,96897
0,38474	28,65444	40,06000	28,65779	40,08000	0,93530
0,30011	28,65779	40,08000	28,66000	40,09322	0,87304
0,36008	28,66000	40,09322	28,66113	40,10000	0,85854
0,49587	28,66113	40,10000	28,66448	40,12000	0,75309
0,25683	28,66448	40,12000	28,66782	40,14000	0,61124
0,44883	28,66782	40,14000	28,67117	40,16000	0,65482
0,29051	28,67117	40,16000	28,67451	40,18000	0,72880
0,39393	28,67451	40,18000	28,67786	40,20000	0,88177
0,43587	28,67786	40,20000	28,68000	40,21282	0,97298
0,43135	28,68000	40,21282	28,68120	40,22000	0,97717
0,29025	28,68120	40,22000	28,68287	40,23000	0,99808
0,25813	28,98589	39,87394	28,98607	39,88000	0,99997
0,26576	28,98607	39,88000	28,98668	39,90000	0,99869
0,40639	28,98668	39,90000	28,98729	39,92000	0,99899
0,26344	28,98729	39,92000	28,98789	39,94000	0,99999
0,30030	28,98789	39,94000	28,98850	39,96000	0,99997
0,35348	28,98850	39,96000	28,98911	39,98000	0,99992
0,41094	28,98911	39,98000	28,98971	40,00000	0,99991
0,32845	28,98971	40,00670	28,99032	40,02000	0,99987
0,40496	28,99032	40,02000	28,99093	40,04000	0,99958
0,34319	28,99093	40,04000	28,99154	40,06000	0,99760
0,44955	28,99154	40,06000	28,99214	40,08000	0,98457
0,26710	28,99214	40,08000	28,99275	40,10000	0,92792
0,43135	28,99275	40,10000	28,99336	40,12000	0,78138
0,34908	28,99336	40,12000	28,99396	40,14000	0,58196
0,30482	28,99396	40,14000	28,99457	40,16000	0,72629
0,34234	28,99457	40,16000	28,99518	40,18000	0,92055
0,40413	28,99518	40,18000	28,99578	40,20000	0,99118

Table 6.1 cont.

PGA, N,15,13	FIRST_X, N,15,13	FIRST_Y, N,15,13	LAST_X ,N,15,13	LAST_Y, N,15,13	probs, N.15.13
0,23953	28,99578	40,20000	28,99639	40,22000	0,99984
0,38468	28,99639	40,22000	28,99700	40,24000	0,99987
0,59031	28,99700	40,24000	28,99761	40,26000	0,99899
0,48157	28,99761	40,26000	28,99821	40,28000	0,99977
0,43164	28,99821	40,28000	28,99882	40,30000	0,98888
0,54844	28,99882	40,30000	28,99927	40,31488	0,99999
0,24555	28,98589	39,87394	28,98976	39,88000	0,99987
0,41441	28,98976	39,88000	29,00000	39,89606	0,99998
0,34407	29,00000	39,89606	29,00251	39,90000	0,99897
0,26858	29,00251	39,90000	29,01527	39,92000	0,99978
0,41883	29,01527	39,92000	29,02000	39,92742	0,99896
0,24781	29,02000	39,92742	29,02802	39,94000	0,99974
0,40261	29,02802	39,94000	40,29700	39,95878	0,99988
0,26390	40,29700	39,95878	29,04078	39,96000	0,99981
0,33633	29,04078	39,96000	29,05353	39,98000	0,00000
0,28592	29,05353	39,98000	40,35800	39,99014	0,99999
0,29572	40,35800	39,99014	29,06629	40,00000	0,99999
0,30210	29,06629	40,00000	29,07904	40,02000	0,99999
0,55094	29,07904	40,02000	40,41900	40,02150	0,99999
0,42194	40,41900	40,02150	29,09180	40,04000	0,99999
0,25475	29,09180	40,04000	40,20700	40,05286	0,99994
0,39716	40,20700	40,05286	29,10455	40,06000	0,99452
0,38364	29,10455	40,06000	29,11731	40,08000	0,63088
0,59763	29,11731	40,08000	40,54100	40,08422	0,94818
0,25246	40,54100	40,08422	29,13006	40,10000	0,99914
0,33496	29,13006	40,10000	29,14000	40,11558	0,99999
0,30054	29,14000	40,11558	29,14282	40,12000	0,99999
0,30098	29,14282	40,12000	29,15558	40,14000	0,99994
0,28585	29,15558	40,14000	29,16000	40,14694	0,99997
0,45941	29,16000	40,14694	29,16833	40,16000	0,99999
0,29572	29,16833	40,16000	29,18000	40,17830	0,99996
0,39443	29,18000	40,17830	29,18109	40,18000	0,99999
0,29581	29,18109	40,18000	29,19384	40,20000	0,99989
0,27417	29,19384	40,20000	29,20000	40,20966	0,99998
0,38422	29,20000	40,20966	29,20660	40,22000	0,99999
0,30383	29,20660	40,22000	29,21935	40,24000	0,99998
0,23581	29,21935	40,24000	29,22000	40,24102	0,99999
0,40944	29,22000	40,24102	29,23211	40,26000	0,99987
0,30869	29,23211	40,26000	29,24000	40,27237	0,99999
0,38448	29,24000	40,27237	29,24244	40,27620	0,99999
0,34113	28,98342	39,88000	28,98589	39,87394	0,99996
0,59796	28,97530	39,90000	28,98000	39,88843	0,99812
0,42504	28,98000	39,88843	28,98342	39,88000	0,99446
0,39710	28,96718	39,92000	28,97530	39,90000	0,92663

