NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

MODELING DATA ENCAPSULATION AND A COMMUNICATION NETWORK FOR THE NATIONAL TRAINING CENTER, FORT IRWIN, CA.

by

Kirk C. Benson

March 1997

Thesis Advisor: Samuel H. Parry

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1997	Master's Thesis

4. TITLE AND SUBTITLE
MODELING DATA ENCAPSULATION AND A COMMUNICATION NETWORK FOR THE NATIONAL TRAINING CENTER, FORT IRWIN, CA.

5. FUNDING NUMBERS

6. AUTHOR(S)
Benson, Kirk C.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release, distribution is unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

The National Training Center (NTC) located at Fort Irwin, California provides the U.S. Army sole replication of a desert combat environment. The NTC provides U.S. Army brigade size heavy forces both realistic training scenarios and an accurate record of mission execution. The primary emphasis of this research is to develop the methodology for modeling both data encapsulation and transmission via a fiber optic cable for the NTC. To capitalize on technological advances, the NTC requires a relational database for data encapsulation. The database structure in this effort efficiently stores Range Data Management System (RDMS) and Observer/Controller (OC) data input. The NTC also requires a mathematical modeling (network) tool with the capability of flexible analysis of a modular fiber optic cable system. The NTC Route Optimizer program developed in this effort provides a tool for rapid manipulation of design factors with immediate graphical and numerical feedback. Additionally, the reader is given methods to design future upgrades to the database and change specifications of the fiber optic cable system. This allows the reader to manipulate technology for specific goals instead of receiving transparent improvements that are disconnected.

14. SUBJECT TERMS	15. NUMBER OF PAGES
National Training Center (NTC), Relational Database, Network	130
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18
298-102

Approved for public release; distribution is unlimited.

MODELING DATA ENCAPSULATION AND A COMMUNICATION NETWORK FOR THE NATIONAL TRAINING CENTER, FORT IRWIN, CA.

Kirk C. Benson
Captain, United States Army
B.S., United States Military Academy, 1986

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
March 1997

Author: _____
Kirk C. Benson

Approved by: _____
Samuel H. Parry, Thesis Advisor

_____
Charles H. Shaw III, Second Reader

_____
Frank C. Petho, Chairman
Department of Operations Research

ABSTRACT

The National Training Center (NTC) located at Fort Irwin, California provides the U.S. Army sole replication of a desert combat environment. The NTC provides U.S. Army brigade size heavy forces both realistic training scenarios and an accurate record of mission execution. The primary emphasis of this research is to develop the methodology for modeling both data encapsulation and transmission via a fiber optic cable for the NTC. To capitalize on technological advances, the NTC requires a relational database for data encapsulation. The database structure in this effort efficiently stores Range Data Management System (RDMS) and Observer/Controller (OC) data input. The NTC also requires a mathematical modeling (network) tool with the capability of flexible analysis of a modular fiber optic cable system. The NTC Route Optimizer program developed in this effort provides a tool for rapid manipulation of design factors with immediate graphical and numerical feedback. Additionally, the reader is given methods to design future upgrades to the database and change specifications of the fiber optic cable system. This allows the reader to manipulate technology for specific goals instead of receiving transparent improvements that are disconnected.

THESIS DISCLAIMER

The reader is cautioned that computer programs developed in this research may not have been exercised for all cases of interest. While every effort has been made, within the time available, to ensure that the programs are free of computational and logic errors, they cannot be considered validated. Any application of these programs without additional verification is at the risk of the user.

TABLE OF CONTENTS

EXECUTIVE SUMMARY

The National Training Center (NTC) located at Fort Irwin, California provides the U.S. Army sole replication of a desert combat environment. Extensively developed over the last 20 years, the NTC provides U.S. Army brigade size heavy forces both realistic training scenarios and an accurate record of mission execution. The primary emphasis of this research is to develop the methodology for modeling both data encapsulation and transmission via a fiber optic cable for the NTC. To capitalize on technological advances, the NTC requires a relational database for data encapsulation. The NTC also requires a mathematical modeling (network) tool with the capability of flexible analysis of a modular fiber optic cable system. Additionally, the reader is given methods to design future upgrades to the database and change specifications of the fiber optic cable system.

The database structure presented in this effort efficiently stores Range Data Management System (RDMS) and Observer/Controller (OC) data input. This database structure facilitates modular implementation and reflects simplicity. Previous efforts often combined subjective and objective data with complex input requirements as a byproduct. The first step in designing a database entails development of semantic objects modeling the NTC environment. The database obtained from these semantic objects accommodates administrative, objective, and subjective data encapsulation. In general, input of administrative data occurs either at the start of a rotation or during change of mission. Objective and subjective data inputs are event driven. Next, the logical schema or structure of the database relations is produced through normalization aided by computer modeling tools. At this point reports representing meaningful relationships among data can be drawn from the database. The focus of this methodology lies in structuring both administrative and objective data into a tractable database using a bottom-up perspective. Additionally, Measures of Effectiveness (MOE) relevant to the

unique NTC environment are developed with supporting Structured Query Language (SQL).

The fiber optic cable system must support current data transmissions and be adaptable to future requirements. Given a foundation in network modeling concepts in this thesis, the reader is shown the development of a tool for the NTC capable of comparing alternative communication networks. This tool facilitates simple manipulation of user inputs that specify network requirements and terrain characteristics. The goal of these inputs is to accurately model the terrain that separates demand locations. Demand locations or nodes denote a user requirement for a link into the communication system or a forced routing of the cable. At this point, the tool calculates the cheapest set of paths or routes that connect all demand nodes. This thesis details the development and use of a computer application which provides these capabilities. The name of this application is the NTC Route Optimizer. The NTC Route Optimizer program provides a tool for rapid manipulation of cable design factors with immediate graphical and numerical feedback. A discussion of user inputs and subsequent outputs is also given.

The strength of these methodologies lies in the long term direction they provide to the NTC. Data encapsulation and transmission techniques are explosive in terms of technological improvement. New and improved software and hardware will introduce great capability, yet the core concept of data encapsulation and transmission must remain simple and robust. This will allow the user to manipulate technology for specific goals instead of receiving transparent improvements that are disconnected.

I. INTRODUCTION

A. GENERAL

The National Training Center (NTC) located at Fort Irwin, California provides the U.S. Army sole replication of a desert combat environment. Extensively developed over the last 20 years, the NTC provides U.S. Army brigade size heavy forces both realistic training scenarios and an accurate record of mission execution. Training exercises include large scale force-on-force battles and live fire training. Brigade elements consisting of combined arms task forces engage a permanent opposing force (OPFOR) of equivalent size. OPFOR units employ Soviet-style equipment and can replicate a wide range of possible regional threats. In support of the training, the Range Data Measurement Subsystem (RDMS) is a state of the art instrumentation system that records objective data at player level for each unit rotation. At the NTC a player connotes a vehicle or an individual soldier. These data, coupled with subjective input from eight hundred full-time combat trainers who observe and control units (OC), form the basis for effective critiques of unit performance.

B. PROBLEM DESCRIPTION

The instrumentation system at the NTC feeds data into an antiquated Ingres database that has been both extensively and exclusively modified for use by NTC Tactical Analysis and Feedback (TAF) center analysts. Utilization of the database provides graphical representation of mission execution but little, if any, objective analysis. All data currently used in After Action Reviews (AAR) are manually collected by OCs, not by the executing queries to the database. Also, the current version of Data Base Management Software (DBMS) is not maintained by the manufacturer. All DBMS modifications are performed under a Cost Plus Contract with the Hughes Corporation at significant expense to the government.

Upgrades to the RDMS necessary to accommodate the introduction of the Multiple Integrated Laser Engagement System II (MILES II), Simulated Area Weapons Effects (SAWE), Mine Effects Simulator (MES), and the Air-Ground Engagement System (AGES) render the current objective analysis system ineffective. The current database structure is unable to accommodate both the increase and change in data inputs. Another factor influencing the situation is that the NTC will introduce a modern off-the-shelf computer operating system (Solaris) incorporating a new database application (Oracle) in early 1997.

Additionally, the NTC faces the loss of frequency bandwidths. As early as 2003, U.S. Government frequency auctions will deny NTC use of several bandwidths that are essential for training. The NTC currently utilizes a broad portion of the frequency spectrum to transmit instrumentation data. Repeater antennas on Tiefort and Granite Mountains provide coverage over 95% of the training area. Current technology affected by the loss of these bandwidths include microwave communications between the "Star Wars" building and both field AAR sites and Mobile Video Units (MVU). The "Star Wars" building is the information hub for all storage and manipulation of instrumentation data. AAR sites encompass sophisticated feedback of the recent battle and the ability to record the actual AAR event. MVUs provide audio/video support of actual battles from various locations. Additional subsystems at NTC inhibited by the lack of bandwidths or inability to transmit data include:

1. Control system for live fire targets.
2. Automated AARs at Company/Team level. Currently, they are only given at Task Force level.
3. Wireless or plug-in Local Area Network (LAN) capability.
4. Land expansion instrumentation support. NTC will acquire an additional 500 square miles in the near future.

C. SCOPE OF THESIS

The primary emphasis of this research is to develop the methodology for modeling both data encapsulation and transmission via a fiber optic cable for the NTC.

To capitalize on technological advances, the NTC requires a relational database for data encapsulation. The NTC also requires a mathematical modeling (network) tool with the capability of flexible analysis of a modular fiber optic cable system. This effort provides an initial solution to both problems.

The database structure presented in this effort will efficiently store RDMS and OC data input. This database structure facilitates modular implementation and reflects simplicity. Previous efforts often combined subjective and objective data with complex input requirements as a byproduct. The first step in designing a database entails development of semantic objects modeling the NTC environment. The database obtained from these semantic objects accommodates administrative, objective, and subjective data encapsulation. In general, input of administrative data occurs either at the start of a rotation or during change of mission. Objective and subjective data inputs are event driven. Next, the logical schema or structure of the database relations is produced through normalization aided by computer modeling tools. At this point reports representing meaningful relationships among data can be drawn from the database. The focus of this methodology lies in structuring both administrative and objective data into a tractable database using a bottom-up perspective. Concurrent efforts in a Naval Postgraduate School thesis titled *A Methodology for Modeling Subjective Data Encapsulation at the National Training Center, Fort Irwin, CA.* address subjective data encapsulation [Ref. 1]. Additionally, Measures of Effectiveness (MOE) relevant to the unique NTC environment are developed with supporting Structured Query Language (SQL).

The fiber optic cable system must support current data transmissions and be adaptable to future requirements. Given a foundation in network modeling concepts in this thesis, the reader is shown the development of a tool for the NTC capable of comparing alternative communication networks. This tool facilitates simple manipulation of user inputs that specify network requirements and terrain characteristics. The goal of these inputs is to accurately model the terrain that separates demand

locations. Demand locations or nodes denote a user requirement for a link into the communication system or a forced routing of the cable. At this point, the tool calculates the cheapest set of paths or routes that connect all demand nodes. This thesis details the development and use of a computer application that provides this capability. The name of this application is the NTC Route Optimizer. The NTC Route Optimizer program provides a tool for rapid manipulation of cable design factors with immediate graphical and numerical feedback. Discussion of user inputs and subsequent outputs is also given. The endpoint of the model is the capability to do cost comparisons on multiple fiber optic cable system designs.

D. SUMMARY OF CONTENTS

This thesis consists of seven chapters with the intent of giving the reader a thorough understanding of the database design and network optimization. Additionally, the reader is given methods to both design future upgrades to the database and change specifications of the fiber optic cable system. This chapter presents the purpose, problem description, and scope of the thesis.

Chapter II details pertinent background information on this effort. Chapter III includes a literature review that describes database design techniques. As an aid to the reader, it also presents simplified examples of current data modeling techniques. Additionally, terms and definitions relevant to this research are defined and discussed in this chapter. Chapter IV provides the methodology for modeling data encapsulation for the NTC. It provides the relational structure for current data encapsulation. Also, the reader is presented with sample analytic representations of manipulated data. Appendix A contains relations typifying rotational data. Chapter V provides the methodology for modeling the NTC data transmission network. Explanations are provided for terrain representation. Shortest path and minimum spanning tree algorithms are also presented in this section. Chapter VI details both the capability and a sample solution produced from a Windows based program designed for this particular problem. Additionally, an estimate for future fiber optic cable design is shown. Appendix B details user inputs to

the program. Appendix C explains a sample session of the NTC Route Optimizer. Appendix D contains source code of the application. Chapter VII discusses the conclusions of this research. It also provides recommendations for the future improvements to the database and the fiber optic cable system.

II. BACKGROUND

Twelve times a year, Army units stationed in the continental United States (CONUS) go to Fort Irwin, CA for NTC rotations. Each rotation brings 3,500 to 5,000 soldiers representing major combat, combat support, and combat service support elements of a U.S. Army brigade. A typical rotation lasts twenty-four days and involves several days of equipment issue followed by fourteen days of intensive force-on-force and live-fire training. Platoon to brigade level After Action Reviews (AAR) focus on cause and effect. For each engagement, soldiers and leaders assess what happened, why it happened, and determine how to improve their battlefield skills. The units then spend time performing maintenance, turning in equipment, and returning to their home station.

The purpose of training at the NTC is to identify areas where battalion task forces and brigade staffs need improvement. The goal of Observer/Controllers (OC) is to assist the rotating unit in that purpose by providing objective and subjective observations on all training missions. To assist in the collection of objective observations, the U.S. Army invests millions of dollars instrumenting player vehicles and personnel to provide accurate feedback. Vehicles and personnel at the NTC are equipped with the Multiple Integrated Laser Engagement System II (MILES II) which is an eye-safe laser system that simulates combat engagements. MILES II allows one combat system to "kill" another system through the emission of a laser beam. A laser detector belt fitted on each vehicle measures the strength of incoming laser beams. If the beam was emitted by a combat system that is capable of killing it and within its maximum effective range, then a hit is registered. At this point the MILES II system randomly selects an outcome of the engagement with the assistance of pre-determined probabilities of kill. One of six outcomes is possible: near-miss, hit, catastrophic kill, communications kill, mobility kill, or a firepower kill. The previous system, MILES I, simply returned a near miss or a kill. Under the old system, when a vehicle was killed, the OC would assess the type of kill and would dictate whether or not the vehicle could continue in the battle.

In addition to MILES II, the NTC mounts several other instrumentation devices on vehicles. A Global Position Satellite (GPS) receiver records the location of the vehicle on the battlefield. The Simulated Area Weapons Effects (SAWE) receiver replicates the effect of indirect fire and chemical munition strikes. The Mine Effects Simulator (MES) receiver simulates the effects of damage sustained from minefields. The Air Ground Engagement System (AGES) simulates close air support and maneuver force conflict. Three additional instrumentation systems measure the hull to turret angle, the type of ammunition selected for engagements, and the number of rounds of ammunition by type currently on the vehicle.

All of these systems feed information into a central processor located on each vehicle called the Data Communications Interface (DCI). The DCI provides a bi-directional data link via a Radio Relay Subsystem (RRS). The DCI transmits its data upon the occurrence of an event. Specifically, DCIs report to the central node when there is a change in event status, predetermined movement distance, or at selected time intervals. DCIs vary in technical specifications depending on individual, vehicle, or rotary wing utilization. Reporting rates vary by platform with a maximum of one per 45 seconds for individuals, one per five seconds for vehicles, and two per second for rotary wing aircraft. Additionally, DCIs have a Built In Test (BIT) capability. The RRS provides bi-directional linkage between DCIs and the central node. It has the capacity of 6,000 events per minute with a maximum peak rate of 120 events per second. Over 95% of the training area has coverage [Ref. 2].

The central node is the hardware and software subsystem that links the RSS to the Core Instrumentation System (CIS) or “brain” located in the main post area. Once the event reaches the central node, two functions occur. First, data are buffered for 12 hours to provide system redundancy. Second, data are formatted and transmitted via a Local Area Network (LAN) for CIS processing within five seconds of occurrence. The CIS takes the information received from the central node to create a computerized picture of the battlefield that displays vehicles moving, vehicles firing, and vehicles being engaged

by other vehicles. This animated war is superimposed on a computerized terrain map of the NTC that includes operational graphics of the training or rotational unit and manual inputs which allow minefields, chemical strikes, and artillery fire missions to be displayed almost as soon as they occur during the battle. The CIS stores the raw data received from the central node into a database for the purpose of reports generation and the archiving of information for further analysis. Additionally, analysts process data in the CIS for real time critique of unit performance. Scheduled upgrades to the RDMS will link and control 2,000 (with a planned expansion to 4,000) players.

III. DATABASE THEORY

A. CONCEPT OF DESIGN

There are several methods for designing a database system. Generally, a user starts with the simple goal of storing and retrieving data which evolves into a complex application of computer science and logic. A successful database design incorporates user requirements into a flexible architecture that allows for both efficient execution and future modification. The end product is a self-describing collection of integrated records. The database designer can use a variety of approaches to the problem including the entity-relationship model and the semantic object model. Both approaches allow a designer to model a user's perspective of reality. The focus of this section is on both the goals of database design and highlighting prevalent methods.

Maciaszek presents the concept of "lifecycle" considerations in database design [Ref. 3]. Specifically, the process of designing a database must be iterative over the life of the system. The first phase of design entails analysis of user requirements. Basically, the designer must define the data model and identify the scope of the database. Next, the designer conceptualizes relationships between data elements. This phase is independent of actual software applications and technical implementation considerations. Then, logical schema or the structure of the database is derived from these relationships. At this point, the designer must implement the logical schema on a particular hardware platform with a software application. Popular software applications include ORACLE, FOXBASE, and ACCESS. Next, the designer programs user applications which include tables for data input and queries that generate reports for data retrieval. Lastly, the database system must be continually tested and maintained. Through data audits and periodic upgrades, the database can continue to be effective.

The most important goal of the database requirements phase is creating an accurate model of the users' data [Ref. 4, p. 53]. The designer can approach this problem

either from a top-down or bottom-up perspective. The top-down strategy involves data modeling from the general to the specific. This strategy allows an organization to set broad data manipulation goals. Then, the organization can develop information requirements to reach these goals. The top-down strategy is particularly effective in designing databases with a global view. The bottom-up perspective operates in the opposite direction. Abstraction begins at the specific and culminates in the representation of relationships. For example, a designer may take available data and develop measures of effectiveness for the organization. This strategy can be implemented relatively fast, but may entail future rework and modification to adjust the system to a global view.

A true approach to modeling user data began when P.P. Chen introduced the entity-relationship model (E-R model) in 1976 [Ref 4]. Based on mathematical set theory, this model provided a language for expressing data and relationships from the users' environment. The basic component of the model is an entity which is an object from the users' world. Each entity has attributes that describe its characteristics. Instances denote a particular entity. Developing relationships or associations between entities provides utility to the database. At this point the normalization process occurs to transform the model into logical schema. This step is both difficult and critical for platform implementation. The E-R model is very effective for the top-down modeling approach.

As an alternative, Kroenke first presented the semantic object model in 1988. E.F. Codd and others originally developed the concept in the 1970s. This model provides a different perspective on modeling the users' environment. The basic component of this method is the semantic object. The formal definition explains the semantic object as a "named collection of attributes that sufficiently describes a distinct identity." [Ref. 4, p. 80] Semantic objects model the users' requirements more realistically than the E-R model. Another distinct advantage of the semantic object model is the introduction of software that transforms models into logical physical schema. The Wall Data

Corporation produces an excellent computer application to develop and validate semantic object models called SALSA. The schema produced by SALSA is compliant with industry norms which provides versatility. Specifically, it is compatible with multiple hardware and software platforms.

B. SEMANTIC OBJECT MODEL

The semantic object model provides the most flexible approach to modeling new database applications. Not only is the model easier to develop, it is more readily implemented on a variety of platforms. Also, in terms of "lifecycle" design, a semantic model allows for rapid and easy future modification. This section provides definitions and examples of different semantic objects. Additionally, transformation of these objects into two-dimensional tables called relations is explained. The goal of this section is to provide both the method and tools for semantic object design.

1. Definitions

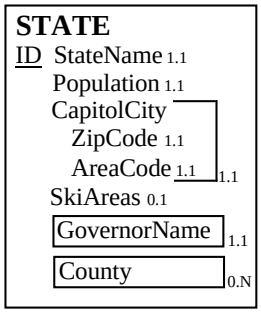


Figure 1. Semantic Object.

Semantic objects represent distinct identities that may or may not have a physical existence. Attributes with associated cardinalities describe the object. Figure 1 represents an object called STATE. StateName is the object identifier. An instance of STATE could be Nevada or New York. StateName is a key attribute that uniquely identifies an instance of the object. A simple attribute of STATE is population because it has a single numerical value. Capitol city is group attribute containing the set Capitol City, ZipCode, and AreaCode. GovernorName is an object attribute that establishes a relationship between one semantic object and another.

Each attribute of the object has a minimum and maximum cardinality. Normally, the minimum is 0 or 1. If it is 0, then no value is required. If it is 1, then it must have a value for the object to be valid. In the previous example, a STATE object may or may not have SkiAreas (in this case a numerical value). Yet, there must be a StateName. The maximum cardinality is the maximum number of instances of the object. Generally, it is either 1 or N (many). If it is 1, then object can have no more than one instance. If it is N, then the attribute can have many different values. In the STATE object example, numerous county names are accommodated. Yet, there is only one GovernorName. Additionally, each attribute has a range of possible values called a domain. The domain may be numeric, string, or enumerated.

Relational database designs require transformation of semantic objects to facilitate platform implementation. To this end, a relation structured as a two-dimensional table containing data is developed from a semantic object. The general rules governing a relation specify that only single values are allowed for each cell. Also, all entries in a column must be of the same type. Lastly, the order of either rows or columns is insignificant. Depending on the perspective, multiple terms define the same characteristics of a relation. Kroenke provides this summary of terms [Ref. 4, p. 126].

Relational Model	Programmer	User
Relation	File	Table
Tuple(Row)	Record	Row
Attribute	Field	Column

Table 1. Terminology.

2. Semantic Objects

There are seven types of semantic objects. Each type is closely related to different aspects of object-oriented programming. Both methods seek to model data and their relationships. Both involve encapsulated data structures and object inheritance. Yet, a large difference resides in application. Semantic objects usually model business data structures, while object-oriented programming techniques generally address scientific data structures. Nevertheless, object-oriented relationships aid the user in understanding semantic objects. The remainder of this section describes relevant semantic objects and shows their transformation into relations.

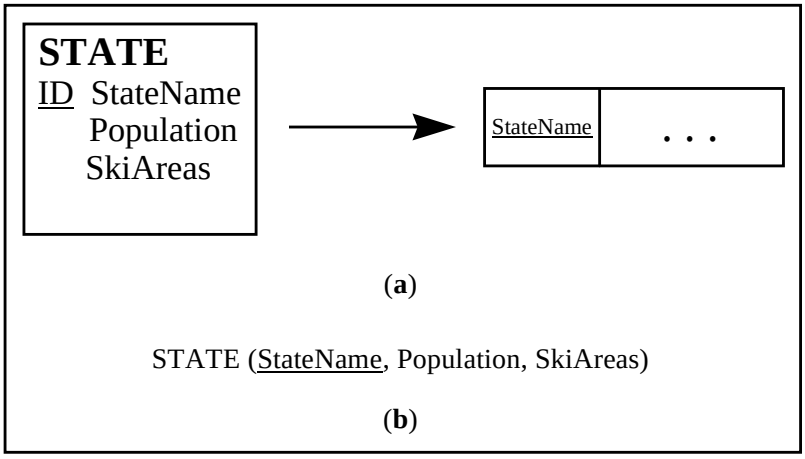


Figure 2. Simple Object Transformation.

Figure 2 shows a simple object and its transformation into a relation. A simple object contains only single-valued attributes and no object attributes. A single relation can represent a simple object in the database. In this case, Figure 2(a) shows the object STATE transformed into a single relation with StateName as a key attribute. Note that if there are no key attributes, then a surrogate key is introduced to ensure that each row is uniquely identified. The ". . ." represent one or more non-key attributes, which in this case are Population and SkiAreas. Figure 2(b) is an alternate representation of the same relation closely resembling column headings of a relation.

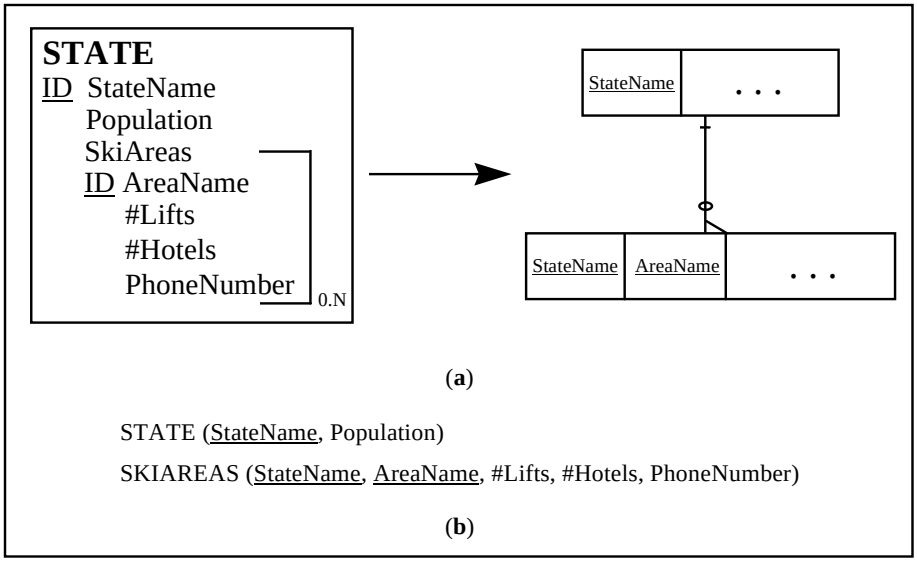


Figure 3. Composite Object Transformation.