Table 6.1 cont.

PGA, N,15,13	FIRST_X, N,15,13	FIRST_Y, N,15,13	LAST_X ,N,15,13	LAST_Y, N,15,13	probs, N.15.13
0,36437	28,95906	39,94000	28,96000	39,93769	0,72100
0,42169	28,96000	39,93769	28,96718	39,92000	0,56388
0,52522	28,95094	39,96000	28,95906	39,94000	0,61471
0,47354	28,94282	39,98000	28,95094	39,96000	0,73316
0,24719	28,93470	40,00000	28,94000	39,98695	0,61636
0,36696	28,94000	39,98695	28,94282	39,98000	0,72062
0,25965	28,92658	40,02000	28,93470	40,00000	0,54648
0,24957	28,91846	40,04000	28,92000	40,03621	0,66890
0,31932	28,92000	40,03621	28,92658	40,02000	0,51110
0,31736	28,91034	40,06000	28,91846	40,04000	0,67180
0,30520	28,90222	40,08000	28,91034	40,06000	0,59668
0,36593	28,89410	40,10000	40,44900	40,08546	0,69354
0,28642	40,44900	40,08546	28,90222	40,08000	0,55836
0,29087	28,88598	40,12000	28,89410	40,10000	0,55219
0,23953	28,87786	40,14000	28,88000	40,13472	0,51893
0,31775	28,88000	40,13472	28,88598	40,12000	0,74836
0,34234	28,86974	40,16000	28,87786	40,14000	0,79482
0,43564	28,86162	40,18000	28,86974	40,16000	0,95382
0,25590	28,85349	40,20000	28,86000	40,18398	0,98070
0,43747	28,86000	40,18398	28,86162	40,18000	0,99598
0,40963	28,84809	40,21332	28,85349	40,20000	0,99920
0,32830	28,84560	40,22000	28,84809	40,21332	0,99999
0,31844	28,83816	40,24000	28,84000	40,23505	0,99991
0,31424	28,84000	40,23505	28,84560	40,22000	0,99930
0,53208	28,83072	40,26000	28,83816	40,24000	0,99522
0,30511	28,82327	40,28000	28,83072	40,26000	0,90227
0,36772	28,81583	40,30000	28,82000	40,28879	0,69919
0,27609	28,82000	40,28879	28,82327	40,28000	0,57560
0,28459	28,80839	40,32000	28,81583	40,30000	0,84568
0,30193	28,80094	40,34000	28,80839	40,32000	0,99535
0,33663	28,79350	40,36000	40,41800	40,34254	0,99987
0,26285	40,41800	40,34254	28,80094	40,34000	0,99990
0,47834	28,78606	40,38000	28,79350	40,36000	0,99971
0,31353	28,77862	40,40000	28,78000	40,39628	0,95202
0,23719	28,78000	40,39628	28,78606	40,38000	0,97185
0,33339	28,77117	40,42000	28,77862	40,40000	0,64478
0,38471	28,76451	40,43790	28,77117	40,42000	0,98127
0,42634	28,45968	40,31203	28,46000	40,31277	0,99217
0,30488	28,46000	40,31277	28,46315	40,32000	0,97029
0,31287	28,46315	40,32000	28,47188	40,34000	0,99979
0,26311	28,47188	40,34000	28,48000	40,35863	0,99987
0,32232	28,48000	40,35863	28,48060	40,36000	0,99999
0,43440	28,48060	40,36000	28,48932	40,38000	0,99999
0,29867	28,48932	40,38000	28,49805	40,40000	0,99999

Table 6.1 cont.

PGA, N,15,13	FIRST_X, N,15,13	FIRST_Y, N,15,13	LAST_X ,N,15,13	LAST_Y, N,15,13	probs, N.15.13
0,29737	28,49805	40,40000	40,32600	40,40448	0,99989
0,42259	40,32600	40,40448	28,50677	40,42000	0,99992
0,32831	28,50677	40,42000	28,51549	40,44000	0,89802
0,26259	28,51549	40,44000	28,52000	40,45034	0,90506
0,45902	28,52000	40,45034	28,52422	40,46000	0,95834
0,34155	28,52422	40,46000	28,53294	40,48000	0,99993
0,44514	28,53294	40,48000	28,54000	40,49619	0,99999
0,29719	28,54000	40,49619	28,54166	40,50000	0,99999
0,27638	28,54166	40,50000	28,55038	40,52000	0,99999
0,39938	28,55038	40,52000	28,55911	40,54000	0,99999
0,32987	28,55911	40,54000	28,56000	40,54205	0,99999
0,32285	28,56000	40,54205	28,56595	40,55569	0,99999
0,35670	28,67928	40,24000	28,68000	40,23800	0,99986
0,27685	28,68000	40,23800	28,68287	40,23000	0,99999
0,45420	28,67210	40,26000	28,67928	40,24000	0,99999
0,52801	28,66492	40,28000	28,67210	40,26000	0,99116
0,36523	28,65774	40,30000	28,66000	40,29372	0,71623
0,33376	28,66000	40,29372	28,66492	40,28000	0,65981
0,37065	28,65056	40,32000	28,65774	40,30000	0,99996
0,31046	28,64338	40,34000	28,65056	40,32000	0,99999
0,29637	28,63620	40,36000	28,64000	40,34943	0,99999
0,31806	28,64000	40,34943	28,64338	40,34000	0,99999
0,36605	28,62902	40,38000	28,63620	40,36000	0,99999
0,29008	28,62184	40,40000	28,62902	40,38000	0,99999
0,25885	28,61466	40,42000	28,62000	40,40514	0,99996
0,24154	28,62000	40,40514	28,62184	40,40000	0,99974
0,44154	28,60748	40,44000	28,61466	40,42000	0,81741
0,31363	28,60030	40,46000	28,60748	40,44000	0,93735
0,28767	28,59313	40,48000	40,35700	40,46085	0,99893
0,45902	40,35700	40,46085	28,60030	40,46000	0,99960
0,31652	28,58595	40,50000	28,59313	40,48000	0,99999
0,30637	28,57877	40,52000	28,58000	40,51656	0,99999
0,41702	28,58000	40,51656	28,58595	40,50000	0,99999
0,45902	28,57159	40,54000	28,57877	40,52000	0,99999
0,29479	28,56595	40,55569	28,57159	40,54000	0,99999
0,32870	28,73041	40,78000	28,74000	40,77529	0,99999
0,49554	28,74000	40,77529	28,75063	40,77007	0,99986
0,31044	28,68969	40,80000	40,38700	40,79494	0,79736
0,30248	40,38700	40,79494	28,72000	40,78511	0,70719
0,26833	28,72000	40,78511	28,73041	40,78000	0,76413
0,39692	28,64897	40,82000	28,66000	40,81458	0,99999
0,31980	28,66000	40,81458	28,68000	40,80476	0,99999
0,41050	28,68000	40,80476	28,68969	40,80000	0,99999
0,25370	28,60825	40,84000	28,62000	40,83423	0,99999

Table 6.1 cont.