Figure 3 is an example of a composite object transformation. A composite object has one or more simple or group attributes with multiple values. There are no object attributes in a composite object. In this example SkiArea is a group attribute. A relation for both the base object STATE and the repeating group attribute SkiArea are created to represent this object. Figure 3(b) shows the attributes of these relations. Notice that SkiArea has two key attributes. AreaName belongs in SkiArea, while StateName is a foreign key attribute taken from the base object. The cardinality of SkiArea is 0.N signifying STATE may have from zero to many SkiAreas. However, STATE must exist for there to be any instances of SkiArea. A hash mark and oval on Figure 3(a) depict this information. Generally, transformations of composite objects require a relation for the base object and one for each multiple valued attribute.

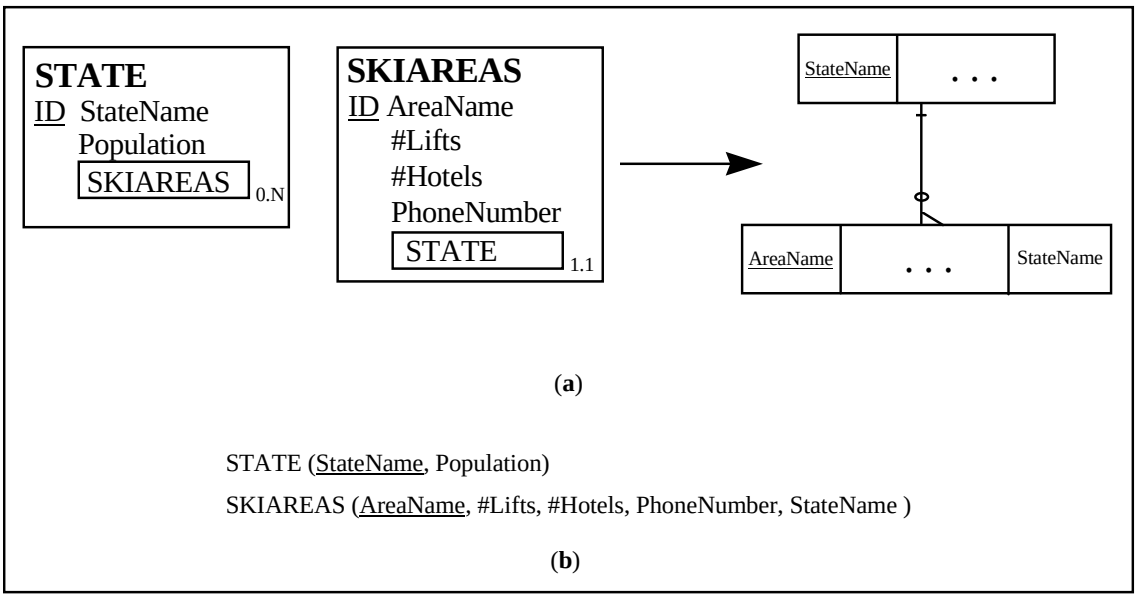


Figure 4. 1:N Compound Object Transformation.

Minor modification to the composite object example provides insight to the compound object. Similar to entities in the E-R model, a compound object contains one or many instances of other objects. Specifically, relationships between objects can be one-to-one (1:1), one-to-many (1:N), or many-to-many (N:M). The object STATE related to another object STATEFLAG is an example of a 1:1 relationship. Each STATE has one and only one STATEFLAG. The converse is also true. Figure 4 shows a 1:N relationship. A STATE may have many SKIAREAS; but a SKIAREA is related to only one STATE. Once again, ovals or hash marks indicate cardinality. Generally, the transformation of 1:N compound objects involves designating the object of one, as parent, and the object of many, as child. When we place the key attribute of the parent into the relation of the child as shown in Figure 4(b), the child receives a foreign key from the parent.

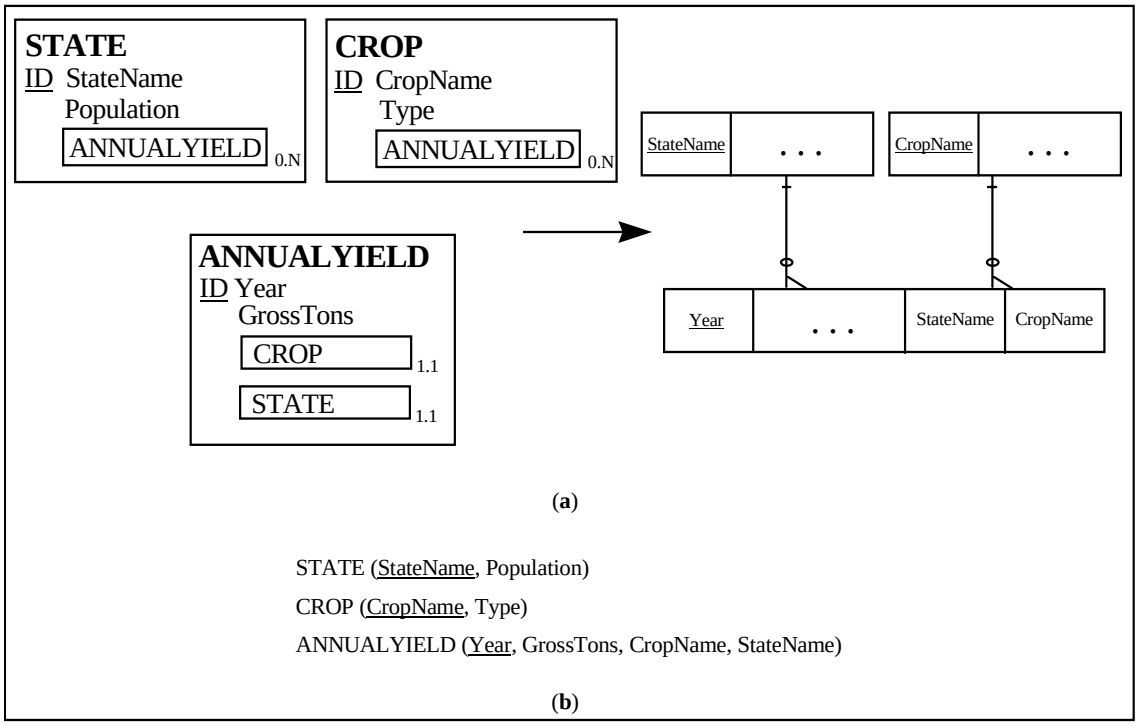


Figure 5. Association Object Transformation.

Many-to-many relationships between two compound objects require three relations. One relation for each object and a third relation representing the intersection between the objects are developed for the transformation. This third relation takes the key attributes of each object. If there are additional data representing aspects of this relationship, then it is called an association object. Figure 5 details the transformation of an association object. In this example, ANNUALYIELD encapsulates data on the gross tons of a crop from a particular STATE object. Generally, transformation of association objects requires the same process as a compound object. However, the child receives a foreign key from each parent in this case.

The remaining semantic objects consist of variations on a simple, composite, compound, or association object. Hybrid objects combine objects of different types. Deceptively simple in theory, a hybrid object can allow logical errors in data encapsulation. Parent/subtype objects allow for inheritance of object attributes. Basically, they are extensions of the four basic object types which preclude data

redundancy. However, a parent/subtype object can increase platform run time by requiring multiple relation searches. The archetype/version object produces other semantic objects tagged as a version or release. The primary focus of this effort will be on implementation of the four basic semantic object types.

C. STRUCTURED QUERY LANGUAGE

The designer must possess keen knowledge of Structured Query Language (SQL) to properly structure the database. Basically, SQL is the language used to retrieve data from the database. Developed in the 1970s by IBM, it is the industry standard in terms of a data manipulation language. The American National Standards Institute (ANSI) maintains SQL and updates versions of the language periodically to provide "lifecycle" management. This section provides a basic understanding of SQL capabilities.

Relational algebra sets the foundation for SQL implementation. There are six primary operations in relational algebra. Similar to algebra, relational algebra treats relations as variables. Relational algebra operators manipulate relations to form new relations. The common term for these new relations is report. The *union* of relations adds the rows of the relations into a third relation. The *difference* between two relations produces a new relation with rows not in the two original relations. The *intersection* of two relations is a third relation of rows in both original relations. The *product* of two relations, consisting of m and n rows respectively, produces an m-row by n-column square matrix relation. *Projection* places specified attributes from a relation into a new relation. The *selection* operator identifies rows to be placed in a new relation. The *join* operator brings together applications of the product, selection, and projection operations. For example, a join command could combine a selection of rows of certain logical value with a projection of certain attributes into a new relation.

StateName	Population	TimeZone
Arizona	3,665,228	Central
Washington	4,866,692	Pacific
Hawaii	1,108,229	Pacific
California	29,760,021	Pacific
Vermont	562,758	Eastern

AreaName	#Lifts	#Hotels	PhoneNumber	StateName
Crystal	6	0	212-555-1212	Washington
Vail	18	21	313-555-1122	Colorado
Squaw Valley	12	3	415-555-1312	California
Taos	4	16	209-555-1414	Arizona
Stowe	8	10	914-555-1513	Vermont

(**a**)

AreaName	StateName
Crystal	Washington
Squaw Valley	California

(**b**)

Figure 6. Sample Query.

Figure 6 portrays a multiple relation query from earlier semantic object examples. In this case, Figure 6(b) is a relation derived from Figure 6(a) relations. A join command consisting of selection commands ensuring both TimeZone = Pacific and #Lifts $\geq$ 5 discriminate row inclusion. A join command consisting of projection commands identifies column inclusion into the resulting relation. Seemingly complex, SQL facilitates simple set theory logic to semantic objects.

D. MEASURES OF EFFECTIVENESS

The motivation for developing a database comes from organizational goals. Using SQL, the user has the ability to query multiple relations to extract relevant information. Generally, the user wants a metric or Measure of Effectiveness (MOE) to aid classification of organizational performance. Now, the user must classify the origin of the data in order to produce meaningful interpretations. Specifically, the user must consider the rules for assigning a value to an observation. At this point, the user must articulate the meaning of encapsulated data.

MOEs establish a consistent, measurable, performance orientated view of the effectiveness of an organization. Built on data, a MOE is dependent on the origin of the

measurement. There are four levels of measurements: nominal, ordinal, interval, and ratio. Table 2 provides a summary of these levels of measurement [Ref 5, p. 51].

Scale Type	Basic Operations	Permissible Statistics	Examples
Nominal	Determine equality or difference	Mode	Type or model
Ordinal	Determine greater or less	Median Percentiles	Military rank School Grades
Interval	Determine equal intervals	Mean Standard Deviation	Temperature Time
Ratio	Determine equal ratios	Percent variation	Length Density

Table 2. Scales of Measurement.

The user must avoid mixing scale types when formulating relationships between data. Specifically, MOEs should be drawn from data possessing the same scale type to preclude skewed interpretation. The user may translate data from one scale type to another if attention to the assumptions taken in this process is focused. This process of translation will introduce subjective evaluation of the data.

Seemingly an element of common sense, developing relevant MOEs entails a never ending search for the pertinent portrayal of the meaning of data. Success in this endeavor comes from iterative attempts to present data in a form useable to the decision maker. Each attempt reflects refinement based on the goals of the decision maker.

IV. METHODOLOGY FOR MODELING DATA ENCAPSULATION

The NTC requires a database structure that both facilitates modular implementation and reflects simplicity. Previous efforts often combined subjective and objective data with complex input requirements as a byproduct. The first step in this effort is the development of semantic objects modeling the NTC environment. The database will accommodate administrative, objective, and subjective data encapsulation. In general, input of administrative data occurs either at the start of a rotation or during change of mission. Objective and subjective data inputs are event driven. Next, the logical schema or structure of the database relations is produced through normalization aided by SALSA [Ref. 4]. At this point reports representing meaningful relationships among data can be drawn from the database. The focus of this chapter lies in structuring both administrative and objective data into a tractable database using a bottom-up design perspective. Additionally, Measures of Effectiveness (MOE) relevant to the unique NTC environment are presented with supporting Structured Query Language (SQL). Concurrent efforts in a Naval Postgraduate School thesis titled *A Methodology for Modeling Subjective Data Encapsulation at the National Training Center, Fort Irwin, CA.* address subjective data encapsulation [Ref. 1].

A. SEMANTIC OBJECT DEVELOPMENT

1. Administrative Data Semantic Objects

The NTC references a player in multiple fashions. Driven by incremental implementation of training evaluation methods, each reference serves a different role. The structure of this portion of the database is crucial for several reasons. First, it must allow access to present data storage conventions. Next, the structure must allow future simplification of these conventions. Lastly, the structure of administrative data must facilitate simple input requirements for the user.

Reference	Definition	Source	Form	Unique	Example
Icon number	Graphical reference number	Core Instrumentation Subsystem (CIS)	String	Yes	T21A
DCI number	Data Communication Interface number	Serial number for accountability/identification	String	Yes	1234
PID	Player Identification	Vehicle/Personal Detection Device (VDD or PDD)	Hexa-decimal	95%	FF050A0F
FID	Fire Identification	CIS transformation of PID into decimal form	Decimal	95%	255.5.10.15
Administrative Bumper number	Unit specific method of marking a vehicle chassis	Unit	String	No	C100
NTC Bumper number	NTC side markings on a vehicle chassis	NTC	String	Yes	1223
Line number	Database surrogate key	CIS	Numeric	Yes	2101

Table 3. Player References.

Table 3 summarizes NTC methods for identifying players. Basically, a player can be referenced in seven different fashions. Of interest to the database designer are those forms of reference that are both emitted and unique. The only element in this category is the Data Communication Interface (DCI) number. The database also requires a means for the OC to identify a player for data input. Generally, an OC utilizes the administrative bumper number along with the task force designation for player identification. This combination of data is unique. The model design is based on the logic that attribute uniqueness is important.

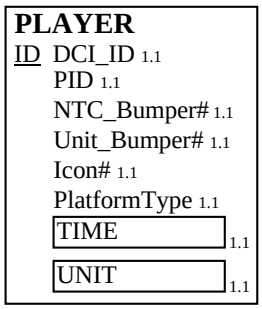


Figure 7. PLAYER Object.

The core element of the database is a player. At the NTC a player connotes a vehicle or individual soldier. Vehicle players include M1 main battle tanks, M2 infantry

fighting vehicles, and other elements of a task force involved in the training. The PLAYER object, shown in Figure 7, uniquely describes each player with the key attribute DCI_ID. The attribute NTC_Bumper# (1244) uniquely identifies the NTC assigned chassis number for a vehicle player. In the case of an individual soldier, it identifies the assigned vehicle, or the lack thereof, with an entry of zero. Unit_Bumper# (B210) follows the same logic; however, it is not unique. Rotating units designate administrative bumper numbers by unit paradigm. Note that a Unit_Bumper# is unique to a battalion size unit or below, but may not be unique across a task force. Icon# uniquely identifies the CIS graphical reference of the player. The remaining attributes are non-unique. PID identifies the player in hexadecimal (base 16) format. The attribute PlatformType distinguishes the type of platform such as M2 or T80. TIME marks the occurrence of this relationship between a DCI_ID and the other attributes. UNIT designates unit assignment of the vehicle or individual soldier. Both UNIT and TIME are references to other objects. The cardinality of all attributes is one-to-one.

The ability to draw information based on the grouping of players is an important capability of the database. One must have the ability to query the database by either different unit sizes or unit groups such as a platoon, company, or task force. A task force consists of a consortium of platoons and companies organized to execute a given mission. A player is permanently assigned to a unit at home station, but it may fight under the control of several different companies or task forces during a rotation. Also, the user must be able to differentiate between task force composition during different phases. The ability to group players into a unit and subsequent unit groupings into a task force necessitates the formation of new objects.

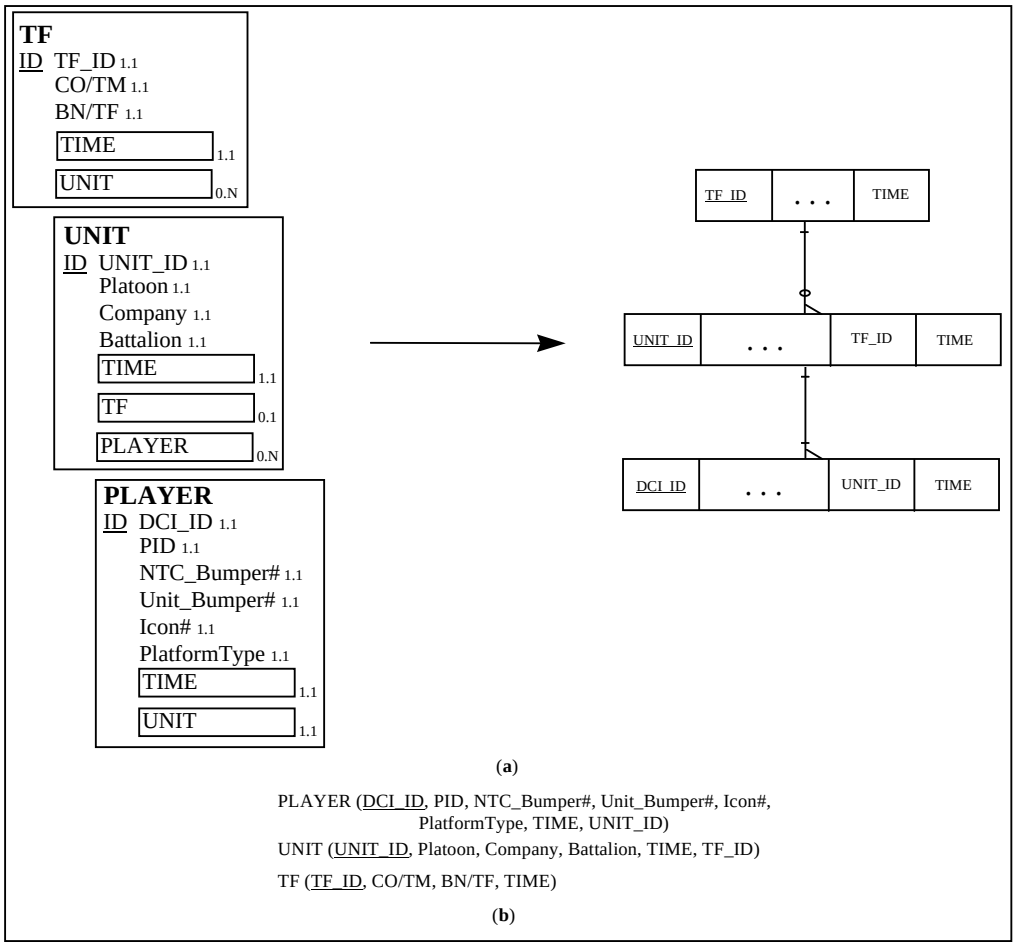


Figure 8. UNIT/TASK FORCE Objects.

Figure 8(a) portrays the relational representation between compound objects PLAYER, UNIT, and TF. While the UNIT object groups players into a unit, the TF object groups units into a task force. UNIT_ID and TF_ID are surrogate key attributes that accommodate non-unique relationships between a player and either a unit or task force over time. For example, a unit may be task organized to a different task force during a rotation dependent upon mission requirements. Battle losses could force unit consolidation. The UNIT object contains the one-to-one attributes Platoon, Company, Battalion, and TIME. DCI_ID and TF_ID have a cardinality of 0 to N and 0 or 1, respectively. The TF object contains the one-to-one attributes CO/TM (company/team), BN/TF (battalion/task force), and TIME. UNIT_ID has a cardinality of 0 or 1. The

minimum cardinality plays an important factor in these objects since if it is 1, an entry must exist in each instance. For these objects, a player must have a unit; while a unit may not belong to a task force. Figure 8(b) shows the resulting relations from these objects after the application of the SALSA program. Note that compound object reference in relations will require a unique attribute name. For simplicity, compound object reference is left intact in all figures.

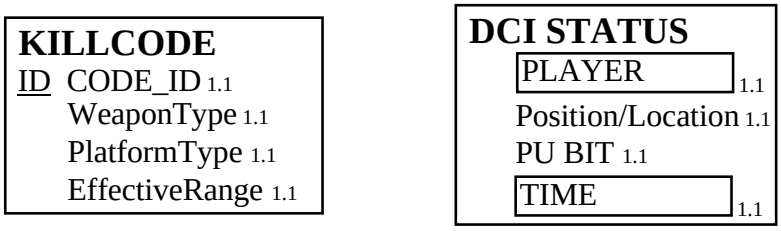


Figure 9. KILLCODE/DCI STATUS Objects.

Additional administrative data must be included in the database to draw meaningful interpretations of objective inputs. MILES II emits both a "kill code" and the Firer Identification (FID) of the aggressor in an engagement. Note that the MILES II emission degrades over distance to facilitate a differential in probability of hit and kill calculations on the engaged player. Yet, range of engagement is not transmitted. The KILLCODE object in Figure 9 allows categorization of MILES II kill codes. One-to-one attributes of WeaponType (120mm, 25mm), PlatformType (M1, M2), and EffectiveRange (1,500 meters) describe this object. Given a kill code, this object describes the aggressor to a greater extent. Objects of this variety, those that highlight player parameters, will give substantive meaning to interpretations drawn from subsequent semantic objects.

DCI STATUS provides updated instrumentation functionality information including the Position/Location, Built-in-Test (BIT), and TIME. Although no attribute is unique in this object, the most recent time provides the relevant information. At the NTC a DCI is replaced as soon as possible. A numeric surrogate key will be introduced into the DCI STATUS relation for normalization. Efficient implementation of this database

mandates buffered storage of objects like DCI STATUS; specifically, if the information is only relevant for a short period of time, then it should only be stored temporarily.

2. Objective Data Semantic Objects

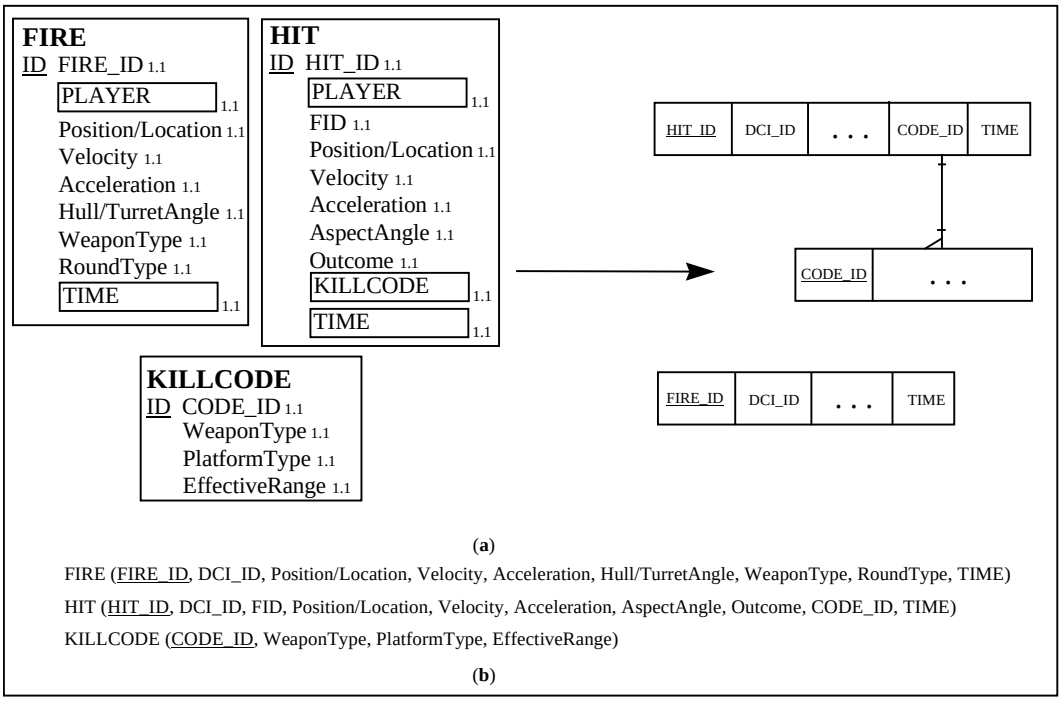


Figure 10. FIRE/HIT Objects.