PGA, N,15,13	FIRST_X, N,15,13	FIRST_Y, N,15,13	LAST_X ,N,15,13	LAST_Y, N,15,13	probs, N.15.13
0,23593	28,62000	40,83423	28,64000	40,82441	0,99999
0,39895	28,64000	40,82441	28,64897	40,82000	0,99999
0,31613	28,56753	40,86000	28,58000	40,85388	0,99999
0,41641	28,58000	40,85388	40,35700	40,84405	0,99999
0,39332	40,35700	40,84405	28,60825	40,84000	0,99999
0,28839	28,52682	40,88000	28,54000	40,87352	0,99999
0,40310	28,54000	40,87352	28,56000	40,86370	0,99999
0,30733	28,56000	40,86370	28,56753	40,86000	0,99999
0,28355	28,48610	40,90000	40,32600	40,89317	0,98893
0,49416	40,32600	40,89317	28,52000	40,88335	0,99367
0,30087	28,52000	40,88335	28,52682	40,88000	0,99352
0,38695	28,44538	40,92000	28,46000	40,91282	0,99999
0,32205	28,46000	40,91282	28,48000	40,90299	0,99999
0,48302	28,48000	40,90299	28,48610	40,90000	0,99999
0,32885	28,40466	40,94000	28,42000	40,93247	0,99999
0,26061	28,42000	40,93247	28,44000	40,92264	0,99999
0,46923	28,44000	40,92264	28,44538	40,92000	0,99999
0,36282	28,38306	40,95061	40,29600	40,94229	0,99999
0,24347	40,29600	40,94229	28,40466	40,94000	0,99999
0,25532	28,63537	40,04000	28,64000	40,03283	0,99999
0,26409	28,64000	40,03283	28,64786	40,02065	0,99999
0,47159	28,62245	40,06000	28,63537	40,04000	0,99999
0,30850	28,60953	40,08000	28,62000	40,06379	0,99999
0,30833	28,62000	40,06379	28,62245	40,06000	0,99999
0,31806	28,59662	40,10000	40,35700	40,09476	0,99999
0,24771	40,35700	40,09476	28,60953	40,08000	0,99998
0,44250	28,58370	40,12000	28,59662	40,10000	0,99989
0,42428	28,57078	40,14000	28,58000	40,12573	0,99951
0,31091	28,58000	40,12573	28,58370	40,12000	0,99933
0,23953	28,55786	40,16000	28,56000	40,15669	0,99781
0,38283	28,56000	40,15669	28,57078	40,14000	0,99732
0,37222	28,54495	40,18000	28,55786	40,16000	0,98643
0,35977	28,53203	40,20000	28,54000	40,18766	0,90951
0,46879	28,54000	40,18766	28,54495	40,18000	0,91465
0,36633	28,51911	40,22000	28,52000	40,21863	0,57990
0,33865	28,52000	40,21863	28,53203	40,20000	0,58504
0,37627	28,50620	40,24000	28,51911	40,22000	0,90930
0,31562	28,49328	40,26000	40,32600	40,24959	0,99950
0,47045	40,32600	40,24959	28,50620	40,24000	0,99865
0,34586	28,48036	40,28000	28,49328	40,26000	0,99999
0,30071	28,46744	40,30000	28,48000	40,28056	0,99999
0,33536	28,48000	40,28056	28,48036	40,28000	0,99999
0,42400	28,45968	40,31203	28,46000	40,31153	0,99999
0,34026	28,46000	40,31153	28,46744	40,30000	0,99999

Table 6.1 cont.

PGA, N,15,13	FIRST_X, N,15,13	FIRST_Y, N,15,13	LAST_X ,N,15,13	LAST_Y, N,15,13	probs, N.15.13
0,47062	28,98949	40,32000	28,99927	40,31488	0,99999
0,31168	28,95133	40,34000	28,96000	40,33546	0,99873
0,31563	28,96000	40,33546	28,98000	40,32498	0,99997
0,25198	28,98000	40,32498	28,98949	40,32000	0,99999
0,28029	28,91316	40,36000	28,92000	40,35642	0,89053
0,40261	28,92000	40,35642	28,94000	40,34594	0,97251
0,32622	28,94000	40,34594	28,95133	40,34000	0,99800
0,45802	28,87500	40,38000	28,88000	40,37738	0,70163
0,31502	28,88000	40,37738	40,44900	40,36690	0,71509
0,24268	40,44900	40,36690	28,91316	40,36000	0,74327
0,30248	28,83683	40,40000	28,84000	40,39834	0,83176
0,29243	28,84000	40,39834	28,86000	40,38786	0,76856
0,44653	28,86000	40,38786	28,87500	40,38000	0,74538
0,42225	28,79867	40,42000	40,41800	40,41930	0,99992
0,36146	40,41800	40,41930	28,82000	40,40882	0,99966
0,41975	28,82000	40,40882	28,83683	40,40000	0,99874
0,36540	28,76451	40,43790	28,78000	40,42978	0,99999
0,41125	28,78000	40,42978	28,79867	40,42000	0,99999
0,35113	28,68287	40,23000	28,68385	40,24000	0,99999
0,29539	28,68385	40,24000	28,68579	40,26000	0,99994
0,23833	28,68579	40,26000	28,68774	40,28000	0,99142
0,30898	28,68774	40,28000	28,68969	40,30000	0,67856
0,47780	28,68969	40,30000	28,69163	40,32000	0,96977
0,49440	28,69163	40,32000	28,69358	40,34000	0,99998
0,26866	28,69358	40,34000	28,69552	40,36000	0,99999
0,29572	28,69552	40,36000	28,69747	40,38000	0,99999
0,31654	28,69747	40,38000	28,69942	40,40000	0,99814
0,46872	28,69942	40,40000	40,38700	40,40599	0,75913
0,34836	40,38700	40,40599	28,70136	40,42000	0,69871
0,38458	28,70136	40,42000	28,70331	40,44000	0,93577
0,24680	28,70331	40,44000	28,70526	40,46000	0,99774
0,41441	28,70526	40,46000	28,70588	40,46641	0,99991
0,25098	28,70588	40,46641	28,70788	40,48000	0,99994
0,34234	28,70788	40,48000	28,71083	40,50000	0,99999
0,25330	28,71083	40,50000	28,71378	40,52000	0,99999
0,47297	28,71378	40,52000	28,71672	40,54000	0,99999
0,32369	28,71672	40,54000	28,71967	40,56000	0,99999
0,23896	28,71967	40,56000	28,72000	40,56224	0,99999
0,37699	28,72000	40,56224	28,72262	40,58000	0,99989
0,39468	28,72262	40,58000	28,72556	40,60000	0,99742
0,25136	28,72556	40,60000	28,72851	40,62000	0,95187
0,30248	28,72851	40,62000	28,73146	40,64000	0,68880
0,31518	28,73146	40,64000	28,73441	40,66000	0,78346
0,32463	28,73441	40,66000	28,73735	40,68000	0,96026

Table 6.1 cont.