The NTC provides the rare opportunity for a unit to record battlefield events during mission execution. Figure 10(a) highlights two new objects, FIRE and HIT, that encapsulate data from player confrontations. Both objects possess one-to-one attributes: PLAYER, Position/Location (Grid Reference), Velocity, Acceleration, and TIME. The FIRE object is further described by a surrogate key FIRE_ID and information peculiar to firing a round. Hull/TurretAngle is the angle between the weapon and chassis when a player fires. WeaponType identifies the weapon while RoundType is the type of round. The HIT object also contains a surrogate key called HIT_ID. This object obtains two attributes from MILES II transmitted data. The FID references the foe. Note that a FID is a decimal format transformation of a PID hexadecimal value. The KILLCODE references the engaging weapon type. The Outcome indicates the result of the event as

determined by the MILES II system. Figure 10(b) shows the resulting relations after the application of SALSA without compound object variable transformation.

3. Calculated Data Semantic Objects

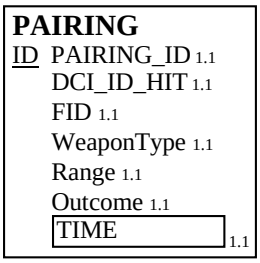


Figure 11. PAIRING Object.

Previous objects modeled objective inputs obtained through the NTC data stream via the Range Data Management Subsystem (RDMS). To further enhance the ability to draw meaning from the database requires some basic calculations. Specifically, the range differential between confrontational players is not emitted. To solve this problem a new object must be modeled from calculated data. Figure 11 details the object PAIRING with surrogate key PAIRING_ID. DCI_ID_HIT and FID identify the two player vehicles in the event. Note that there is only a 95% chance the FID value is unique. WeaponType identifies the type of weapon used in the event. Range is obtained through a position comparison of each vehicle in the confrontation. Outcome and TIME provide the same function as in the HIT object.

B. MEASURES OF EFFECTIVENESS

The NTC provides realistic training; however, mission execution is skewed. Specifically, disengagement criteria are generally ignored in that most battles at the NTC are fought to a cataclysmic end. To maximize training value and minimize expense, the NTC executes training scenarios fast with complete player participation. These factors drive MOE development to focus on time, distance, and casualties in a quantitated fashion. The following section considers each factor and presents a meaningful MOE with supporting SQL logic. Database manipulation utilizes previously defined relations.

All MOEs utilize fictitious forces, (TF 1/61 Armor, TF 1/52 Mech) versus a Motorized Rifle Battalion (1/MRB), that are designated Blue and Red, respectively. This mock battle encompasses 200 players with 16,000 random engagements. Appendix A contains portions of each relation with associated sample data. Software used for this section includes the Microsoft Office Suite. Of interest to the reader is the multiple platform capability of the SQL commands.

1. Time

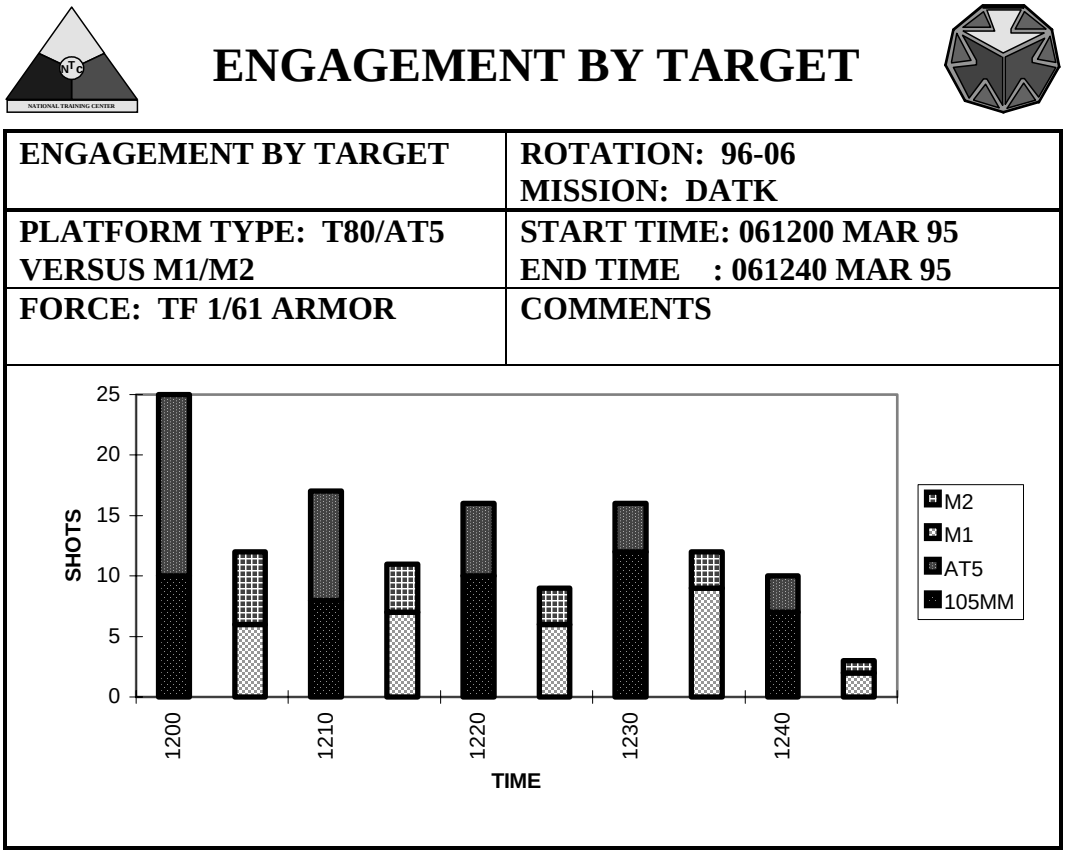


Figure 12. Time MOE.

Beyond the simple comparison of objective data comes the combination of objective data into something more meaningful. Figure 12 portrays the number of rounds fired by Red tank killing weapons (105MM, AT5) versus the number of Blue tank killing

platform (M1, M2) hits. From time 1200 to 1210 there were 25 total shots by Red weapons. AT5s fired 15 rounds, while T80s fired 10. Blue players incurred 12 hits, with 6 each for M1 and M2 platforms. Subsequent time periods are also shown.

Indexed by time, this chart affords the opportunity to draw several conclusions. First, one can quickly determine which Red weapon system is firing the most frequently. Coupled with knowledge of the actual battle, one can draw conclusions on Red responses to Blue activities during specific time periods. Also, the ratio of Blue hits inflicted show both Red force ability and priority of fire. Mainly, one can infer a higher mass of fire if there are more hits incurred. Additionally, the difference in column height, between Red shots and Blue hits for a given time, indicates the effectiveness of Red fires. Specifically, a smaller differential indicates higher effectiveness. Lastly, Priority Intelligence Requirements (PIR) could be driven by data in this form.

Obtaining this report requires queries to the generated database. This database can be formed by altering the present data stream at the NTC. In this case, both the FIRE and HIT relations are queried to draw information on both the total Red fires and total Blue hits. Note that just querying the HIT relation does not account for any Red shots that did not register as a Blue hit. The following SQL produced the Red portion of the necessary output:

```
SELECT DISTINCTROW FIRE.TIME_ID4, FIRE.WeaponType
FROM ((PLAYER INNER JOIN FIRE ON PLAYER.DCI_ID = FIRE.DCI_ID1) INNER JOIN UNIT ON
PLAYER.UNIT_ID1 = UNIT.UNIT_ID) INNER JOIN TF ON UNIT.TF_ID1 = TF.TF_ID
WHERE (((FIRE.TIME_ID4)>=1200 And (FIRE.TIME_ID4)<=1210) AND (([TF]![TIME_ID3])>=1200
And ([TF]![TIME_ID3])<=1800) AND (([TF]![BN/TF])="1/MRB") AND
(([FIRE]![WeaponType])="AT5" Or ([FIRE]![WeaponType])="105MM"))
ORDER BY FIRE.TIME_ID4;
```

This SQL selects a distinct row from the FIRE relation when a Red weapon is of the correct type and in the time span queried. The JOIN commands logically connect relations involved in the query. In this example, the objects PLAYER, UNIT, and TF contain relevant information to the query. Using a different perspective on the time attribute in these objects, the JOIN command mandates selection of rows in the FIRE

relation only if the player is assigned to the selected task force. In this case, a Red player must belong to the 1/MRB in the selected time frame which may or may not be the same as the event time frame.

One can draw a parallel between the term rotation “phase” and the previous manipulation of the time attribute in PLAYER, UNIT, and TF. This subtle point requires distinction between the time of an event and the time of player groupings. Generally, units switch task forces during a mission change or phase of rotation. Therefore the time span of a unit grouping is usually much larger than one queried for events. Also, an event time period is a subset of a unit grouping time period. The following queries utilize this concept of phase to discriminate in row selection.

The following SQL produced the Blue output:

```
SELECT DISTINCTROW HIT.TIME_ID5, PLAYER.PlatformType, [KILL CODE].PlatformType,
HIT.Outcome
FROM (((PLAYER INNER JOIN HIT ON PLAYER.DCI_ID = HIT.DCI_ID2) INNER JOIN UNIT ON
PLAYER.UNIT_ID1 = UNIT.UNIT_ID) INNER JOIN TF ON UNIT.TF_ID1 = TF.TF_ID) INNER JOIN
[KILL CODE] ON HIT.CODE_ID1 = [KILL CODE].CODE_ID
WHERE (((HIT.TIME_ID5)>=1200 And (HIT.TIME_ID5)<=1210) AND ((HIT.Outcome)<>"NM") AND
(([TF]![TIME_ID3])>=1200 And ([TF]![TIME_ID3])<=1800) AND (([TF]![BN/TF])="1/61") AND
(([PLAYER]![PlatformType])="M1" Or ([PLAYER]![PlatformType])="M2") AND
(([HIT]![CODE_ID1])=3 Or ([HIT]![CODE_ID1])=4))
ORDER BY HIT.TIME_ID5;
```

This SQL selects a distinct row from the HIT relation when a Blue platform is in the selected time span and of the correct type. Also, the Blue player must have the correct kill code with an outcome other than “near miss”. Lastly, the Blue player must be in the selected task force during the given phase. Combining queries over pertinent spans of time produces the resulting MOE.

User input for this MOE entails Blue and Red parameter selection. Both the HIT and FIRE relation must be queried by either weapon type or platform type. The user must also select a relevant time span, such as a peak confrontation period, for the query. The user must select appropriate units or task forces for comparison. Lastly, the user

should select compatible weapons for comparison (e.g., tank killer versus tank killer, instead of dismounted soldier versus tank).

2. Distance

Presently, distance data are obtainable at the NTC; however, their accuracy is questionable. The main contributor to inaccuracy is the non-uniqueness of PID/FID values. Nevertheless, these data could provide extremely useful MOEs. Modifications to the data stream are required to produce relations such as the previously defined PAIRING. Additionally, a statistical comparison of pairing information with other objective data is needed to classify data accuracy.

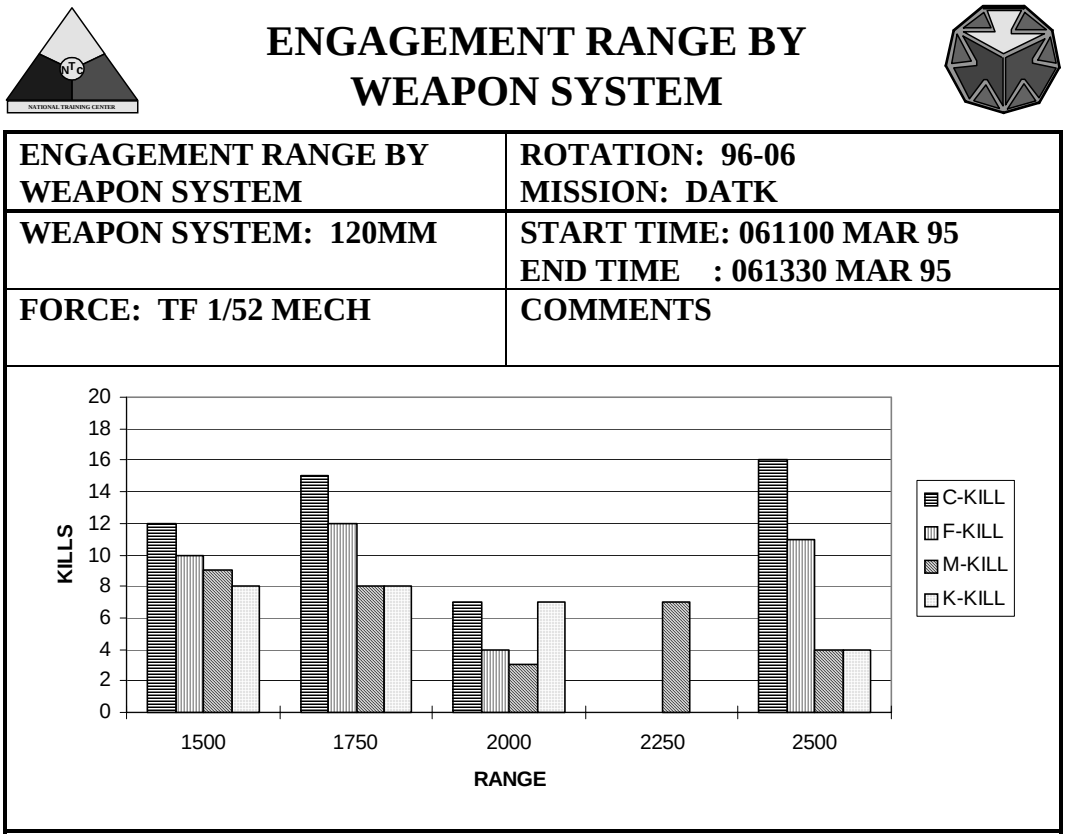


Figure 13. Distance MOE.

Figure 13 incorporates vision from NTC leadership on a MOE related to distance. [Ref. 6] This MOE details the effectiveness of Blue 120MM weapon systems against Red forces over distance. Also, it provides information on the type of Red kills obtained. For example, at a range band of 1,500-1,750 meters, Blue 120MM weapon systems from TF 1/52 MECH obtained 12 Commo-kills, 10 Firepower-kills, 9 Mobility-kills, and 8 Catastrophic-kills in the selected time frame. This MOE relates multiple concepts to the decision maker. First, if Blue obstacles are emplaced at certain ranges for this mission, then the adequacy of direct fire coverage can be ascertained. Also, the relative distribution of Blue fires compared to the known effective range may direct future fire planning.

Obtaining data for this MOE entails a relatively simplistic query on the PLAYER, UNIT, TF, and PAIRING relations. The following SQL produced the necessary output:

```
SELECT DISTINCTROW PAIRING.TIME_ID7, PLAYER.PlatformType, TF.[BN/TF], PAIRING.Range
FROM ((PLAYER INNER JOIN UNIT ON PLAYER.UNIT_ID1 = UNIT.UNIT_ID) INNER JOIN TF
ON UNIT.TF_ID1 = TF.TF_ID) INNER JOIN PAIRING ON PLAYER.DCI_ID =
PAIRING.DCI_ID_FIRE
WHERE (((PAIRING.TIME_ID7)<=1400 And (PAIRING.TIME_ID7)>=1200) AND
(([PAIRING]![WeaponType])="120MM") AND (([TF]![TIME_ID3])>=1200 And
([TF]![TIME_ID3])<=1800) AND (([PAIRING]![Outcome])="NM") AND (([PAIRING]![Range])<=1750
And ([PAIRING]![Range])>=1500) AND (([TF]![BN/TF])="1/52"))
ORDER BY PAIRING.TIME_ID7;
```

The logic of the query requires selection of a row from the PAIRING relation if a Red player is engaged in the selected time span. The Blue player must be assigned to the selected task force during the phase. Also, the Blue player must utilize the correct weapon. Lastly, the engagement must occur in the proper range band.

3. Casualties

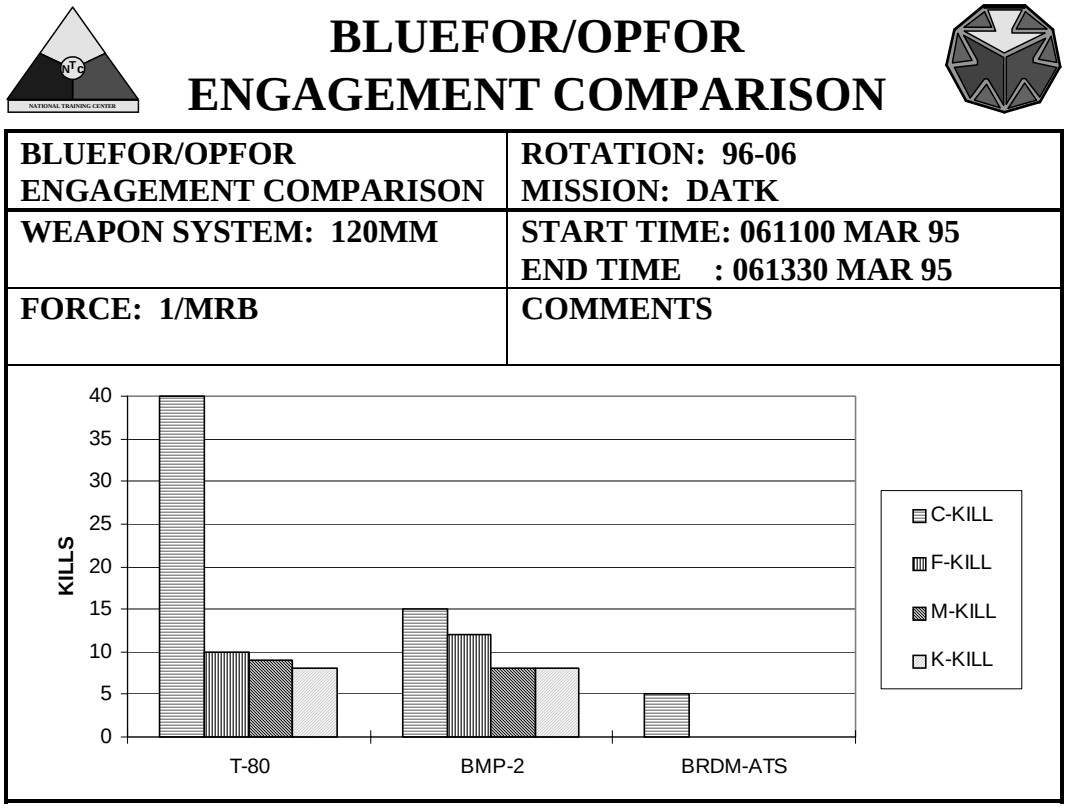


Figure 14. Casualty MOE.

Taking direction from NTC leadership on MOEs, Figure 14 highlights the effectiveness of Blue 120MM weapon systems on several different Red platforms from 1/MRB [Ref. 6]. In this case, 120MM weapon systems obtained 30 C-kill, 10 F-kill, 8 M-kill, and 7 K-kill in the time frame. This MOE relates to the decision maker key information on several aspects of the battle. Predefining the priority of fires, the decision maker can see the execution of these directives. Also, future focus on targeting priorities may be identified.

Production of this MOE requires a query of the PLAYER, UNIT, TF, HIT, and KILLCODE relations. The following SQL produced the output:

```
SELECT DISTINCTROW HIT.TIME_ID5, PLAYER.PlatformType, TF.[BN/TF], HIT.Outcome
FROM ((PLAYER INNER JOIN (HIT INNER JOIN [KILL CODE] ON HIT.CODE_ID1 = [KILL
CODE].CODE_ID) ON PLAYER.DCI_ID = HIT.DCI_ID2) INNER JOIN UNIT ON
PLAYER.UNIT_ID1 = UNIT.UNIT_ID) INNER JOIN TF ON UNIT.TF_ID1 = TF.TF_ID
WHERE (((HIT.TIME_ID5)<=1230 And (HIT.TIME_ID5)>=1200) AND (([TF]![TIME_ID3])>=1200
And ([TF]![TIME_ID3])<=1800) AND (([HIT]![Outcome])<>"NM") AND (([TF]![BN/TF])="1/MRB")
AND (([HIT]![CODE_ID1])=1) AND (([PLAYER]![PlatformType])="T80" Or
([PLAYER]![PlatformType])="BMP" Or ([PLAYER]![PlatformType])="BRDM"))
ORDER BY HIT.TIME_ID5;
```

The logic of the query mandates a Blue player must utilize the correct weapon. A Red player must have the appropriate kill code and be assigned to the selected task force during the phase. All events must occur in the selected time span. Alternative methods of obtaining this output include querying the PAIRING relation with the same logic.

C. SYNOPSIS

The NTC database structure must start simply to afford the possibility of incremental contractor implementation. Also, the use of simple objects affords widespread understanding of the database structure. To this end, the basic structure must focus on the core element of emissive data, an individual player. If necessary, create multiple objects which correlate different player identification conventions. With any convention, unit and task force relationships build on the core. The next step is the alteration of the data stream in the RDMS somewhat akin to the method of wargame statistical data manipulation.

Implementation of this structure requires effective organization of DCI installation on the players. As the only unique emitted data, it is the only choice for the key attribute of the PLAYER object. Logically, assigning a DCI permanently to a particular vehicle will greatly reduce administrative data input requirements. At this point, administrative data input lies in NTC bumper number and unit bumper number correlation with unit and task force groupings.

As an alternative to the DCI number, PID (Player Identification) number usage as the key attribute will necessitate statistical analysis to classify MOE accuracy. Currently, PID numbers are the prevalent method of vehicle identification in the Core Instrumentation Subsystem (CIS); however, PID numbers are only 95% unique. Assuming randomness, utilization of the PID as a key attribute incurs a 10% error in data. Further statistical analysis is required under this option.

At this point, MOE development will be driven by the decision maker and relevant doctrinal procedure. Just a few of many possible combinations of objective data are presented in this chapter. However, one must remember several facts in MOE development. First, the NTC is a unique training environment with skewed withdrawal criteria. Additionally, any data holds the lure of comparison; but, each rotation is unique unto itself. Yet, manipulation of objective data in all possible combinations with statistical reference has intrinsic value.

V. METHODOLOGY FOR MODELING NTC COMMUNICATION NETWORK

The NTC requires a mathematical model (network) with the capability for flexible analysis of a modular fiber optic cable system to capture emitted data. This cable system will traverse the NTC connecting information centers. To this end, incorporation of algorithms and user requirements into a tractable computer application is an efficient method to derive a solution to this problem. User requirements span consideration of terrain impact, cost of the system, and obtaining interpretable results. Relevant network algorithm categories to this problem include both the Shortest Path and the Minimum Spanning Tree methods. Combining these items into a mathematical model provides the user a means for solving the problem under multiple scenarios. This chapter provides the reader with a basic understanding of network concepts and definitions. Additionally, methods for characterizing user requirements into a network model are shown. The goal of this chapter is to impart to the reader the conceptual capability to do cost comparisons of multiple fiber optic cable designs.

A. NETWORK DEFINITIONS

The first step in modeling the NTC environment requires the portrayal of terrain. Conceptually, terrain can be reduced to a 2-dimensional representation called a *graph* or *network*. The following is an adaptation of definitions provided by Ahuja, Magnanti, and Orlin [Ref. 7]. Figure 15 provides an example of a graph with *N* (in this case 5) *nodes* and *M* (9 for this example) *arcs*. In terms of terrain, a node is a location while an arc represents a means of transit between one location and another. Also, an arc generally has a cost and capacity limit associated with its use. This particular graph is undirected because arcs are unordered pairs of nodes. Specifically, the arc joining nodes i and j can be represented by (i,j) or (j,i). Also, each arc has a head and tail indicating where it terminates and emanates, respectively. The *indegree* of a node is the number of arcs terminating at that node. The *outdegree* is the number of arcs emanating from a node. A

node connected to another node by a single arc is *adjacent*. Lastly, the node adjacency list is the set of nodes adjacent to a particular node.

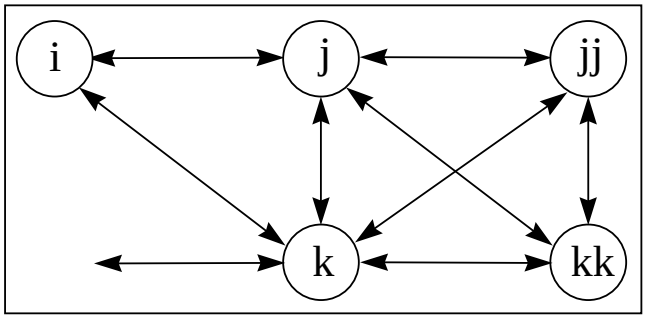


Figure 15. Network Example.

A *path* is a sequence of nodes without repetition of any nodes. A valid path in Figure 15 is i-j-k. A *cycle* is a path that allows a node to be revisited. For example, taking the arc set {(i,j),(j,k)} and adding arc (k,i) creates a cycle. Two nodes are *connected* if there exists at least one path between them. A graph is strongly connected if there is a path from every node to all other nodes. A *tree* is a connected graph without any cycles. Further refinement on this definition dictates that a spanning tree, composed only of selected nodes, meets the same criteria.

There are two methods for storing graph information within a computer relevant to this application of modeling terrain. In this case, there are eight arcs, or directions, a path may transit. The Node-Arc incidence matrix stores the graph by initializing a matrix with a row for each node and a column for each arc. There are three possible entries in the matrix that include a '1' for the head of an arc, a '-1' for the tail of an arc, and a '0' for no arc. The Node-Node Arc adjacency matrix represents the graph with a matrix with a row and column for each node. A '1' signifies adjacency, while a '0' indicates the lack thereof.