PGA, N,15,13	FIRST_X, N,15,13	FIRST_Y, N,15,13	LAST_X ,N,15,13	LAST_Y, N,15,13	probs, N.15.13
0,26710	28,73735	40,68000	28,74000	40,69797	0,99433
0,26710	28,74000	40,69797	28,74030	40,70000	0,99996
0,45130	28,74030	40,70000	28,74325	40,72000	0,99999
0,36947	28,74325	40,72000	28,74619	40,74000	0,99999
0,25475	28,74619	40,74000	28,74914	40,76000	0,99999
0,42466	28,74914	40,76000	28,75063	40,77007	0,99999
0,31214	28,97191	39,88000	28,98000	39,87649	0,55638
0,25507	28,98000	39,87649	28,98589	39,87394	0,63955
0,33648	28,92584	39,90000	28,94000	39,89385	0,72383
0,40496	28,94000	39,89385	28,96000	39,88517	0,84742
0,26390	28,96000	39,88517	28,97191	39,88000	0,91940
0,29140	28,87976	39,92000	28,88000	39,91989	0,66079
0,25525	28,88000	39,91989	40,44900	39,91121	0,81850
0,39425	40,44900	39,91121	28,92000	39,90253	0,92245
0,36442	28,92000	39,90253	28,92584	39,90000	0,97400
0,32079	28,83368	39,94000	28,84000	39,93726	0,65312
0,27023	28,84000	39,93726	28,86000	39,92858	0,84568
0,28530	28,86000	39,92858	28,87976	39,92000	0,94657
0,29534	28,78760	39,96000	40,41800	39,95462	0,59211
0,58218	40,41800	39,95462	28,82000	39,94594	0,68755
0,31736	28,82000	39,94594	28,83368	39,94000	0,89455
0,31736	28,74152	39,98000	28,76000	39,97198	0,84595
0,30558	28,76000	39,97198	28,78000	39,96330	0,67215
0,24281	28,78000	39,96330	28,78760	39,96000	0,59375
0,28130	28,69544	40,00000	40,38700	39,99802	0,96424
0,39716	40,38700	39,99802	28,72000	39,98934	0,93507
0,38982	28,72000	39,98934	28,74000	39,98066	0,87467
0,29171	28,74000	39,98066	28,74152	39,98000	0,75534
0,55395	28,64936	40,02000	28,66000	40,01538	0,96319
0,25960	28,66000	40,01538	28,68000	40,00670	0,95005
0,29247	28,68000	40,00670	28,69544	40,00000	0,92546
0,38430	28,64786	40,02065	28,64936	40,02000	0,91585
0,40709	28,76443	40,44000	28,76451	40,43790	0,95485
0,39632	28,76359	40,46000	28,76443	40,44000	0,99885
0,31888	28,76275	40,48000	28,76359	40,46000	0,99996
0,45656	28,76192	40,50000	28,76275	40,48000	0,99999
0,30688	28,76108	40,52000	28,76192	40,50000	0,99999
0,28988	28,76024	40,54000	28,76108	40,52000	0,99999
0,30022	28,75941	40,56000	28,76000	40,54584	0,99999
0,46916	28,76000	40,54584	28,76024	40,54000	0,99993
0,34493	28,75857	40,58000	28,75941	40,56000	0,99969
0,31581	28,75774	40,60000	28,75857	40,58000	0,98869
0,29506	28,75690	40,62000	28,75774	40,60000	0,79624
0,30634	28,75606	40,64000	28,75690	40,62000	0,80218

Table 6.1 cont.

PGA, N,15,13	FIRST_X, N,15,13	FIRST_Y, N,15,13	LAST_X ,N,15,13	LAST_Y, N,15,13	probs, N.15.13
0,36825	28,75523	40,66000	28,75606	40,64000	0,98784
0,35807	28,75439	40,68000	28,75523	40,66000	0,99947
0,45867	28,75355	40,70000	28,75439	40,68000	0,99994
0,59664	28,75272	40,72000	28,75355	40,70000	0,99999
0,28764	28,75188	40,74000	28,75272	40,72000	0,99999
0,39938	28,75105	40,76000	28,75188	40,74000	0,99999
0,25475	28,75063	40,77007	28,75105	40,76000	0,99999
0,39029	28,68459	40,48000	40,38700	40,47017	0,83186
0,44883	40,38700	40,47017	28,70588	40,46641	0,80995
0,36096	28,65324	40,50000	28,66000	40,49569	0,66927
0,30964	28,66000	40,49569	28,68000	40,48293	0,64737
0,25475	28,68000	40,48293	28,68459	40,48000	0,63458
0,44637	28,62189	40,52000	28,64000	40,50845	0,57549
0,23881	28,64000	40,50845	28,65324	40,50000	0,52819
0,26055	28,59055	40,54000	40,35700	40,53397	0,67650
0,35122	40,35700	40,53397	28,62000	40,52121	0,67350
0,36005	28,62000	40,52121	28,62189	40,52000	0,65966
0,30023	28,56595	40,55569	28,58000	40,54673	0,78760
0,58619	28,58000	40,54673	28,59055	40,54000	0,77262