B. TERRAIN REPRESENTATION

Armed with this knowledge of terminology, translation of terrain information into a network is possible. The first step is assigning node numbers to known locations.

These nodes must encompass the entire area of study to ensure adequate coverage. Next, distance is calculated from one node to adjacent nodes by comparing X, Y, and Z coordinates. At this point the user may dictate the relative or absolute cost of transiting through a node.

The key decision in this process of modeling terrain is determining the data resolution required to obtain accurate results. For example, organizing a network based on 10 kilometer increments which provide area coverage requires little storage space. However, the results will be useless because one cannot typify terrain over such a large distance. On the other hand, a network based on 1 meter increments would provide extremely accurate results. Yet, the storage requirement would blossom on a cubic order to achieve this accuracy.

C. ALGORITHMS

Given a set of nodes that must be connected, the user could simply enumerate all possible routes and choose the best combination of arcs. As an aside, another term for a node requiring cable connection is demand node. The idea of enumerating all possibilities soon becomes intractable considering the number of possible node combinations in a large network. To solve this problem, two algorithm categories assist in finding a solution. First, the Shortest Path method determines the cheapest way to traverse a network from one node to another. Then, the Minimum Spanning Tree method selects the cheapest combination of arcs that connect the demand nodes. The remainder of this section provides the user with a keen understanding of selected computational methods for implementing each algorithm.

1. Shortest Path

Lawphongpanich provides a useful implementation of the Modified Label-Correcting algorithm that solves the shortest path problem [Ref. 8]. The first step in the algorithm requires selection of a node as the source with a resulting distance label of ‘0’ and a predecessor of ‘0’. The distance label indicates the accumulated cost of transiting

to a node. The predecessor is the previous node in the path. Next, all other nodes in the network receive a distance label of large magnitude, or infinity. Now the algorithm updates node distance labels by comparing adjacent node distance labels plus the associated arc cost. An update occurs when the distance label value is larger than the compared value. Predecessor assignments occur simultaneously. The algorithm terminates when no distance labels are updated through a check of all nodes.

2. Minimum Spanning Tree

Prim's algorithm solves the Minimum Spanning Tree problem. [Ref. 8] The first step in this algorithm requires assignment of a node in the network to a set $\{ S \}$. All other nodes are assigned to the complement of set $\{ S \}$ which is $\{ \overline{S} \}$. At this point the cheapest arc transiting from $\{ S \}$ to $\{ \overline{S} \}$ is selected. The node on the head of this arc is added to set $\{ S \}$ and subtracted from $\{ \overline{S} \}$. This algorithm terminates when a sufficient number of arcs connect the network. This is the number of nodes minus one iteration.

D. PROJECTION OF SYSTEM REQUIREMENTS

The determination of future system requirements entails analysis of several factors. First, historic use of locations for information transfer offers a starting place. Presently, the NTC uses multiple field locations for information intensive AARs. Current maps and environmental studies provide locations of both impact and off-limit areas. Also, traditional high usage areas are known from previous rotations. This is an important factor in routing the cable away from areas of likely damage. Future requirements for the cable system are also driven by technology. Wireless computer network connection is possible by an antenna system with area coverage. This system will utilize a cable connection. Chapter VI explores a starting solution to this problem.

VI. NTC COMMUNICATION NETWORK IMPLEMENTATION

Given a foundation in network modeling concepts, the next step is building a tool for the NTC capable of comparing alternative communication networks. This tool must facilitate simple manipulation of user inputs that specify network requirements and terrain characteristics. The goal of these inputs is to accurately model the terrain that separates demand locations. Demand locations or nodes denote a user requirement for a link into the communication system or a forced routing of the cable. At this point, the tool must calculate the cheapest set of paths or routes that connect all demand nodes. Built on the concepts presented in Chapter V, this chapter details the development and use of a computer application that provides this tool. The name of this application is the NTC Route Optimizer. Discussion of user inputs and subsequent outputs is also given. As an aid to the reader, a complete set of user inputs is presented in Appendix B.

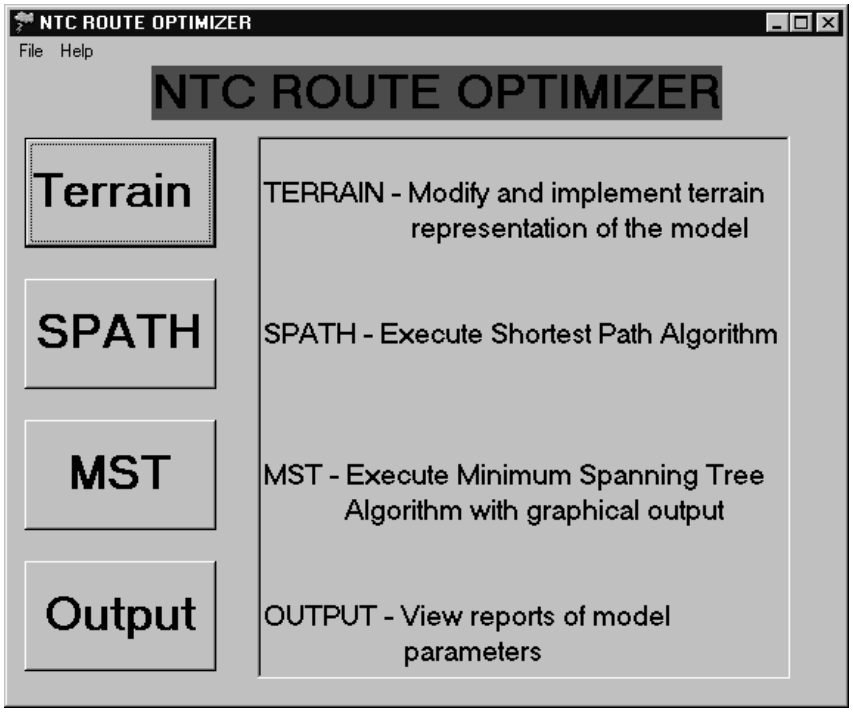


Figure 16. NTC Route Optimizer.

Figure 16 presents the main form for the NTC Route Optimizer. Construction of the program capitalized on the Rapid Application Development (RAD) environment of Delphi Version 2 [Ref. 9]. Object Pascal, a compiled language, is the foundation language in this environment. Conceptually similar to semantic objects discussed earlier in the database portion of this effort, object oriented programming (OOP) allows the developer to efficiently model reality. Objects continue to possess attributes which describe them. Objects within objects are generally referred to as a particular type. Future flexibility is one benefit of OOP in that refinement of object definitions does not necessitate alteration of existing functions and procedures. The Delphi application also aids in the construction of the graphical user interface (GUI). This program display format allows the user to choose commands by pointing and clicking on buttons located on a form. Several forms compose the complete program, with each form containing commands to complete related functions or procedures. Source code to each form is stored in a separate executable file called a unit.

As a note to the user, the NTC Route Optimizer is 32-bit IBM compatible application requiring 8 megabytes of Random Access Memory (RAM) and 7 megabytes of hard disk space. Windows 3.1 or higher is required as the computer operating system. A 486DX processor or higher is required. Installation of the program will create a 'NTC' directory with a 'Data' sub-directory. This program does not alter any system configuration files or registry. This program is a non-threaded application meaning all system resources are utilized during mathematical computations. Appendix C presents a sample program session. Appendix D contains program source code.

A. NETWORK MODEL

The first step in network model development is terrain representation. Selecting the *Terrain* button on the main form delivers the user to the Terrain Data form which contains the commands necessary for this process. Figure 17 shows this form. Initial terrain data were obtained from the geographic information system (GIS). This system contains a database of spatial information. Relevant to this effort were location and

altitude data. The data obtained consist of a reference number, X coordinate, Y coordinate, and a Z value in the form: 1 500002.694 3946420.023 1187.700. The reference or node number identifies the entry starting from the top left or northwest corner of the data set with a value of one. The X and Y entries indicate Universal Transverse Mercator (UTM) grid coordinates. The Z value is the above sea level altitude in meters. Data were obtained for a rectangular region which encompasses the NTC. As a reminder, a node represents a location with a given altitude. Nodes are spaced 500 meters apart thereby providing acceptable resolution in a desert environment.

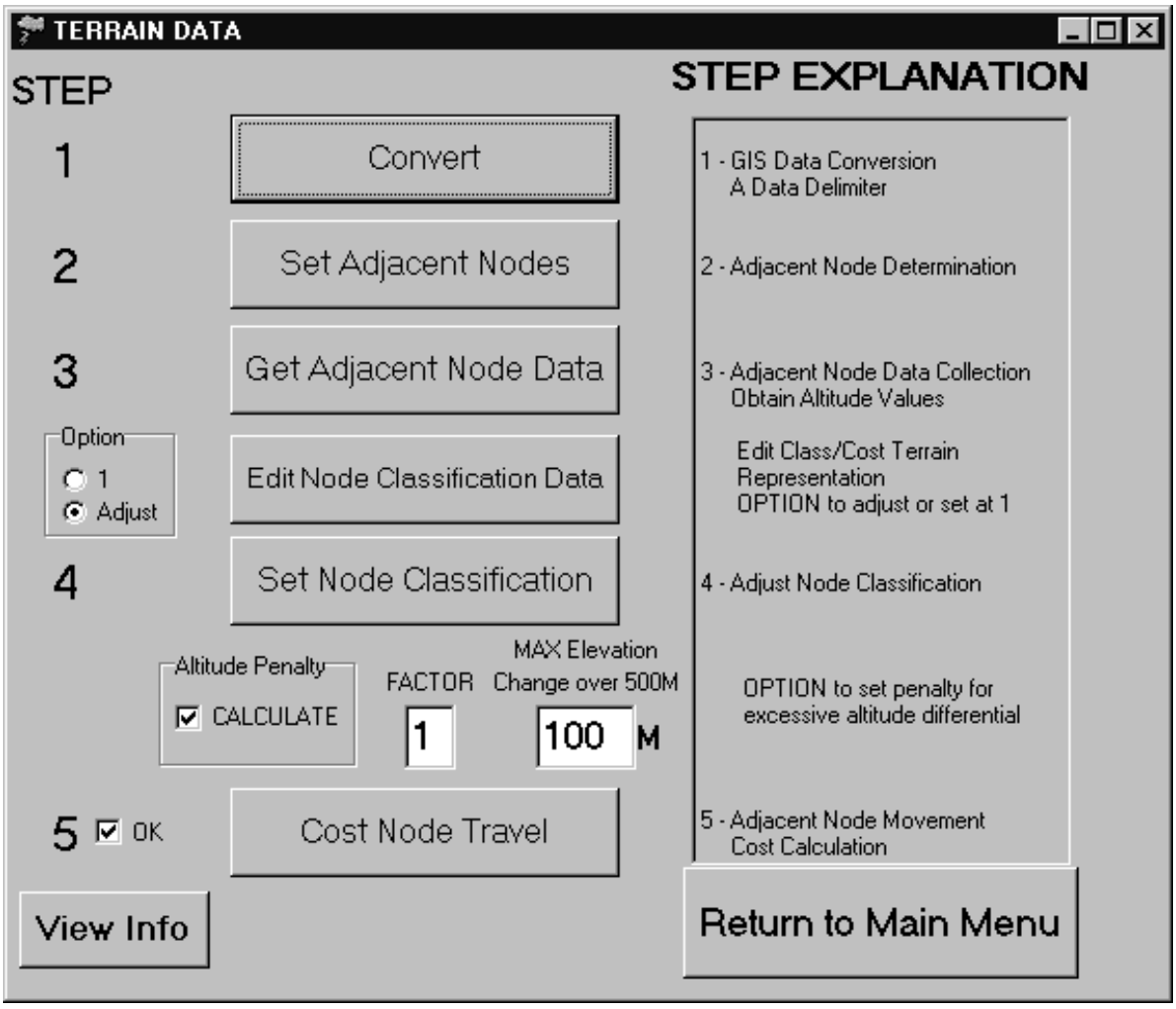


Figure 17. Terrain Data Form.

GIS data require organization and manipulation to allow efficient model implementation. This entails conversion of GIS coordinate and altitude data into integer values to decrease RAM utilization. The *Convert* button accomplishes this task. Then, the eight adjacent node numbers are set using knowledge of the number of data columns in the rectangular region by the *Set Adjacent Nodes* button. Nodes on the border of the region receive a zero for nonexistent adjacent nodes. At this point, the GIS data set is queried to obtain altitude data for each adjacent node with the *Get Adjacent Node Data* button. Execution of these commands entail selection of steps one through three on the Terrain Data form. The results of these operations give the node number, a default class setting, X/Y coordinates, Z value, and adjacent node numbers with corresponding altitudes in the following form (note that Anode denotes an adjacent node):

Node	Class	X	Y	Z	Anode	Z	Anode	Z	Anode	Z	Anode	Z	Anode	Z	Anode	Z	Anode	Z	Anode	Z
18553	7	5040	38855	1091	18400	1086	18401	1078	18402	1077	18552	1092	18554	1089	18704	1101	18705	1104	18706	1106

The class setting of a node can represent the actual or relative cost of transiting through a location. The default class setting is $7 per meter of cable. Adjustment of this value is possible on the Edit Class form detailed in Appendix C. The key concept of the class variable is the association of a node with the cost of transit. Weapon impact areas or designated off-limit areas will logically entail higher installation costs. Required user inputs include the rectangular coordinates of the affected area and the travel cost or class of the respective area. Note that defined areas are sorted starting from the southwest. Algorithm implementation updates node classification from this starting point to the northeast. The scale of the class/cost input is relative to one meter of cable. Step four or the *Set Node Classification* button sets the class of all nodes.

The user may also assign a penalty for high altitude differentials among adjacent nodes. For example, elevation changes over a 500 meter interval in excess of 100 meters may dictate twice the installation cost compared to level terrain. The cost of node transit accounts for distance multiplied by both class and altitude penalty. Step five incorporates all user modifications into the model. Resulting calculations from these user inputs give

the node number, class setting, X/Y coordinates, Z value, and adjacent node numbers with transit costs in the following form (Anode retains the same function while Cost indicates the cost of transit):

Node	Class	X	Y	Z	Anode	Cost	Anode	Cost	Anode	Cost	Anode	Cost	Anode	Cost	Anode	Cost	Anode	Cost
2532	7.00	5495	39385	693	2379	4950	2380	3514	2381	4953	2531	3512	2533	3503	2683	4960	2684	3506	2685	4950

Commands on the Terrain Data form require sequential execution. Exceptions or input errors are handled with appropriate message prompts to the user. GIS data are stored in the file '500.txt'. Node classification data and the network model are stored in the files 'Class.txt' and 'Network.txt', respectively. Direct modification of these text files is not recommended.

B. NETWORK CALCULATIONS

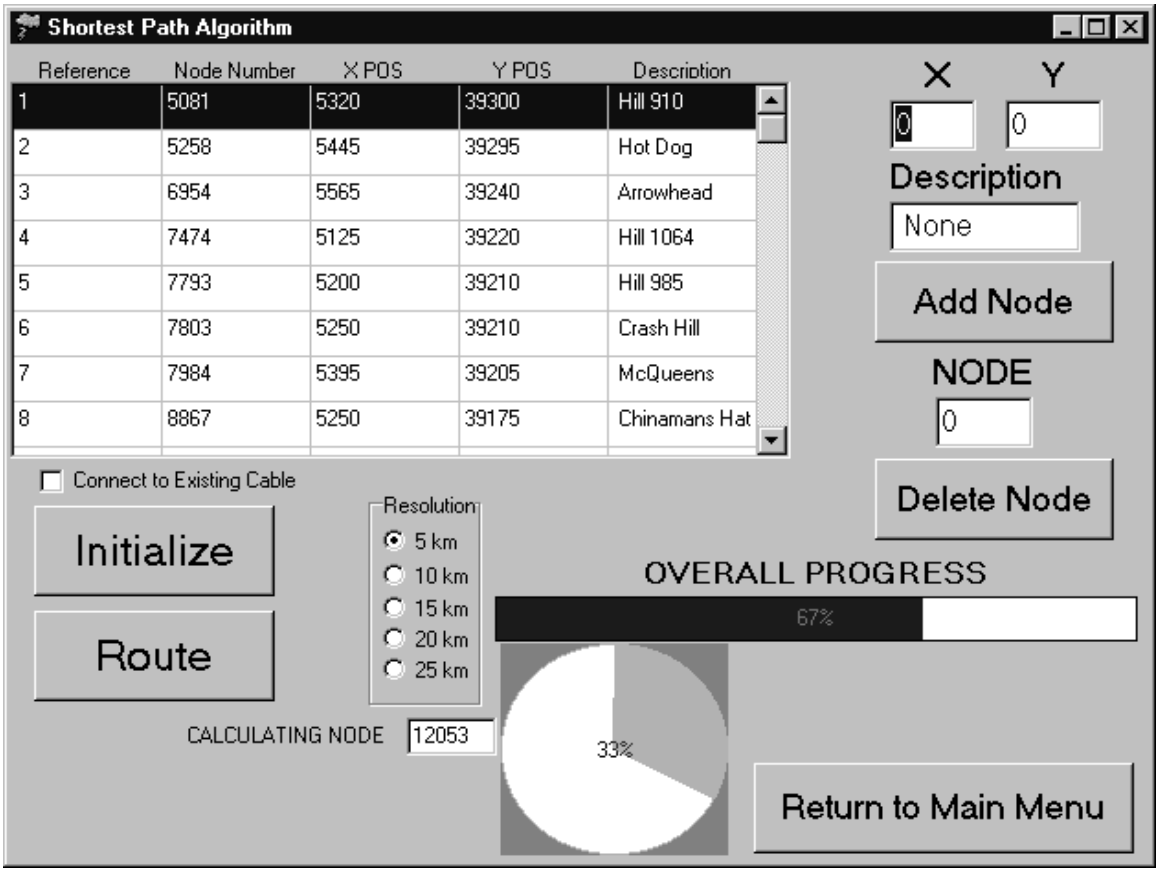


Figure 18. Shortest Path Window.

Selection of the *SPATH* button on the main form brings the user to the form for shortest path algorithm calculations shown in Figure 18. Existing AAR sites provide demand node locations. These sites require both high speed and large volume data transmission capabilities. The user enters the location for each demand node in the appropriate X and Y input boxes. The *Add Node* button inputs these data and identifies the associated node number. Note that demand node entries must be rounded to 500 meter increments. Input of a node number and *Delete Node* removes the selected node. The user has the option to connect to the existing cable shown by a check in the desired box. This option activates nodes on the existing cable with a transit cost of zero. Also, the user selects a search resolution for the program run. Higher resolution requires more system resources, but it will ensure connectivity. The resolution parameter aids in variable reduction during route calculation. Basically, the shortest path algorithm terminates at the given resolution distance.

The *Initialization* button loads network model information into RAM. The structure of the object loaded into RAM is statically set. A known network size eliminates the need for pointer references. The source code reference to this object is defined as GRAPH in the path unit. Attributes of this object include 21,000 elements of NODE type (a different object), size, start, finish, row count, and column count. Start and finish maintain the boundary values of demand locations. The stored value is the first or last node number, respectively. Row count and column count store the dimensions of the rectangular shape. More complex are the 21,000 objects of NODE type. A NODE object is described by the attributes number, X position, Y position, marked, eight objects of ARC type, predecessor, and total cost attributes. The number uniquely identifies the node. X and Y position attributes are self explanatory. The marked attribute is a boolean variable (true or false) required in the shortest path algorithm. Predecessor stores the node number of the closest or cheapest adjacent node in a path. The total cost attribute maintains the value of transit from a particular node to

the node in question. The object ARC contains node number and cost attributes. Both attributes describe equivalent traits as the NODE object, but for an adjacent node. Seemingly complex, this structure allows for recursive identification in RAM instead of resource consuming searches for adjacent node information.

The *Route* button executes the shortest path algorithm for each demand node to all other demand nodes. Results from this process are bi-directional arcs for all demand nodes. Storage of these arcs is in the 'Data' sub-directory 'arcs.txt' file. Progress bars indicate the status of program execution for each demand node calculation and overall computation. The *Return to Main Menu* button delivers the user back to the program control shell or main form. Run time for this process at a 20 kilometer resolution on a Pentium Pro 200Mhz computer is under 60 seconds.

The MST button on the main form delivers the user to the Minimum Spanning Tree form. Sequential execution is required on this form. The *Initialize* button loads the modified network model into RAM. The GRAPH object, detailed in the tree unit, now contains different attributes consisting of 3,800 objects of NODE type, paths, and size. The paths attribute stores the number of arcs under scrutiny. Size maintains the number of demand locations. The object NODE is described by number, X position, Y position, 100 objects of ARC type, predecessor, length, marked, used, status, and total cost attributes. X/Y position, predecessor, marked, and total cost attributes retain the same function. Length stores the number of nodes in an arc that allows for recursive calculations in the algorithm. A boolean attribute is used to indicate arc usage. The status attribute identifies the arc as new or existing. The ARC object is described by node number, X/Y location, and predecessor. All ARC object attributes follow logical precedent.

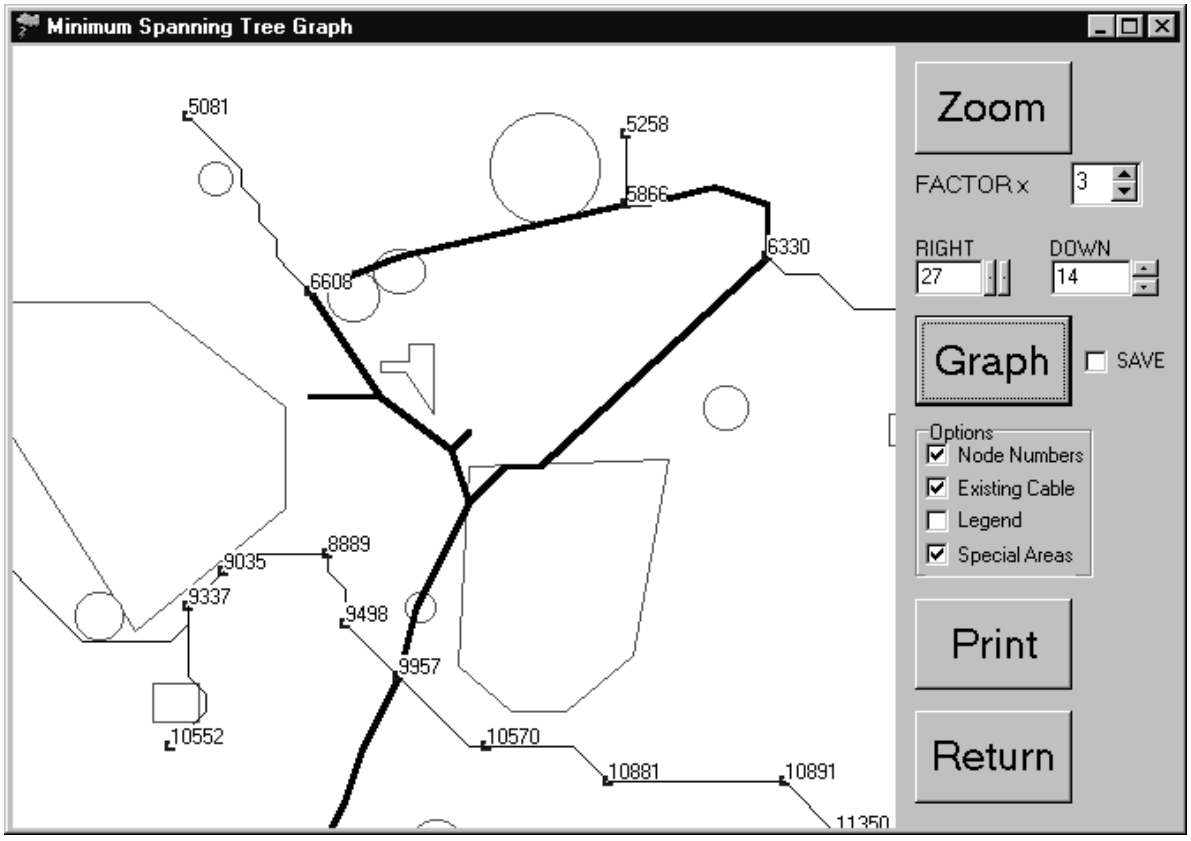


Figure 19. Minimum Spanning Tree Graph.

Selection of the *View Tree* button delivers the user to the form shown in Figure 19. This form presents the graphical result of the previous calculations. In this example, demand nodes tie into the existing cable. Additionally, off-limit areas are avoided in new cable routes. The user has the option for viewing node numbers, existing cable, legend, and special area data. Note that all program outputs are normalized for spatial accuracy in this graphical environment. This spatial accuracy is continued in the *Print* button by automatically adjusting output to paper size. The user also has the option to save the scenario, thereby storing the picture in the 'Data' sub-directory as 'graphic.bmp'. Also, the file 'tree.txt' in the 'Data' sub-directory contains the tree arcs and summary information. Lastly, the user may *Zoom* the graphical view with associated horizontal and vertical control of the representation.

C. NETWORK RESULTS

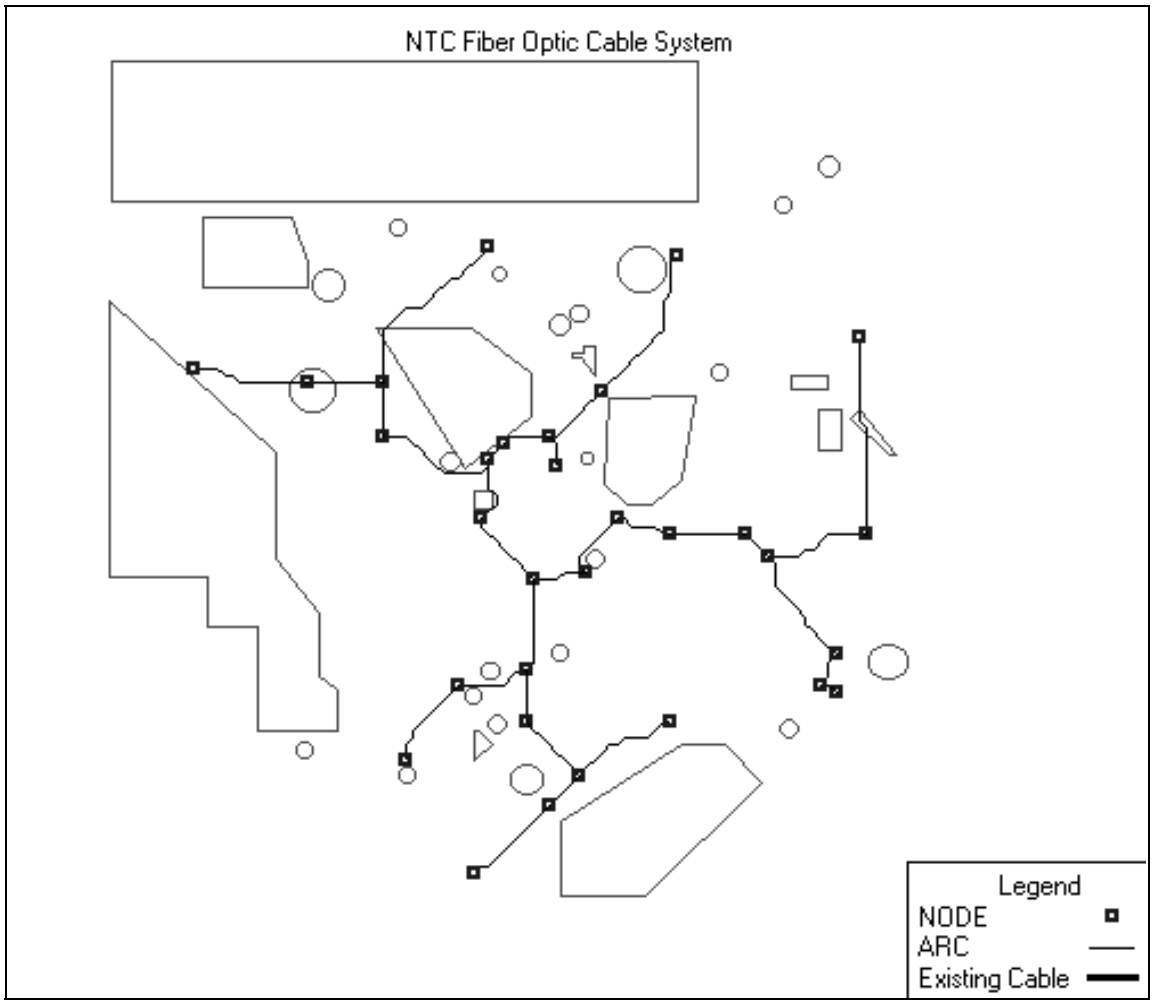


Figure 20. New Cable Network.

Figure 20 presents the results of a 20 kilometer resolution program run without the option to connect to the existing cable. Costs of $7 per meter for normal terrain and $20 for impact or off-limit areas were assigned, respectively. Geometric shapes portray off-limit or impact areas. Notice the cable routing generally avoids these areas except when a much greater distance occurs. This follows logically from differential cost of transit. The total cost of this tree is $1,199,447.00.

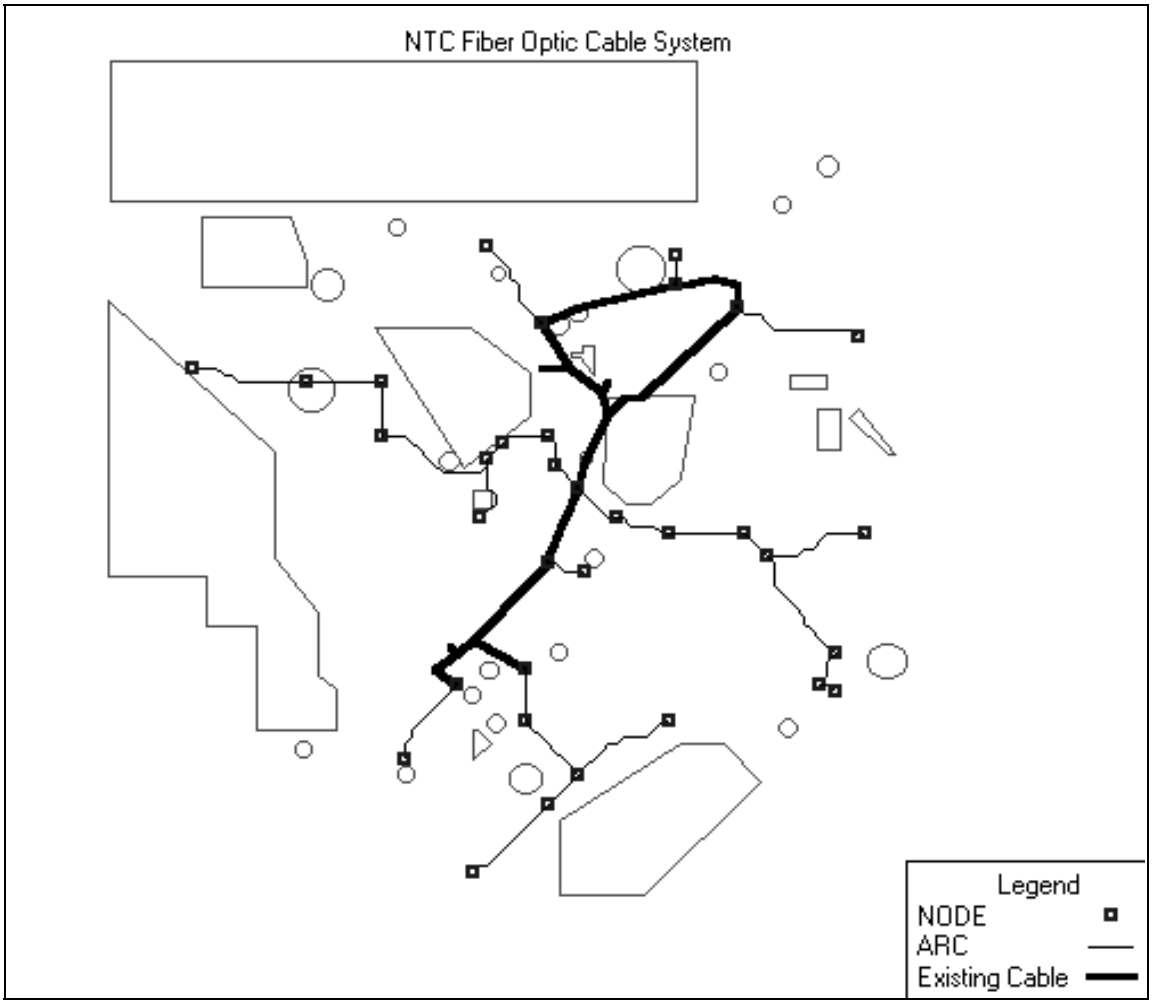


Figure 21. New Cable Network with Existing Cable Linkage.

Figure 21 presents further refinement to the analysis on cable routing. Adding zero cost demand nodes along the pre-existing cable forces linkage to new cable nodes. Under the same scenario as Figure 19, this program run delivers a total cost of $909,178.00. The cheaper cost follows common sense, considering the ability to tie into existing cable, thereby providing shorter distances.

The purpose of this program is not to provide a complete solution to designing a fiber optic cable system for the NTC, yet it does provide a powerful planning tool. The user has the ability to consider multiple scenarios for cable emplacement with relative cost feedback within minutes. Design scenarios will build in complexity and accuracy

over a few iterations of this process. The user will add demand nodes to further refine the routing of the cable according to experience. The design scenarios and actual installation for the cable system will be incremental; however, the final design must address future requirements with flexibility.

VII. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

This research developed two methodologies for the NTC. The first methodology focused on efficient encapsulation of data emissions. The structure of the database utilized in this process is both simple and robust. Simplicity comes from separation of objective and subjective data elements into different tables or relations. The database structure is robust in that it is platform independent. Given the objective data encapsulation structure, this research focused further on opportunities to effectively quantify training performance. The analysis of objective data will not provide indisputable metrics; but, they will provide insight to trends and certain events.

The second methodology presented in this effort relied on a computer-aided analysis of NTC data transmission requirements. The goal of this methodology sought an efficient means for comparing multiple designs of a fiber optic cable system. Central to this methodology were the modeling of terrain, demand location identification, and obtaining a tractable solution in terms of computer runtime. The modeling of terrain involved characterization of pertinent factors to cable emplacement including distance, altitude, and site specific cost manipulations. Demand locations for the cable system mirrored current AAR site use. The end result of this methodology provided a computer application giving the user the ability to rapidly manipulate design factors with immediate graphical and numerical feedback. This computer tool accurately portrays over 3,000 square kilometers to a resolution of 500 meters.

The strength of these methodologies lies in the direction they provide the NTC. Data encapsulation and transmission techniques are explosive in terms of technological improvement. New and improved software and hardware will introduce great capability; yet, the core concept of data encapsulation and transmission must remain simple and

robust. This will allow the user to manipulate technology for specific structured goals instead of receiving transparent improvements that are disconnected.

B. RECOMMENDATIONS

Recommendations for future study encompass two distinct areas. Improvement of the database structure requires further development of relations pertinent to the assessment of training. The addition of objective data is relatively simple. Either modify an existing relation or develop a new relation with a unique attribute. Adding subjective data requires some caution, but it offers unlimited refinement of MOEs. Subjective MOEs require unique attributes such as Task Force identification. Future MOEs will combine objective data with subjective inputs. The ability of combining digital audio or video with subjective inputs is also possible. Probably most important to implementation of this database structure is ensuring platform independence of the final product.

The true goal of data encapsulation is providing accurate feedback to training units. To this end, the data derived from a rotation must be archived in a format interpretable to other organizations. The best case in this situation allows the training unit use of the data. Integration of present NTC Take-Home Package products with the database of the rotation is the true answer. This new product allows the unit the ability to execute queries according to its own training plan and assessment.

Future study efforts for the fiber optic cable system entail development of a 'downward looking' antenna system. Also, integration of proposed land expansion property should be included into the model. Assuming continued growth in data transmission requirements and the loss of bandwidth, fiber optic cable emplacement is the only solution to the NTC transmission problem.

The NTC Route Optimizer currently gives quick feedback on a cable emplacement plan. The program may be improved by the addition of several functions. Rotational data on cable damage provides the basis for introducing reliability concepts

into the model. The cost differential between laser pulse emitters and the type of cable would provide more accurate cost data analysis by the program. Lastly, upgrading the resolution of the model to 100 meters would provide greater accuracy at the price of system resource utilization.

APPENDIX A. SAMPLE RELATIONS

DCI_ID	PID	NTC_Bumper#	Unit_Bumper#	Icon#	PlatformType	UNIT_ID1	TIME_ID1
1	FA23	300	A100	S11A	M1	1	1200
2	23AA	200	A101	S12A	M1	1	1200
3	AE45	100	A102	S13A	M1	1	1200
4	9843	500	A103	S14A	M1	1	1200
5	AEFD	1200	A104	S21A	M1	2	1200
6	8739	2100	A105	S22A	M1	2	1200
7	1100	1800	A106	S23A	M1	2	1200
8	A3D3	900	A107	S24A	M1	2	1200
9	34AB	1500	A108	S31A	M1	3	1200
10	43B6	400	A109	S32A	M1	3	1200
11	E347	1900	A110	S33A	M1	3	1200
12	5344	301	A111	S34A	M1	3	1200

Table 4. PLAYER Relation.

UNIT_ID	Platoon	Company	Battalion	TF_ID1	TIME_ID2
1	1	A	1/61	1	1200
2	2	A	1/61	1	1200
3	3	A	1/61	1	1200
4	HQ	A	1/61	1	1200
5	1	B	1/61	6	1200
6	2	B	1/61	6	1200
7	3	B	1/61	6	1200
8	HQ	B	1/61	6	1200
9	1	C	1/61	3	1200
10	2	C	1/61	3	1200
11	3	C	1/61	3	1200

Table 5. UNIT Relation.

TF_ID	CO/TM	BN/TF	TIME_ID3
1	A	1/61	1200
2	B	1/61	1205
3	C	1/61	1210
4	HHC	1/61	1200
5	A	1/52	1205
6	B	1/52	1210
7	C	1/52	1210
8	HHC	1/52	1210

Table 6. TF Relation.

FIRE_ID	DCI_ID1	Position/Location	Velocity	Acceleration	Hull/ TurretAngle	Weapon Type	RoundType	TIME_ID4
1	793	NK 694438	3.17	-0.15	-18.57	105MM	Heat	1200
2	1935	NK 751507	7.74	0.06	84.16	AT5	ATGM	1200
3	1490	NK 729265	5.96	-2.20	44.12	120MM	Heat	1200
4	333	NK 671440	1.33	-2.66	-59.97	TOW	ATGM	1201
5	1289	NK 719225	5.16	-1.96	26.05	25MM	Sabot	1201
6	1564	NK 732958	6.26	-1.18	50.77	105MM	Sabot	1201
7	1485	NK 729031	5.94	-2.10	43.70	105MM	Sabot	1201
8	1400	NK 724787	5.60	0.36	36.06	AT5	ATGM	1201
9	1896	NK 749583	7.59	-0.13	80.69	120MM	Sabot	1202
10	1283	NK 718909	5.13	-2.10	25.48	TOW	ATGM	1202
11	138	NK 661670	0.55	-1.36	-77.55	25MM	Sabot	1202
12	897	NK 699643	3.59	-1.75	-9.20	105MM	Heat	1202
13	107	NK 660106	0.43	-2.44	-80.37	AT5	ATGM	1202
14	1753	NK 742405	7.01	-0.42	67.77	120MM	Heat	1203
15	371	NK 673331	1.49	-1.81	-56.56	TOW	ATGM	1203

Table 7. FIRE Relation.

HIT_ID	DCI_ID2	FID	Position/ Location	Velocity	Acceleration	AspectAngle	Outcome	CODE_ID1	TIME_ID5
1	365	1634	NK 673013	3.17	-0.15	-24.27	M	1	1200
2	604	1395	NK 684960	7.74	0.06	18.74	NM	2	1200
3	1960	39	NK 752768	5.96	-2.20	262.85	C	3	1200
4	1881	118	NK 748822	1.33	-2.66	248.65	K	4	1201
5	1736	263	NK 741574	5.16	-1.96	222.55	C	1	1201
6	1308	691	NK 720201	6.26	-1.18	145.61	NM	2	1201
7	1895	104	NK 749505	5.94	-2.10	251.10	C	3	1201
8	225	1774	NK 666013	5.60	0.36	-49.47	C	4	1201
9	720	1279	NK 690783	7.59	-0.13	39.71	NM	1	1202
10	1826	173	NK 746089	5.13	-2.10	238.80	C	2	1202
11	955	1044	NK 702552	0.55	-1.36	82.07	K	3	1202
12	1465	534	NK 728010	3.59	-1.75	173.74	M	4	1202
13	1661	338	NK 737807	0.43	-2.44	208.99	K	1	1202
14	867	1132	NK 698130	7.01	-0.42	66.15	NM	2	1203
15	1327	672	NK 7241134	1.49	-1.81	148.97	K	3	1203

Table 8. HIT Relation.

_ID	DCI_ID3	Position/Location		PU BIT	TIME_ID6
1	1634	NK	694438	1	1200
2	1395	NK	751507	1	1200
3	39	NK	729265	1	1200
4	118	NK	671440	1	1200
5	263	NK	719225	1	1200
6	691	NK	732958	1	1200
7	104	NK	729031	1	1200
8	1774	NK	724787	1	1200
9	1279	NK	749583	1	1200
10	173	NK	718909	1	1200

Table 9. DCI STATUS Relation.

CODE_ID	WeaponType	PlatformType	EffectiveRange
1	120MM	M1	3000
2	25MM	M2	1000
3	105MM	T80	1500
4	AT5	BMP	3000
5	TOW	M2	3750

Table 10. KILLCODE Relation.

PAIRING_ID	DCI_ID_FIRE	FID	WeaponType	Range	Outcome	TIME_ID7
1	462	218	105MM	1793.684	M	1200
2	1698	366	AT5	2935.056	NM	1200
3	1044	1367	120MM	2490.229	C	1200
4	101	1263	TOW	1333.719	K	1201
5	682	64	25MM	2289.412	C	1201
6	668	1055	105MM	2564.086	NM	1201
7	947	282	105MM	2485.538	C	1201
8	258	159	AT5	2400.652	C	1201
9	703	688	120MM	2896.579	NM	1202
10	676	431	TOW	2283.108	C	1202
11	35	688	25MM	1138.329	K	1202
12	662	199	105MM	1897.774	M	1202
13	66	253	AT5	1107.04	K	1202
14	708	215	120MM	2753.024	NM	1203
15	10	568	TOW	1371.541	K	1203

Table 11. PAIRING Relation.

APPENDIX B. SAMPLE NTC ROUTE OPTIMIZER INPUT

Demand Node Report				
Ref Number	**Node**	**X**	**Y**	**Name**
1	5081	5320	39300	Hill 910
2	5258	5445	39295	Hot Dog
3	6954	5565	39240	Arrowhead
4	7474	5125	39220	Hill 1064
5	7793	5200	39210	Hill 985
6	7803	5250	39210	Crash Hill
7	7984	5395	39205	McQueens
8	8867	5250	39175	Chinamans Hat
9	8889	5360	39175	Goat Trail
10	9035	5330	39170	OP Brown
11	9337	5320	39160	Prey Cut
12	9498	5365	39155	Old Harry
13	10552	5315	39120	Black Bassalt
14	10570	5405	39120	Hill 876
15	10881	5440	39110	Hill 780
16	10891	5490	39110	Hill 760
17	10907	5570	39110	Hill 720
18	11350	5505	39095	Hill 781
19	11630	5385	39085	Millers Hole
20	11775	5350	39080	BDE Hill
21	13335	5550	39030	FP 200
22	13598	5345	39020	Bike Lake
23	13893	5300	39010	Dust Bowl
24	13941	5540	39010	OP Bone
25	14095	5550	39005	Bug Pit
26	14662	5345	38985	Bowling Alley
27	14681	5440	38985	Words Wadi
28	15406	5265	38960	TV Hill
29	15733	5380	38950	OP Two
30	16337	5360	38930	OP One
31	17695	5310	38885	Hill 899

Table 12. Demand Nodes.

Node Classification Report 7.00 Default Class Model Dimensions and Repective Influence					
Start		Finish		Class	AREA
5068	39080	5168	39164	20	GoldStone 3
5068	39164	5120	39210	20	GoldStone 4
5070	39330	5458	39424	20	Leach Lake
5130	39273	5195	39320	20	Gary Owen
5166	38980	5220	39007	20	GoldStone 1
5166	39007	5220	39059	20	GoldStone 2
5188	39189	5219	39220	20	Nelson Lake
5192	38960	5205	38972	20	O/L
5203	39263	5225	39285	20	Mclean Lake
5255	39220	5310	39247	20	Nelson 3
5255	39306	5267	39318	20	Desert Spring
5260	38944	5273	38958	20	O/L
5288	39152	5302	39164	20	Restricted
5290	39170	5329	39250	20	Nelson 2
5304	38996	5316	39008	20	Golf Course
5305	39154	5318	39170	20	Nelson 1
5310	38966	5314	38980	20	Impact Area
5310	39127	5323	39138	20	Combo
5315	39013	5328	39027	20	O/L
5320	38977	5333	38990	20	Garlic Spring
5323	39277	5333	39287	20	910
5335	38936	5357	38957	20	Langford Well
5360	39241	5375	39254	20	Drinkwater Spr
5361	39024	5374	39036	20	O/L
5370	38870	5475	38920	20	Langford 1
5376	39249	5388	39263	20	O/L
5380	39215	5390	39235	20	Box
5382	39155	5391	39164	20	O/L
5384	39086	5398	39099	20	O/L
5398	39130	5444	39200	20	Lucky Fuse
5406	39269	5438	39301	20	O/L
5410	38930	5490	38970	20	Langford 2
5467	39210	5480	39233	20	O/L
5510	39322	5522	39334	20	O/L
5513	38974	5526	38987	20	Bitter Spring
5520	39206	5545	39215	20	O/L
5538	39345	5553	39360	20	O/L
5538	39165	5562	39192	20	O/L
5572	39012	5598	39037	20	Red Pass Lake

Table 13. Node Classification.

APPENDIX C. SAMPLE NTC ROUTE OPTIMIZER SESSION

The intent of this section is to give the reader basic navigational skills inside the NTC Route Optimizer program.

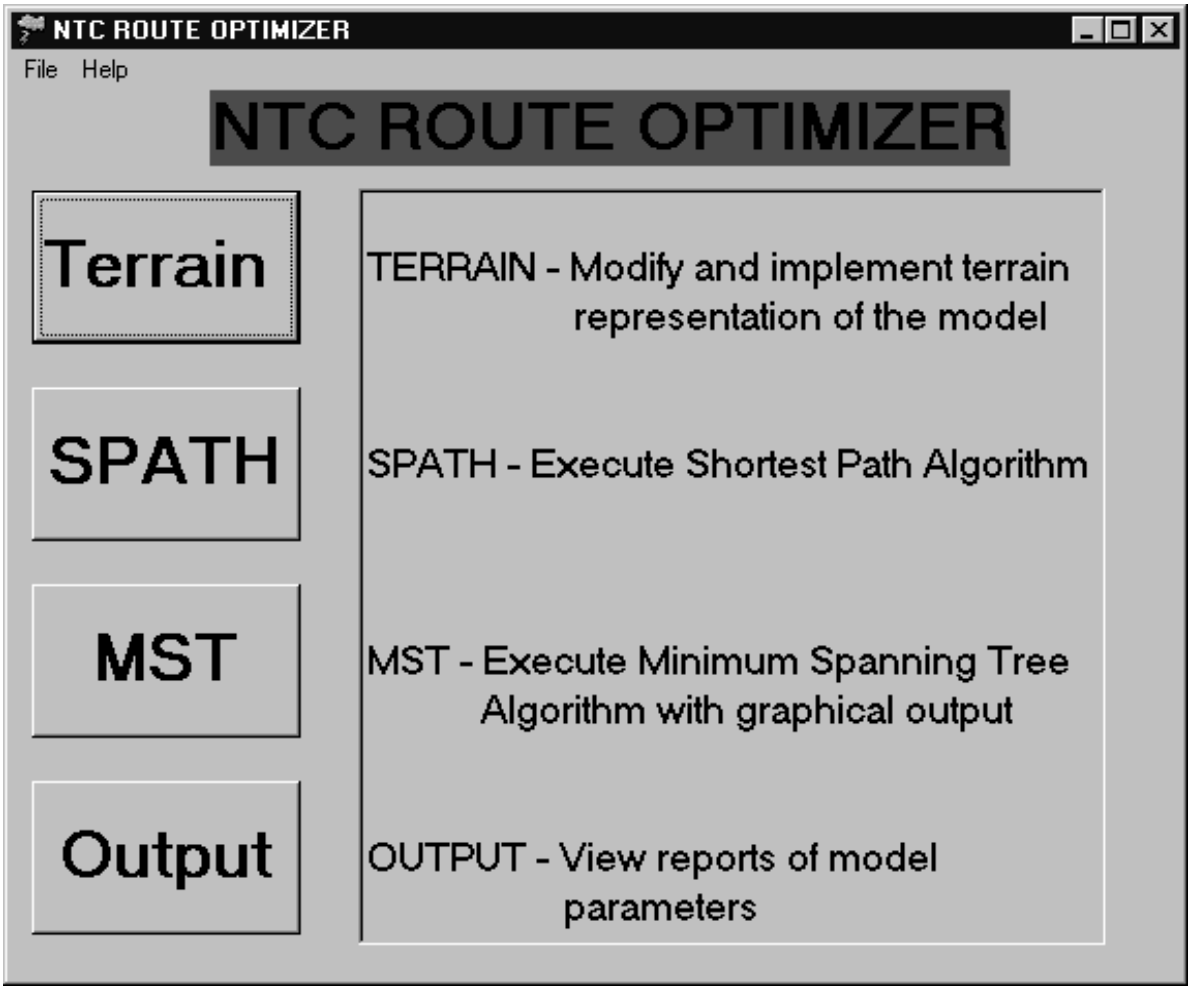


Figure 22. Main Window.

Figure 22 shows the main window or form of the program. Each button, for example *Terrain*, executes a certain command. In this case, each button activates another form.

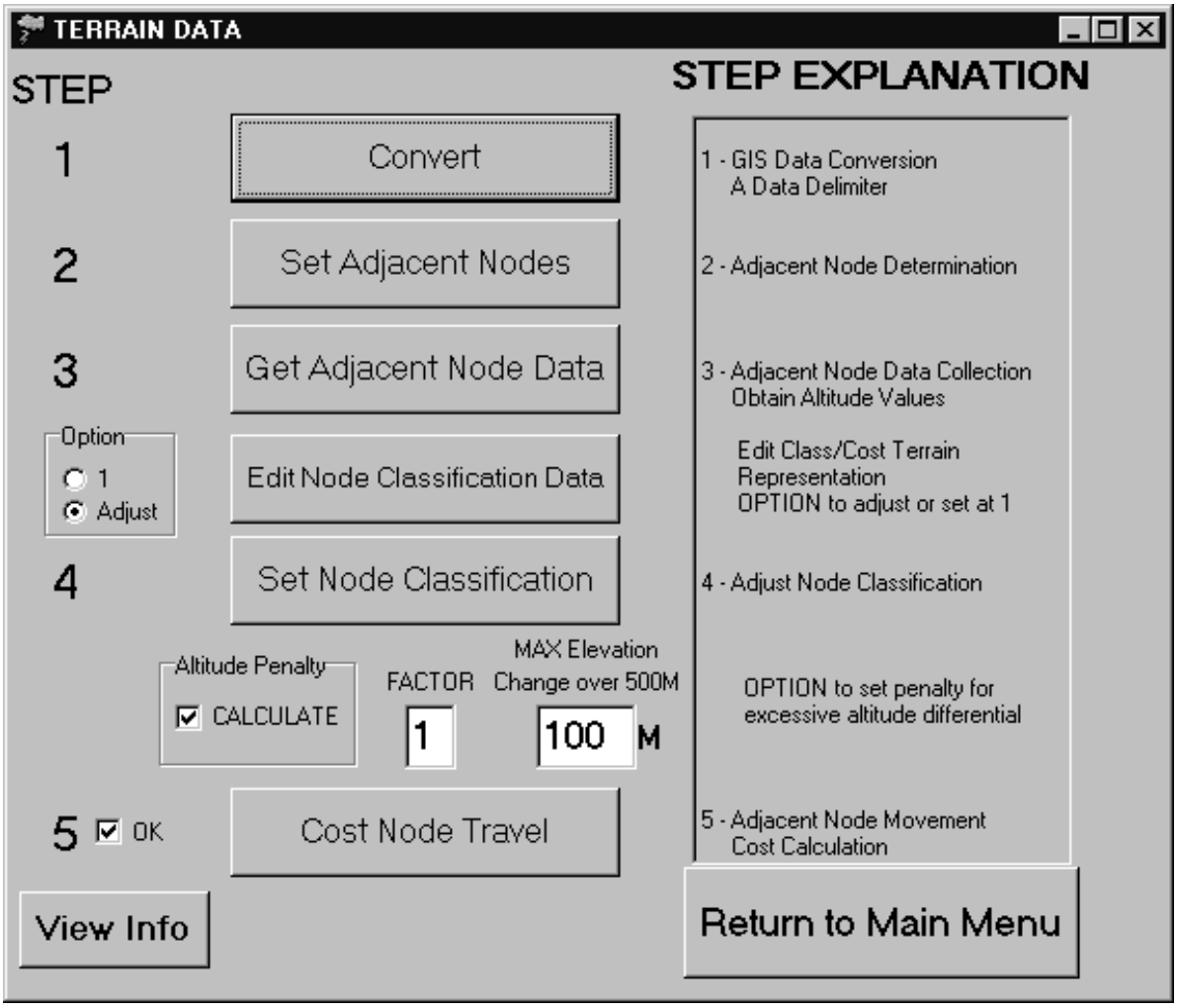


Figure 23. Terrain Data Form.

The *Terrain* button on the main form activates the Terrain Data form shown in Figure 23. There are five steps with several options in the process of data manipulation. Step 1 converts raw data into integer form with a default classification. Step 2 sets adjacent node values. Step 3 determines adjacent node altitude parameters. The next button activates a new form discussed in the next paragraph. The option allows for adjusting node classifications or setting them all to one. Step 4 completes this process. If checked, altitude penalty incurs a weighted penalty for high altitude differential between adjacent nodes. Factor is that penalty with the MAX Elevation value determining affected areas. Step 5 completes the data manipulation. *The Return to Main Menu* button takes the user back to the main form.

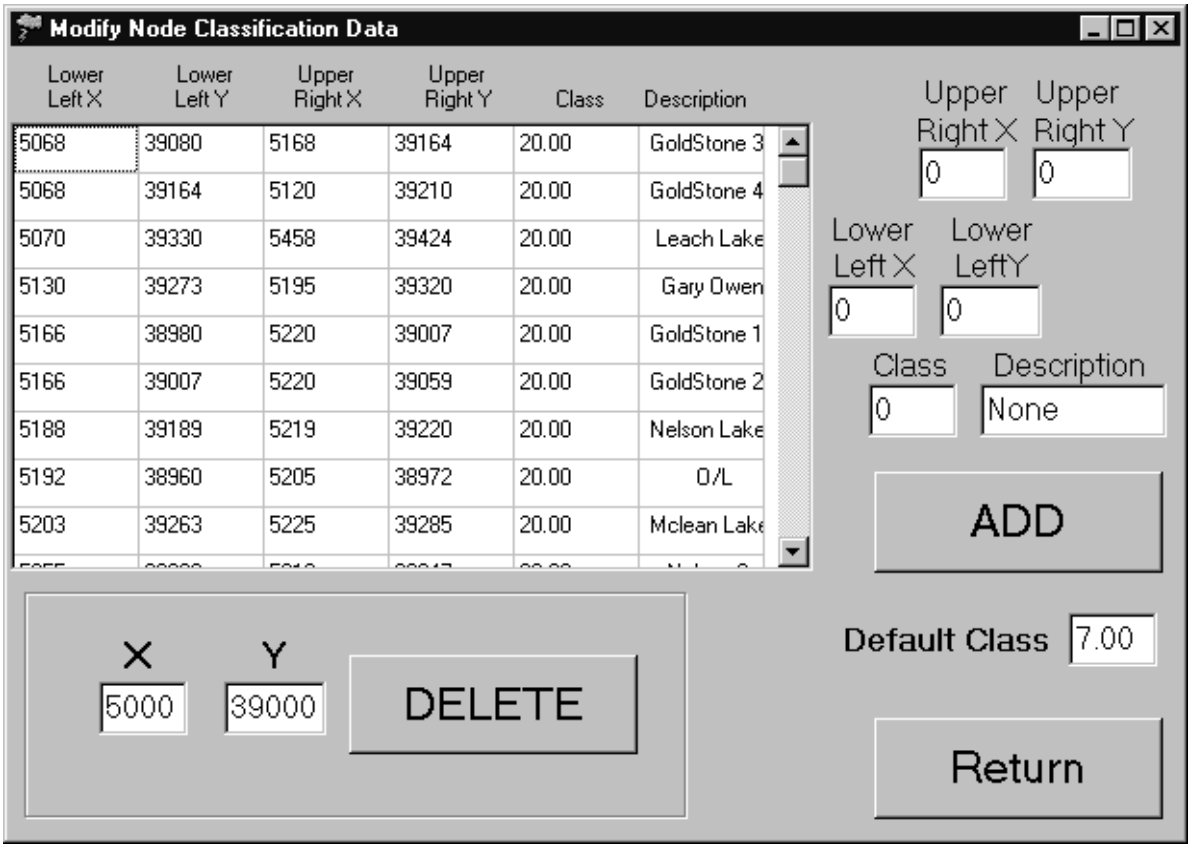


Figure 24. Node Classification Form.

The *Edit Node Classification Data* button on the Terrain Data form delivers the user to the Node Classification form shown in Figure 24. The user may input boundary locations with an associated class and description. Boundary locations are of the form southwest to northeast in universal transverse mercator grid coordinates. The *ADD* button enters the inputs to the data set. Entry of a southwest coordinate and selection of the *DELETE* button removes the input from the data set. Also, the user may modify the default class setting by changing the value and selecting the *ADD* button to update the data set. Note that changing the default class setting is not retained after the program is terminated. The *Return* button takes the user back the Terrain Data form.

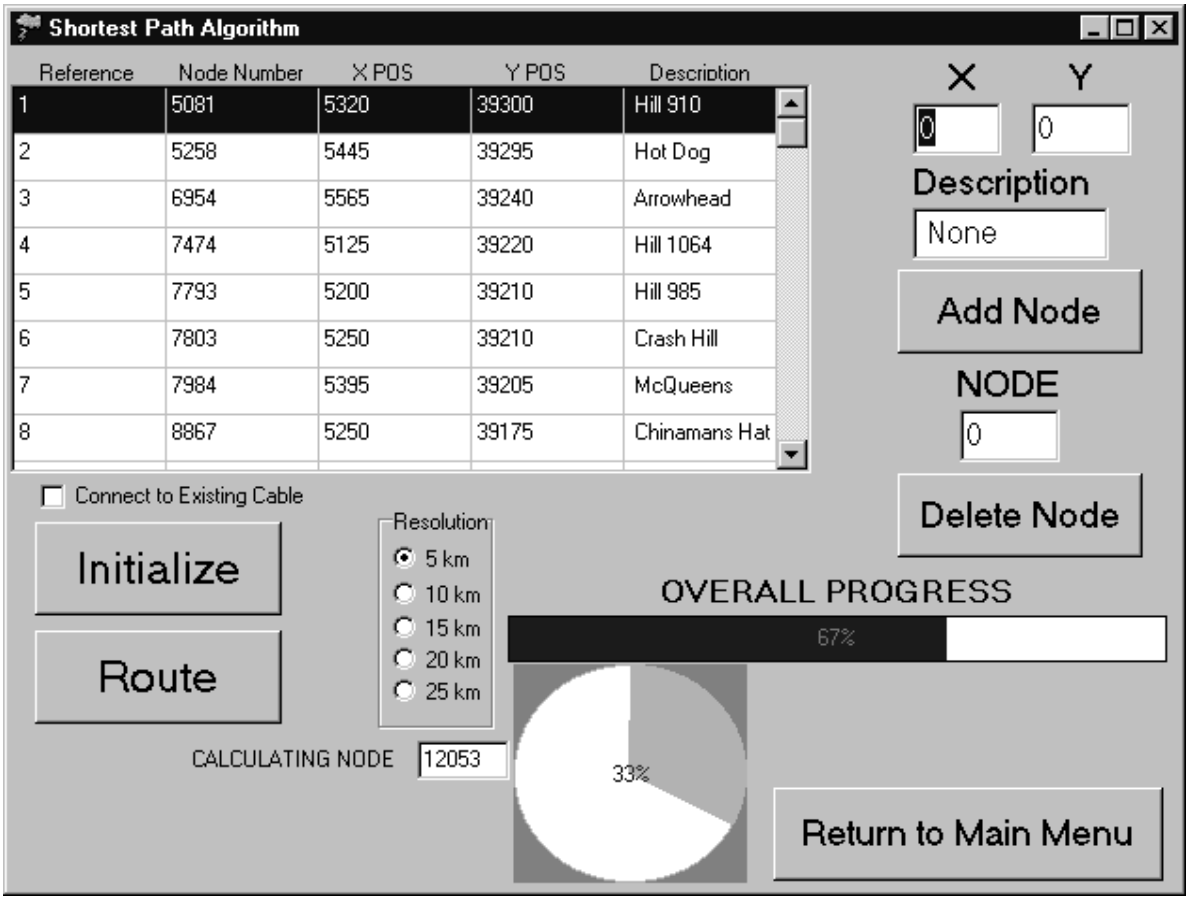


Figure 25. Shortest Path Form.

The *SPATH* button on the main form brings the user the Shortest Path form presented in Figure 25. The *Add Node* button inputs user entries of location and description for new demand nodes. This command automatically determines the node number. *Delete Node* removes a demand node from the data set. *Initialize* loads the model into RAM. If the Connect to Existing Cable option is checked, additional demand nodes are added to the model with a transit cost of zero. The Resolution option determines the amount of variable reduction in upcoming computations. The *Route* button initiates calculations. Progress bars indicate program status. In this case, 67% of overall calculations are complete while computations for node 12053 are 33% complete. The *Return to Main Menu* button takes the user back to the main form.

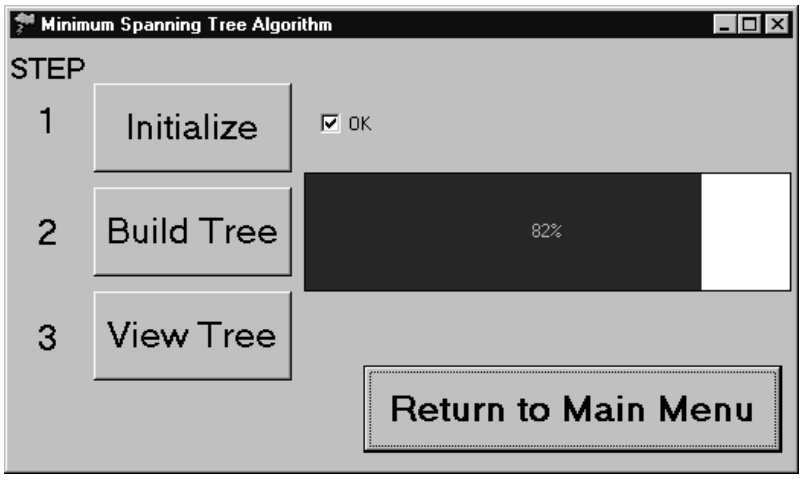


Figure 26. Minimum Spanning Tree Form.

The *MST* button on the main form takes the user to the Minimum Spanning Tree form shown in Figure 26. *Initialize* or Step 1 loads the new network model into RAM. Step 2 executes the calculations. In this example, initialization is complete indicated by the check and 'OK'. Calculations within the algorithm are 82% complete reflected by the progress bar. The *Return to Main Menu* continues the same function. Step 3 delivers the user to the next form.

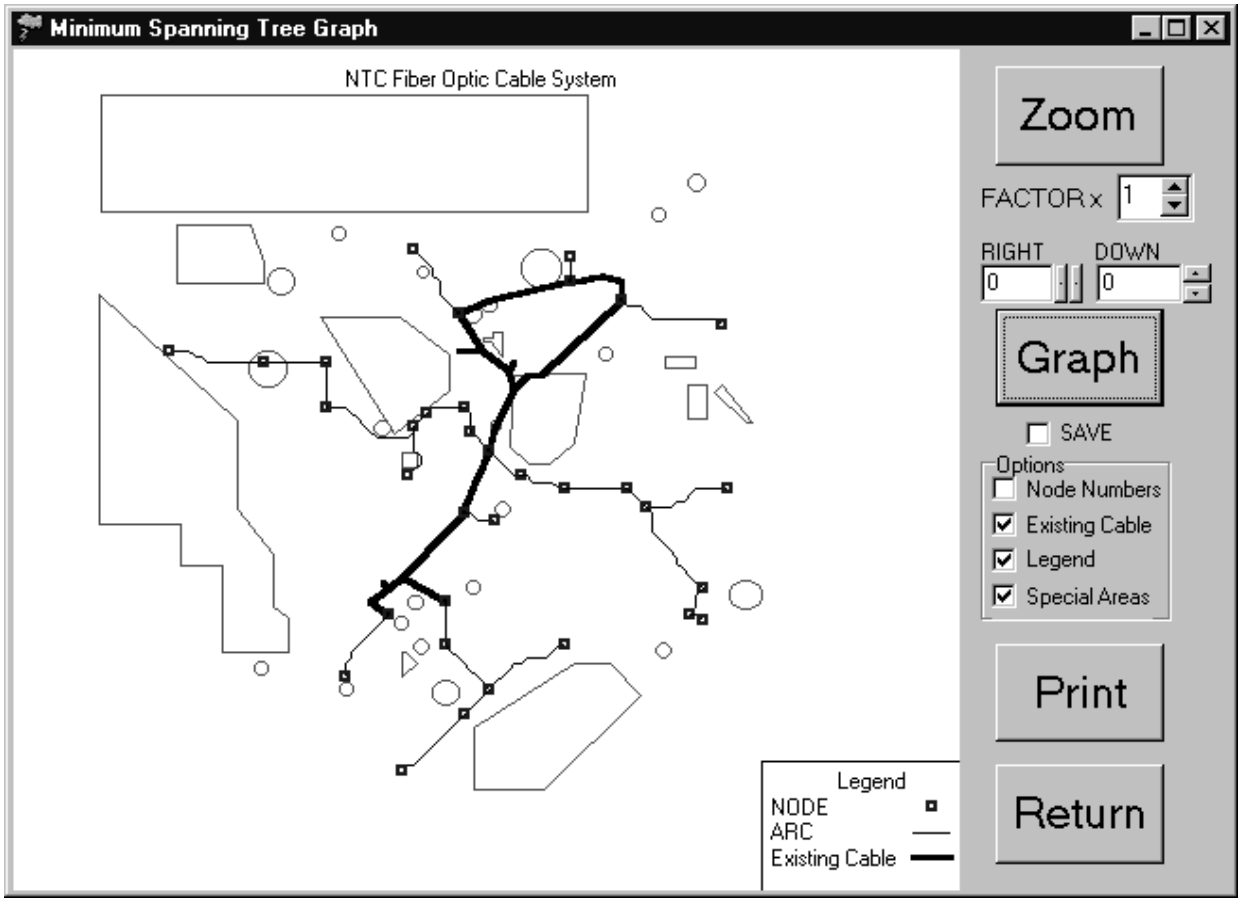


Figure 27. Minimum Spanning Tree Graph Form.

Figure 27 shows the Minimum Spanning Tree Graph form. The user has the option for *Zoom* control along with horizontal and vertical control. The *Graph* button redraws the picture. The Save checkbox archives the picture to the 'Data' sub-directory as 'graphic.bmp'. The user has the option to view node numbers, existing cable, legend, and special areas. Any change in these options requires clicking the *Graph* button to obtain results. The *Print* button sends the graphic to the default printer. Lastly, *Return* delivers the user to the previous form.

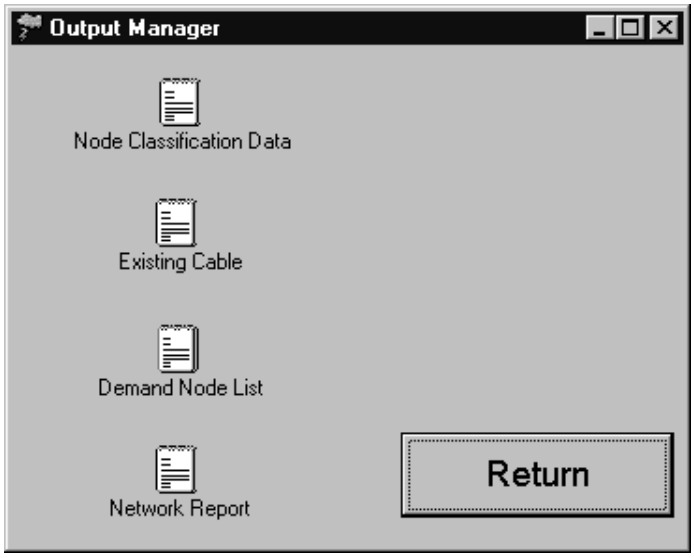
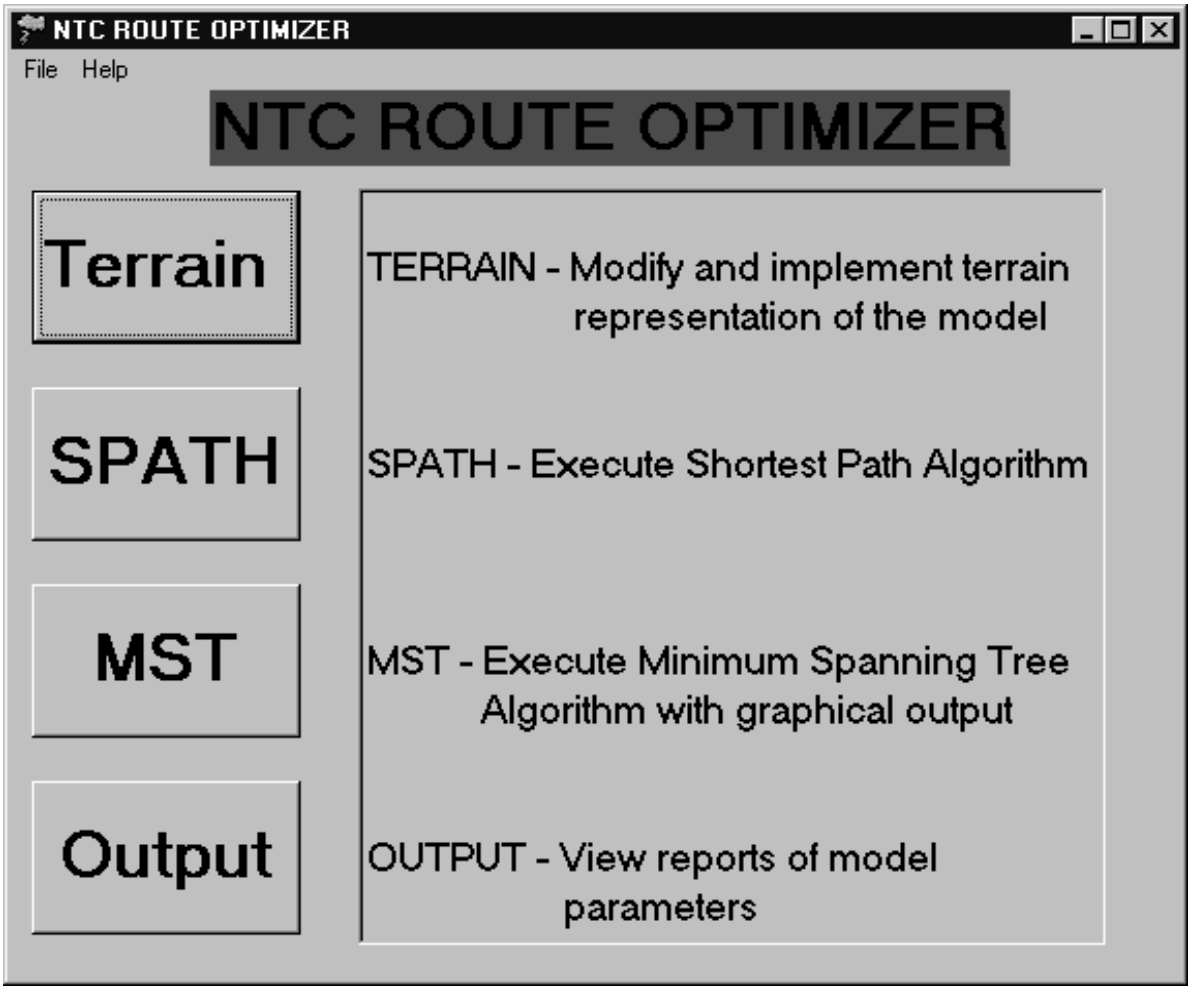


Figure 28. Output Manager Form.

The *Output* button on the main form brings the user to the Output Manager form highlighted in Figure 28. This form allows the user easy access to program data files and generated information. All files referenced in this paragraph exist in the 'Data' sub-directory as text files. The Windows application 'Notepad.exe' views each selected file. *Node Classification Data* takes the user to the file 'Class.txt'. *Existing Cable* opens the file 'Exist.txt'. This file is a series of demand nodes with an eventual transit cost of zero. *Demand Node List* views the same information presented on the Shortest Path form but in text format. Lastly, the *Network Report* button opens the file 'Tree.txt'. This file contains all used arcs in the network and summary information.

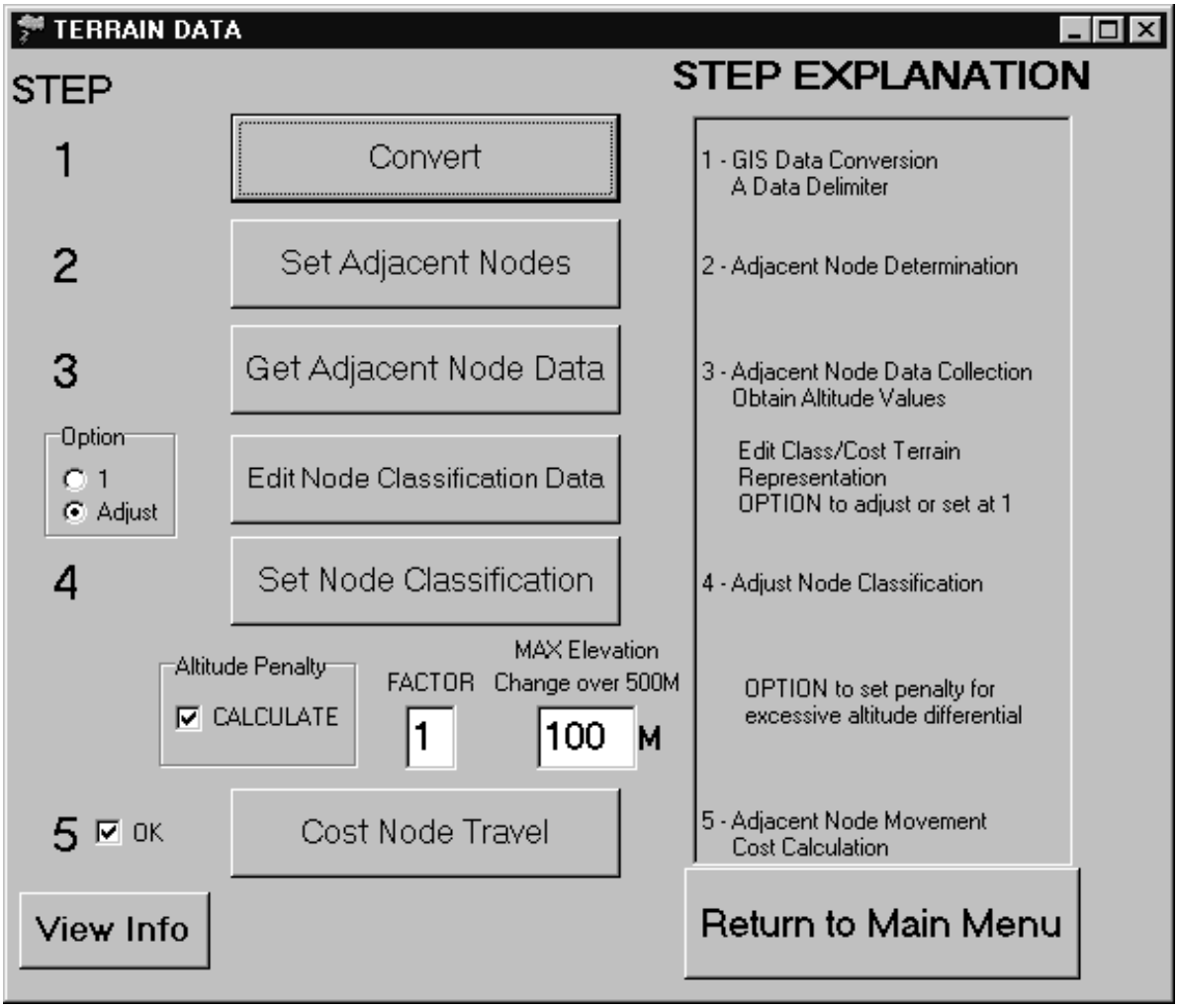
APPENDIX D. NTC ROUTE OPTIMIZER SOURCE CODE




```
{$M 16384, 16777216}

unit NTC;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Grids, Outline, DirOutln, Menus, StdCtrls, ExtCtrls;

type
  TMAIN = class(TForm)
    MainMenu1: TMainMenu;
    Main1: TMenuItem;
    Help1: TMenuItem;
    Exit1: TMenuItem;
    Terrain: TButton;
    SPATH: TButton;
    MST: TButton;
```

```
    Output: TButton;
    About1: TMenuItem;
    ListBox1: TListBox;
    Label1: TLabel;
    procedure TerrainClick(Sender: TObject);
    procedure SPATHClick(Sender: TObject);
    procedure MSTClick(Sender: TObject);
    procedure OutputClick(Sender: TObject);
    procedure Exit1Click(Sender: TObject);
    procedure About1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  MAIN: TMAIN;

implementation

uses PATH, TREE, NodeClass, Data, Output;

{$R *.DFM}

procedure TMAIN.TerrainClick(Sender: TObject);
begin
  mstmain.close;
  DATAMAIN.SHOW;
end;

procedure TMAIN.SPATHClick(Sender: TObject);
begin
  mstmain.close;
  PATHMAIN.SHOW;
end;

procedure TMAIN.MSTClick(Sender: TObject);
begin
  MSTMAIN.SHOW;
end;

procedure TMAIN.OutputClick(Sender: TObject);
begin
  mstmain.close;
  outputform.show;
end;

procedure TMAIN.Exit1Click(Sender: TObject);
begin
  main.close;
end;

procedure TMAIN.About1Click(Sender: TObject);
```

```
begin
  showmessage('                    NTC Route Optimizer' + #13 +
          'Enhanced Planning Tool for Fiber Optic Cable Placement' + #13 +
          '                    ver 1.0b 1 Jan 97');
end;

procedure TMAIN.FormCreate(Sender: TObject);
begin
  listbox1.clear;
  listbox1.visible:=true;
  listbox1.items.add(' ');
  listbox1.items.add('TERRAIN - Modify and implement terrain');
  listbox1.items.add('              representation of the model');
  listbox1.items.add(' ');
  listbox1.items.add(' ');
  listbox1.items.add('SPATH - Execute Shortest Path Algorithm');
  listbox1.items.add(' ');
  listbox1.items.add(' ');
  listbox1.items.add(' ');
  listbox1.items.add('MST - Execute Minimum Spanning Tree');
  listbox1.items.add('      Algorithm with graphical output');
  listbox1.items.add(' ');
  listbox1.items.add(' ');
  listbox1.items.add('OUTPUT - View reports of model');
  listbox1.items.add('              parameters');
end;

end.
```




```
{$M 16384, 16777216}

unit DATA;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, ComCtrls, Gauges, ExtCtrls;
const W = 8;   {Precision on output}
     WW= 4;
     D = 1;   {Decimal precision}
{POSN is just a simple position, ANODE is an adjacent NODE, NODE is self explanatory,
Graph encompasses the network}
type ANode = OBJECT

              private
                Number    : Integer;
                PositionX : Integer;
                PositionY : LongInt;
             end;
```

```
type Node = OBJECT

            private
              NodeNumber: Integer;
              PositionX : Integer;
              PositionY : LongInt;
              Arc       : Array[1..8] of ANode;   {adjacent nodes}
              Cost      : Single;
            end;


type
  TDATAMAIN = class(TForm)
    Button1: TButton;
    Convert: TButton;
    SetAdj: TButton;
    GetAdj: TButton;
    SetNodeClass: TButton;
    CostAdjNode: TButton;
    fileGauge: TGauge;
    editclass: TButton;
    CheckBox1: TCheckBox;
    CheckBox2: TCheckBox;
    CheckBox3: TCheckBox;
    CheckBox4: TCheckBox;
    CheckBox5: TCheckBox;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label6: TLabel;
    Label7: TLabel;
    ListBox1: TListBox;
    Button2: TButton;
    Option: TRadioGroup;
    RadioButton1: TRadioButton;
    RadioButton2: TRadioButton;
    RadioGroup1: TRadioGroup;
    Edit1: TEdit;
    Edit2: TEdit;
    Label5: TLabel;
    Label8: TLabel;
    Label9: TLabel;
    CheckBox6: TCheckBox;
    Label10: TLabel;
    Label11: TLabel;
    procedure Button1Click(Sender: TObject);
    procedure ConvertClick(Sender: TObject);
    procedure SetAdjClick(Sender: TObject);
    procedure GetAdjClick(Sender: TObject);
    procedure SetNodeClassClick(Sender: TObject);
    procedure AdjNodeClassClick(Sender: TObject);
    procedure CostAdjNodeClick(Sender: TObject);
    procedure editclassClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
```

```
    procedure Button2Click(Sender: TObject);


  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  DATAMAIN: TDATAMAIN;

implementation

uses ntc, NodeClass;

{$R *.DFM}

procedure TDATAMAIN.Button1Click(Sender: TObject);
begin
   checkbox1.visible:=false;
   checkbox2.visible:=false;
   checkbox3.visible:=false;
   checkbox4.visible:=false;
   checkbox1.checked:=false;
   checkbox2.checked:=false;
   checkbox3.checked:=false;
   checkbox4.checked:=false;
   DATAMAIN.close;
end;

procedure TDATAMAIN.ConvertClick(Sender: TObject);
{Takes GIS output (with commas removed) and converts X,Y,Z to integer values.
 Stadardizes grid reference - locks to gridlines see rounding algorithms
 Sets standard node class to 1 -                                          }

   var X, Y, Z               : REAL;
      Number, count          : Integer;
      Data, Outfile          : textfile;      {Output File}

   begin
   label10.visible:=false;
   listbox1.visible:=false;
   checkbox2.checked:=false;
   checkbox2.visible:=false;
   checkbox3.checked:=false;
   checkbox3.visible:=false;
   checkbox4.checked:=false;
   checkbox4.visible:=false;
   checkbox5.checked:=false;
   checkbox5.visible:=false;
   filegauge.visible:=true;
   filegauge.update;
   Assignfile(Data,'data\500.txt');
   reset(Data);
```

```
  assignfile(Outfile,'data\Network.txt');
  rewrite(Outfile);
  count:=0;
  writeln(Outfile,20672,' ',136,' ',152);     {hardwired}
  writeln(Outfile,'   Node  Class    X        Y      Z');
  while not eof (Data) do begin
    read(Data,Number);
    read(Data,X);
    read(Data,Y);
    read(Data,Z);
    X:=round(X/100);     {following are align grid system to standard}
    Y:=round(Y/100)+1;   {1km 1000,*10 500m 100 +1}
    Z:=round(Z);
    writeln(Outfile,Number:W,' ',1:WW,' ',X:W:0,' ',Y:W:0,' ',Z:WW:0);
    filegauge.progress:=trunc(count/20672 *100);
    inc(count);
  end;
  filegauge.visible:=false;
  closefile(Data);
  closefile(Outfile);
  checkbox1.checked:=true;
  checkbox1.visible:=true;
  end;

procedure TDATAMAIN.SetAdjClick(Sender: TObject);

{Takes ConvertNode results - selects adjacent nodes - note boundary conditions       }

  var i, Number, Class1, X,Z              : Integer;
      Y                                   : LongInt;
      NumberNodes,Rows,Columns,ColumnEnd,
      ColumnStart,RowEnd,
      RowCounter                          : Integer;
      TL,TC,TR,L,R,BL,BC,BR               : Integer;
      Data,Outfile                        : textfile;       {Output File}

  begin
  label10.visible:=false;
  listbox1.visible:=false;
  if checkbox1.checked<>true then
    begin
      showmessage('Sequential Execution Required');
    end
  else
  begin
  checkbox1.checked:=false;
  checkbox1.visible:=false;
  checkbox3.checked:=false;
  checkbox3.visible:=false;
  checkbox4.checked:=false;
  checkbox4.visible:=false;
  checkbox5.checked:=false;
  checkbox5.visible:=false;
  filegauge.visible:=true;
  filegauge.update;
```

```
Assignfile(Data,'data\Network.txt');
reset(Data);
assignfile(Outfile,'data\Buffer.txt');
rewrite(Outfile);
readln(Data,NumberNodes,Rows,Columns);
readln(Data);
writeln(Outfile,'Number of Nodes  Rows  Columns');
writeln(Outfile,NumberNodes,' ',Rows,' ',Columns);
ColumnStart:=1;
ColumnEnd:=Columns;
RowEnd:=Rows;
RowCounter:=0;
writeln(Outfile,'    Node Class    X       Y     Z    TL   TC   TR    L    R   BL   BC   BR');
for i:=1 to NumberNodes do
  begin
    filegauge.progress:=trunc(i/numberNodes *100);
    readln(Data,Number,Class1,X,Y,Z);
    TL:=Number-Columns-1;
    TC:=Number-Columns;
    TR:=Number-Columns+1;
    L:=Number-1;
    R:=Number+1;
    BL:=Number+Columns-1;
    BC:=Number+Columns;
    BR:=Number+Columns+1;
    IF Number<=Columns then       {top row}
     begin
       TL:=0;
       TC:=0;
       TR:=0;
     end;
    IF Number=ColumnEnd then     {end of column nodes}
      begin
        ColumnEnd:=ColumnEnd+Columns;
        TR:=0;
        R:=0;
        BR:=0;
      end;
    If Number=ColumnStart then      {start of column nodes}
      begin
        RowCounter:=RowCounter+1;
        ColumnStart:=ColumnStart+Columns;
        BL:=0;
        L:=0;
        TL:=0;
      end;
    If RowEnd=RowCounter then          {last row}
      begin
        BL:=0;
        BC:=0;
        BR:=0;
      end;
    writeln(Outfile,Number:W,' ',Class1:WW,' ',X:W,' ',Y:W,' ',Z:WW,' ',TL:WW,' ',TC:WW,' ',
      TR:WW,' ',L:WW,' ',R:WW,' ',BL:WW,' ',BC:WW,' ',BR:WW);
  end;
```

```
  closefile(Data);
  closefile(Outfile);
  filegauge.visible:=false;
  checkbox2.checked:=true;
  checkbox2.visible:=true;
  end;
  end;

procedure TDATAMAIN.GetAdjClick(Sender: TObject);

{Takes SetAdjNode results - finds adjacent node altitude data from database      }

  var i                              : Integer;
     NumberNodes,Rows,Columns            : Integer;
     TLN,TCN,TRN,LN,RN,BLN,BCN,BRN       : Integer;
     Infile, Outfile: textfile;       {Files}
     Number, Class1, X, Z     : Array [0..21000] of Integer;
     Y                        : Array [0..21000] of LongInt;   {hardwired to size of network}
     TL,TC,TR,L,R,BL,BC,BR    : Array [1..21000] of Integer;

  begin
  label10.visible:=false;
  listbox1.visible:=false;
  if checkbox2.checked<>true then
    begin
      showmessage('Sequential Execution Required');
    end
  else
  begin
  checkbox1.checked:=false;
  checkbox1.visible:=false;
  checkbox2.checked:=false;
  checkbox2.visible:=false;
  checkbox4.checked:=false;
  checkbox4.visible:=false;
  checkbox5.checked:=false;
  checkbox5.visible:=false;
  filegauge.visible:=true;
  filegauge.update;
  filegauge.progress:=0;
  Assignfile(Infile,'data\buffer.txt');
  reset(Infile);
  assignfile(Outfile,'data\Network.txt');
  rewrite(Outfile);
  readln(Infile);
  writeln(Outfile,'Number of Nodes  Rows  Columns');
  readln(Infile,NumberNodes,Rows,Columns);
  writeln(Outfile,NumberNodes,' ',Rows,' ',Columns);
  readln(Infile);
  write(Outfile,'     Node Class    X        Y     Z       TL        TC        TR');
  writeln(Outfile,'         L        R         BL        BC        BR');
  for i:=1 to NumberNodes do
    begin
      readln(Infile,Number[i],Class1[i],X[i],Y[i],Z[i],TL[i],TC[i],TR[i],
        L[i],R[i],BL[i],BC[i],BR[i]);
```

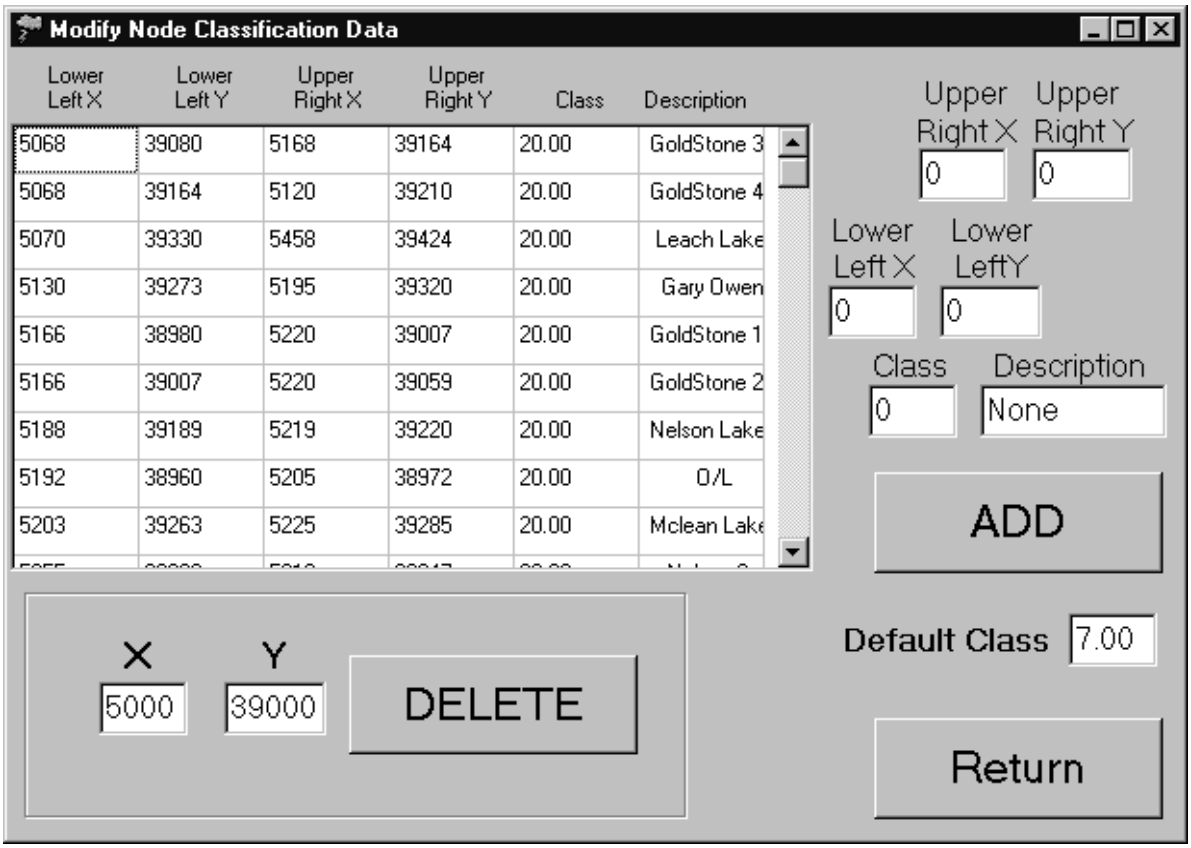
```
    end;
  Z[0]:=0;
  for i:=1 to NumberNodes do
    begin
      filegauge.progress:=trunc(i/numberNodes *100);
      TLN:=TL[i];
      TCN:=TC[i];
      TRN:=TR[i];
      LN:=L[i];
      RN:=R[i];
      BLN:=BL[i];
      BCN:=BC[i];
      BRN:=BR[i];
      writeln(Outfile,Number[i]:W,' ',Class1[i]:WW,' ',X[i]:W,' ',Y[i]:W,' ',Z[i]:WW,' ',
        TL[i]:WW,' ',Z[TLN]:WW,' ',TC[i]:WW,' ',Z[TCN]:WW,' ',TR[i]:WW,' ',Z[TRN]:WW,' ',
        L[i]:WW,' ',Z[LN]:WW,' ',R[i]:WW,' ',Z[RN]:WW,' ',BL[i]:WW,' ',Z[BLN]:WW,' ',
        BC[i]:WW,' ',Z[BCN]:WW,' ',BR[i]:WW,' ',Z[BRN]:WW);
    end;
  closefile(Infile);
  closefile(Outfile);
  filegauge.visible:=false;
  checkbox3.checked:=true;
  checkbox3.visible:=true;
  end;
  end;

procedure TDATAMAIN.SetNodeClassClick(Sender: TObject);

 {Adjusts Node Classification in the database  }

  var i,ii,Finish                          : Integer;
     Number,NumberNodes,Rows,Columns,X,Z          : Integer;
     Y                                    : LongInt;
     OldClass                             : Single;
     Data,DataClass,Outfile                       : Textfile;
     AdjN1,Z1,AdjN2,Z2,AdjN3,Z3,AdjN4,Z4,AdjN5,Z5,
       AdjN6,Z6,AdjN7,Z7,AdjN8,Z8                 : Integer;
     POSN,LLX,URX            : Array[1..100] of integer;    {hardwired - 8 class distinctions}
     LLY,URY                 : Array[1..100] of LongInt;
     Class1                  : Array[1..100] of single;

  begin
  label10.visible:=false;
  listbox1.visible:=false;
  if checkbox3.checked<>true then
    begin
      showmessage('Sequential Execution Required');
    end
  else
  begin
  checkbox2.checked:=false;
  checkbox2.visible:=false;
  checkbox3.checked:=false;
  checkbox3.visible:=false;
  checkbox1.checked:=false;
```

```
  checkbox1.visible:=false;
  checkbox5.checked:=false;
  checkbox5.visible:=false;
  Assignfile(DataClass,'data\Class.txt');
  reset(DataClass);
  readln(Dataclass);
  readln(Dataclass);
  readln(Dataclass);
  Assignfile(Data,'data\network.txt');
  reset(Data);
  Assignfile(Outfile,'data\buffer.txt');
  rewrite(Outfile);
  filegauge.visible:=true;
  filegauge.update;
  i:=1;
  while not eof (DataClass) do
    begin
      readln(DataClass,LLX[i],LLY[i],URX[i],URY[i],Class1[i]);
      inc(i);
    end;
  Finish:=i;
  readln(Data);
  writeln(Outfile,'Number of Nodes  Rows  Columns');
  readln(Data,NumberNodes,Rows,Columns);
  writeln(Outfile,NumberNodes,' ',Rows,' ',Columns);
  readln(Data);
  write(Outfile,'     Node Class    X       Y     Z      TL       TC       TR');
  writeln(Outfile,'       L       R        BL       BC       BR');
  for i:=1 to NumberNodes do
    begin
      filegauge.progress:=trunc(i/numberNodes *100);
      readln(Data,Number,OldClass,X,Y,Z,AdjN1,Z1,AdjN2,Z2,AdjN3,Z3,AdjN4,Z4,AdjN5,Z5,
      AdjN6,Z6,AdjN7,Z7,AdjN8,Z8);
    for ii:=1 to finish do                    {check all class distinctions}
      begin
        if(X >= LLX[ii]) and (X <= URX[ii]) and (Y >= LLY[ii]) and (Y <= URY[ii]) then
          begin
            OldClass:=Class1[ii];
          end;
      end;
      if radiobutton1.checked= true then OldClass:=1.0;
      writeln(Outfile,Number:W,' ',OldClass:WW,' ',X:W,' ',Y:W,' ',Z:WW,' ',AdjN1:WW,' ',Z1:WW,' ',
      AdjN2:WW,' ',Z2:WW,' ',AdjN3:WW,' ',Z3:WW,' ',AdjN4:WW,' ',Z4:WW,' ',AdjN5:WW,'
',Z5:WW,' ',
      AdjN6:WW,' ',Z6:WW,' ',AdjN7:WW,' ',Z7:WW,' ',AdjN8:WW,' ',Z8:WW);
    end;
  closefile(Data);
  closefile(DataClass);
  closefile(Outfile);
  filegauge.visible:=false;
  checkbox4.checked:=true;
  checkbox4.visible:=true;
  end;

 end;
```

```
procedure TDATAMAIN.AdjNodeClassClick(Sender: TObject);

var Test:Node;

begin

end;

procedure TDATAMAIN.CostAdjNodeClick(Sender: TObject);

{Takes GetAdjNodeData results - calculates cost (distance/pythagorean and class)       }

  var i,ii                                  : Integer;
     Number, X                              : Integer;
     NumberNodes,Rows,Columns               : Integer;
     Class1                                 : Single;
     maxalt,altpen,penalty,Up,Down,Left,Right     : Integer;
     Y,Zterm                                : LongInt;
     Z                                      : Real;
     Data, Outfile                          : textfile;  {Output File}
     AdjNode: Array[1..8] of Integer;
     ZAdj   : Array[1..8] of Real;
     Dist   : Array[1..8] of LongInt;

  begin
  label10.visible:=false;
  listbox1.visible:=false;
  if checkbox4.checked<>true then
    begin
      showmessage('Sequential Execution Required');
    end
  else
  begin
  maxalt:=strtointdef(edit2.text,0);
  altpen:=strtointdef(edit1.text,0);
  if (maxalt=0) OR (altpen=0) THEN
    begin
      maxalt:=1000;
      altpen:=1;
      showmessage('Invalid Penalty Entry - NO CALCULATION!');
    end;
  checkbox2.checked:=false;
  checkbox2.visible:=false;
  checkbox3.checked:=false;
  checkbox3.visible:=false;
  checkbox4.checked:=false;
  checkbox4.visible:=false;
  checkbox1.checked:=false;
  checkbox1.visible:=false;
  Assignfile(Data,'data\buffer.txt');
  reset(Data);
  assignfile(Outfile,'data\network.txt');
  rewrite(Outfile);
  readln(Data);
```

```
writeln(Outfile,'Number of Nodes  Rows  Columns');
readln(Data,NumberNodes,Rows,Columns);
writeln(Outfile,NumberNodes,' ',Rows,' ',Columns);
readln(Data);
write(Outfile,'     Node Class    X       Y     Z      TL       TC       TR');
writeln(Outfile,'       L       R       BL       BC       BR');
filegauge.visible:=true;
filegauge.update;
for i:=1 to NumberNodes do
  begin
    filegauge.progress:=trunc(i/numberNodes *100);
    read(Data,Number,Class1,X,Y,Z);
    readln(Data,AdjNode[1],ZAdj[1],AdjNode[2],ZAdj[2],AdjNode[3],ZAdj[3],
      AdjNode[4],ZAdj[4],AdjNode[5],ZAdj[5],AdjNode[6],ZAdj[6],
      AdjNode[7],ZAdj[7],AdjNode[8],ZAdj[8]);
    for ii:=1 to 8 do
      begin
        Up:=Number-Columns;
        Down:=Number+Columns;
        Right:=Number+1;
        Left:=Number-1;
        if (Z-ZAdj[ii]>maxalt) and (checkbox6.checked=true)
          then penalty:=altpen else penalty:=1;
        if (AdjNode[ii]=Up) or (AdjNode[ii]=Down) or
          (AdjNode[ii]=Right) or (AdjNode[ii]=Left) then
          begin
                {hardwired - adjust for network size}
            Dist[ii]:=round(Class1*(sqrt((Z-ZAdj[ii])*(Z-ZAdj[ii])+(500*500))));
          end
        else
          begin
                {hardwired - adjust for network size}
            Dist[ii]:=round(Class1*(sqrt((Z-ZAdj[ii])*(Z-ZAdj[ii])+(707*707))));
          end;
        if (AdjNode[ii]=0) then    {calculate distance only if arc exists}
          begin
            Dist[ii]:=0;
          end;
      end;
    writeln(Outfile,Number:W,' ',Class1:WW:2,' ',X:W,' ',Y:W,' ',round(Z):WW,' ',
    AdjNode[1]:WW,' ',Dist[1]:WW,' ',AdjNode[2]:WW,' ',Dist[2]:WW,' ',
    AdjNode[3]:WW,' ',Dist[3]:WW,' ',AdjNode[4]:WW,' ',Dist[4]:WW,' ',
    AdjNode[5]:WW,' ',Dist[5]:WW,' ',AdjNode[6]:WW,' ',Dist[6]:WW,' ',
    AdjNode[7]:WW,' ',Dist[7]:WW,' ',AdjNode[8]:WW,' ',Dist[8]:WW);
  end;
closefile(Data);
closefile(Outfile);
filegauge.visible:=false;
checkbox5.checked:=true;
checkbox5.visible:=true;
Assignfile(Data,'data\buffer.txt');
rewrite(Data);
closefile(Data);
end;
showmessage('Terrain Model Complete' + #13 + 'Data stored in \data\Network.txt');
```

```
  end;

procedure TDATAMAIN.editclassClick(Sender: TObject);
begin
  label10.visible:=false;
  listbox1.visible:=false;
  checkbox5.checked:=true;
  checkbox5.visible:=false;
  classform.show;
end;

procedure TDATAMAIN.FormCreate(Sender: TObject);
begin
  checkbox5.checked:=true;
  button2.click;
  end;

procedure TDATAMAIN.Button2Click(Sender: TObject);
begin
  listbox1.clear;
  LABEL10.VISIBLE:=TRUE;
  listbox1.visible:=true;
  listbox1.items.add(' ');
  listbox1.items.add('1 - GIS Data Conversion');
  listbox1.items.add('     A Data Delimiter');
  listbox1.items.add(' ');
  listbox1.items.add(' ');
  listbox1.items.add('2 - Adjacent Node Determination');
  listbox1.items.add('     ');
  listbox1.items.add(' ');
  listbox1.items.add(' ');
  listbox1.items.add('3 - Adjacent Node Data Collection');
  listbox1.items.add('     Obtain Altitude Values');
  listbox1.items.add(' ');
  listbox1.items.add('      Edit Class/Cost Terrain ');
  listbox1.items.add('      Representation');
  listbox1.items.add('      OPTION to adjust or set at 1');
  listbox1.items.add(' ');
  listbox1.items.add(' ');
  listbox1.items.add('4 - Adjust Node Classification');
  listbox1.items.add(' ');
  listbox1.items.add(' ');
  listbox1.items.add(' ');
  listbox1.items.add('       OPTION to set penalty for ');
  listbox1.items.add('       excessive altitude differential');
  listbox1.items.add(' ');
  listbox1.items.add(' ');
  listbox1.items.add(' ');
  listbox1.items.add('5 - Adjacent Node Movement');
  listbox1.items.add('     Cost Calculation');
 end;

end.
```




```
unit NodeClass;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, Grids;

type
  TClassform = class(TForm)
    classGrid: TStringGrid;
    URYedit: TEdit;
    LLYedit: TEdit;
    CLASSTedit: TEdit;
    URXedit: TEdit;
    LLXedit: TEdit;
    ADD: TButton;
    t1: TLabel;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    Label7: TLabel;
    Label8: TLabel;
    Label9: TLabel;
```

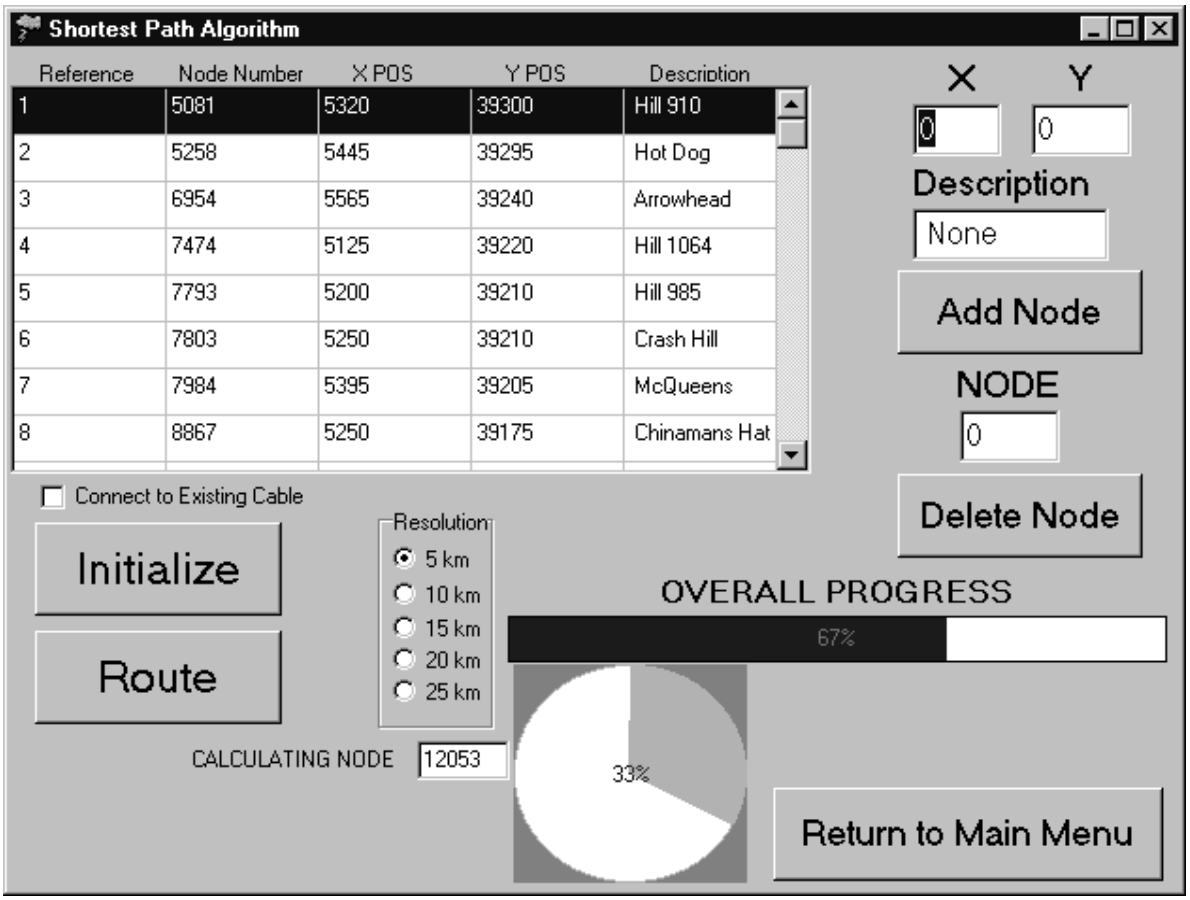
```
    Label10: TLabel;
    Label11: TLabel;
    Label12: TLabel;
    Label15: TLabel;
    Label16: TLabel;
    Label14: TLabel;
    Label17: TLabel;
    GroupBox1: TGroupBox;
    DELETE: TButton;
    delXedit: TEdit;
    DELYedit: TEdit;
    previous: TButton;
    Label18: TLabel;
    Label13: TLabel;
    Label19: TLabel;
    Label20: TLabel;
    DescEdit: TEdit;
    Label21: TLabel;
    procedure FormCreate(Sender: TObject);
    procedure ADDClick(Sender: TObject);
    procedure DELETEClick(Sender: TObject);
    procedure previousClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Classform: TClassform;
  Data:textfile;

implementation

{$R *.DFM}
uses ntc;

procedure TClassform.FormCreate(Sender: TObject);

var LLX,LLY,URX,URY: Array[0..100] of longint;
  Class1:array[0..100] of single;
  Desc:array[0..100] of string;
  i,Finish: integer;

begin
   assignfile(Data,'data\class.txt');
   reset(Data);
   readln(Data);
   readln(Data);
   readln(Data);
   LLXedit.text:='0';
   LLYedit.text:='0';
   URXedit.text:='0';
   URYedit.text:='0';
   CLASSTedit.text:='0';
```

```
  DescEdit.text:='None';
  i:=0;
  while not eof (Data) do
    begin
      readln(Data,LLX[i],LLY[i],URX[i],URY[i],Class1[i],Desc[i]);
      inc(i);
    end;
  Finish:=i;
  classgrid.rowcount:=Finish;
  Finish:=i-1;
  for i:=0 to Finish do
    begin
      classgrid.cells[0,i]:=inttostr(LLX[i]);
      classgrid.cells[1,i]:=inttostr(LLY[i]);
      classgrid.cells[2,i]:=inttostr(URX[i]);
      classgrid.cells[3,i]:=inttostr(URY[i]);
      classgrid.cells[4,i]:=floattostrf(Class1[i],fffixed,6,2);
      classgrid.cells[5,i]:=Desc[i];
    end;
 closefile(data);

end;

procedure TClassform.ADDClick(Sender: TObject);

var BLX,BLY,BRX,BRY:LONGINT;
   BClass:single;
   BDesc:string;
   LLX,LLY,URX,URY: Array[0..100] of longint;
   Class1:array[0..100] of single;
   Desc:array[0..100] of string;
   i,count,Finish: integer;

begin
  assignfile(Data,'data\class.txt');
  reset(Data);
  i:=0;
  readln(Data);
  readln(Data);
  readln(Data);
  while not eof (Data) do
    begin
      readln(Data,LLX[i],LLY[i],URX[i],URY[i],Class1[i],Desc[i]);
      inc(i);
    end;
  LLX[i]:=100000;
  closefile(data);
  Finish:=i;
  BLX:=strtointdef(LLXedit.text,0);
  BLY:=strtointdef(LLYedit.text,0);
  BRX:=strtointdef(URXedit.text,0);
  BRY:=strtointdef(URYedit.text,0);
  BCLASS:=strtofloat(CLASSTedit.text);
  BDesc:=DescEdit.text;
  if (BLX=0) OR (BLY=0) OR (BRX=0) OR (BRY=0) OR (BCLASS=1000) THEN
```

```
      begin
        BLX:=100000;
        BLY:=0;
        BRX:=0;
        BRY:=0;
        BCLASS:=1000;
        BDesc:='badnode';
        showmessage('Entry Error - Integer Values Only!');
        Finish:=Finish-1;
      end;
  classgrid.rowcount:=Finish+1;
  count:=0;
  for i:=0 to Finish do        {SORT BY LLX}
    begin
      if BLX<LLX[count] then
      begin
        classgrid.cells[0,i]:=inttostr(BLX);
        classgrid.cells[1,i]:=inttostr(BLY);
        classgrid.cells[2,i]:=inttostr(BRX);
        classgrid.cells[3,i]:=inttostr(BRY);
        classgrid.cells[4,i]:=floattostrf(BClass,fffixed,8,2);
        classgrid.cells[5,i]:=' ' + BDesc;
        BLX:=100000;
      end
      else
      begin
        classgrid.cells[0,i]:=inttostr(LLX[count]);
        classgrid.cells[1,i]:=inttostr(LLY[count]);
        classgrid.cells[2,i]:=inttostr(URX[count]);
        classgrid.cells[3,i]:=inttostr(URY[count]);
        classgrid.cells[4,i]:=floattostrf(Class1[count],fffixed,8,2);
        classgrid.cells[5,i]:=Desc[count];
        inc(count);
      end;
    end;
  assignfile(Data,'data\class.txt');
  rewrite(Data);
  writeln(Data,'Node Classification Report');
  writeln(Data);
  writeln(Data,' Model Dimensions and Repective Influence');
  for i:=0 to finish do
    begin
      writeln(Data,classgrid.cells[0,i]:8,
      classgrid.cells[1,i]:8,classgrid.cells[2,i]:8,
      classgrid.cells[3,i]:8,classgrid.cells[4,i]:8,classgrid.cells[5,i]:12);
    end;
  closefile(data);
end;

procedure TClassform.DELETEClick(Sender: TObject);

var BLX,BLY:LONGINT;
   LLX,LLY,URX,URY: Array[0..100] of longint;
   Class1:array[0..100] of single;
   Desc:array[0..100] of string;
```

```
  i,count,Finish: integer;
  identified:boolean;

begin
  assignfile(Data,'data\class.txt');
  reset(Data);
  readln(Data);
  readln(Data);
  readln(Data);
  i:=0;
  BLX:=strtointdef(delxedit.text,0);
  BLY:=strtointdef(delyedit.text,0);
  identified:=false;
  while not eof (Data) do
    begin
      readln(Data,LLX[i],LLY[i],URX[i],URY[i],Class1[i],Desc[i]);
      if (BLX=LLX[i]) and (BLY=LLY[i]) then
        begin
          i:=i-1;
          identified:=true;
        end;
      inc(i);
    end;
    Finish:=i-1;
    closefile(data);
    if identified=false then
    begin
      showmessage('Node Not on List!');
    end;
  if finish>-1 then
    begin
      classgrid.rowcount:=Finish+1;
    end;
  for i:=0 to Finish do
    begin
      classgrid.cells[0,i]:=inttostr(LLX[i]);
      classgrid.cells[1,i]:=inttostr(LLY[i]);
      classgrid.cells[2,i]:=inttostr(URX[i]);
      classgrid.cells[3,i]:=inttostr(URY[i]);
      classgrid.cells[4,i]:=floattostrf(Class1[i],fffixed,6,2);
      classgrid.cells[5,i]:=Desc[i];
    end;
  assignfile(Data,'data\class.txt');
  rewrite(Data);
  writeln(Data,'Node Classification Report');
  writeln(Data);
  writeln(Data,' here it is');
  if finish<0 then
    begin
      classgrid.rowcount:=1;
      classgrid.cells[0,0]:='3';
      classgrid.cells[1,0]:='4';
      classgrid.cells[2,0]:='5';
      classgrid.cells[3,0]:='6';
      classgrid.cells[4,0]:='7';
```

```
        classgrid.cells[5,0]:='dummy';
      end;
    for i:=0 to finish do
      begin
        writeln(Data,classgrid.cells[0,i]:8,
        classgrid.cells[1,i]:8,classgrid.cells[2,i]:8,
        classgrid.cells[3,i]:8,' ',classgrid.cells[4,i]:8,classgrid.cells[5,i]:8);
      end;
    closefile(data);
    end;


procedure TClassform.previousClick(Sender: TObject);
begin

    classform.close;
end;

end.
```




```
{$M 16384, 16777216}

unit PATH;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, Grids, Gauges,MMsystem, ExtCtrls;

{POSN is just a simple position, ANODE is an adjacent NODE, NODE is self explanatory,
Graph encompasses the network}

const W = 8;   {Precision on output}
     WW= 4;
     D = 1;   {Decimal precision}

type DEMPOSN  = OBJECT

              private
                Number    : Integer;
                PositionX : Integer;
                PositionY : LongInt;
                Status    : string;
             end;
```

```
type POSN  = OBJECT

              private
                Number    : Integer;
                PositionX : Integer;
                PositionY : LongInt;
             end;

type ANode = OBJECT

              private
                Number    : Integer;
                Cost      : Single;
             end;

type Node = OBJECT

             private
                Number    : Integer;
                PositionX : Integer;
                PositionY : LongInt;
                Marked    : Integer;
                Arc       : Array[1..8] of ANode;   {adjacent nodes}
                Pred      : Integer;
                TotalCost : Single;
            end;


type Graph = OBJECT

              private
                Element: Array[0..21000] of Node;
                Size   : Integer;              {size of network}
                Start  : Integer;
                Finish : Integer;
                RowCount: Integer;
                ColCount: Integer;
            end;

type
  TPATHMAIN = class(TForm)
    ReturnB1: TButton;
    Initialize: TButton;
    demandgrid: TStringGrid;
    Reference: TLabel;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    addnode: TButton;
    deletenode: TButton;
    Label10: TLabel;
    Label12: TLabel;
    Yedit: TEdit;
    Xedit: TEdit;
```

```
    Label5: TLabel;
    Nodeedit: TEdit;
    pathgauge: TGauge;
    Route: TButton;
    SPATH: TButton;
    spathgauge: TGauge;
    Resolution: TRadioGroup;
    RadioButton1: TRadioButton;
    RadioButton2: TRadioButton;
    RadioButton3: TRadioButton;
    RadioButton4: TRadioButton;
    RadioButton5: TRadioButton;
    initializedbox: TCheckBox;
    Description: TEdit;
    Label4: TLabel;
    Label6: TLabel;
    Edit1: TEdit;
    Label7: TLabel;
    Label8: TLabel;
    CheckBox1: TCheckBox;
    procedure ReturnB1Click(Sender: TObject);
    procedure InitializeClick(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure addnodeClick(Sender: TObject);
    procedure deletenodeClick(Sender: TObject);
    procedure SPATHClick(Sender: TObject);
    procedure RouteClick(Sender: TObject);

  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  PATHMAIN: TPATHMAIN;
  data:textfile;
  Test:Graph;
  StartLook,Endlook: Longint;

implementation

uses ntc;

{$R *.DFM}

procedure TPATHMAIN.ReturnB1Click(Sender: TObject);
begin
    MAIN.SHOW;
    PATHMAIN.CLOSE;
end;

procedure TPATHMAIN.InitializeClick(Sender: TObject);
```

```
{Read in all graph data from file network.txt - format is location then adjacent nodes with costs}

  var i,count:integer;
     dummy:single;

  begin
  Assignfile(Data,'data\network.txt');
  reset(Data);
  readln(Data);
  readln(Data,Test.Size,Test.RowCount,Test.ColCount);
  readln(Data);
  pathgauge.update;
  pathgauge.visible:=true;
  count:=0;
  while not eof (Data) do begin
    pathgauge.progress:=trunc(count/Test.Size*100);
    count:=count+1;
    read(Data,Test.Element[count].Number);
    read(Data,dummy);
    read(Data,Test.Element[count].PositionX);
    read(Data,Test.Element[count].PositionY);
    read(Data,dummy);
    for i:=1 to 8 do begin                    {number of adjacent arcs}
      read(Data,Test.Element[count].Arc[i].Number);
      read(Data,Test.Element[count].Arc[i].Cost);
    end;
    readln(Data);
  end;
  Test.Element[0].Number:=0;
  Test.Element[0].TotalCost:=3000000;
  pathgauge.visible:=false;
  closefile(Data);
  initializedbox.visible:=true;
  initializedbox.checked:=true;
  end;


procedure TPATHMAIN.Button1Click(Sender: TObject);

var buffer:textfile;
   node,x,y: array[1..100] of longint;
   i,count,dummy: integer;

begin
  i:=1;
  assignfile(buffer,'data\demand.txt');
  reset(buffer);
    while not eof (buffer) do
      begin
        readln(buffer,node[i],x[i],y[i]);
        inc(i);
      end;
  demandgrid.rowcount:=i-1;
  count:=i-1;
  for i:=1 to count do
```

```
      begin
        demandgrid.cells[0,i-1]:=inttostr(i);
        demandgrid.cells[1,i-1]:=inttostr(node[i]);
        demandgrid.cells[2,i-1]:=inttostr(x[i]);
        demandgrid.cells[3,i-1]:=inttostr(y[i]);
      end;
  closefile(buffer);
end;

procedure TPATHMAIN.FormCreate(Sender: TObject);

var ref,Node,X,Y: Array[0..100] of longint;
    i,Finish: integer;
    Desc:array[0..100] of string;

begin
  assignfile(Data,'data\demand.txt');
  reset(Data);
  Xedit.text:='0';
  Yedit.text:='0';
  nodeedit.text:='0';
  i:=0;
  readln(Data);
  readln(Data);
  readln(Data);
  while not eof (Data) do
    begin
      readln(Data,ref[i],Node[i],X[i],Y[i],Desc[i]);
      inc(i);
    end;
  Finish:=i;
  demandgrid.rowcount:=Finish;
  Finish:=i;
  for i:=0 to Finish do
    begin
      demandgrid.cells[0,i]:=inttostr(ref[i]);
      demandgrid.cells[1,i]:=inttostr(Node[i]);
      demandgrid.cells[2,i]:=inttostr(X[i]);
      demandgrid.cells[3,i]:=inttostr(Y[i]);
      demandgrid.cells[4,i]:=Desc[i];
    end;
  closefile(data);
    {default values for clipart}
    pathgauge.update;
    pathgauge.visible:=true;
    pathgauge.progress:=trunc(67);
    spathgauge.update;
    spathgauge.visible:=true;
    spathgauge.progress:=trunc(33);
    edit1.visible:=true;
    label7.visible:=true;
    label8.visible:=true;
    edit1.text:='12053';
  end;
```

```
procedure TPATHMAIN.addnodeClick(Sender: TObject);

var BNode,BX,BY:LongInt;
    BDesc:string;
    NumberofNodes,NodeDummy,XDummy,YDummy:LONGINT;
    ClassDummy:single;
    DescDummy:string;
    Node,X,Y : Array[0..100] of longint;
    Desc: Array[0..100] of string;
    ref,i,count,Finish: integer;
    found:boolean;

begin
   assignfile(Data,'data\demand.txt');
   reset(Data);
   i:=0;
   readln(Data);
   readln(Data);
   readln(Data);
   while not eof (Data) do
     begin
       readln(Data,ref,Node[i],X[i],Y[i],Desc[i]);
       inc(i);
     end;
   closefile(data);
   Finish:=i;
   BX:=strtointdef(Xedit.text,0);
   BY:=strtointdef(Yedit.text,0);
   BDesc:=(' ' + Description.text);
   Found:=false;
   BNode:=100000;
   if (BX=0) OR (BY=0) THEN
     begin
       BX:=0;
       BY:=0;
       showmessage('Entry Error - Integer Values Only!');
       found:=true;
     end
     else
     begin
       pathgauge.update;
       pathgauge.visible:=true;
       assignfile(Data,'data\network.txt');
       reset(Data);
       readln(Data);
       readln(Data,NumberofNodes);
       readln(Data);
       found:=false;
       for i:=1 to NumberofNodes do
         begin
           pathgauge.progress:=trunc(i/NumberofNodes *100);
           readln(Data,NodeDummy,ClassDummy,XDummy,YDummy,DescDummy);
           If (XDummy=BX) and (YDummy=BY) then
             begin
               BNode:=NodeDummy;
```

```
                Found:=true;
                Finish:=Finish+1;
              end;
          end;
        closefile(Data);
      pathgauge.visible:=false;
      end;
  if found=false then
    begin
      showmessage('Node Unavailable!');
    end;
  demandgrid.rowcount:=Finish;
  Finish:=Finish-1;
  count:=0;
  for i:=0 to Finish do
    begin
      if BNode<Node[count] then
      begin
        demandgrid.cells[0,i]:=inttostr(i+1);
        demandgrid.cells[1,i]:=inttostr(BNode);
        demandgrid.cells[2,i]:=inttostr(BX);
        demandgrid.cells[3,i]:=inttostr(BY);
        demandgrid.cells[4,i]:=BDesc;
        BNode:=100000;
      end
      else
      begin
        demandgrid.cells[0,i]:=inttostr(i+1);
        demandgrid.cells[1,i]:=inttostr(Node[count]);
        demandgrid.cells[2,i]:=inttostr(X[count]);
        demandgrid.cells[3,i]:=inttostr(Y[count]);
        demandgrid.cells[4,i]:=Desc[count];
        inc(count);
      end;
    end;
  assignfile(Data,'data\demand.txt');
  rewrite(Data);
  writeln(Data,'Demand Node Report ');
  writeln(Data);
  writeln(Data,'Ref Number  Node     X       Y    Name');
  for i:=0 to finish do
    begin
      writeln(Data,(i+1):8,demandgrid.cells[1,i]:8,demandgrid.cells[2,i]:8,
      demandgrid.cells[3,i]:8,demandgrid.cells[4,i]);
    end;
  closefile(data);
end;

procedure TPATHMAIN.deletenodeClick(Sender: TObject);

var BNode:LONGINT;
   Node,X,Y: Array[0..100] of longint;
   Desc: Array[0..100] of string;
  ref,i,count,Finish: integer;
  identified:boolean;
```

```
begin
  assignfile(Data,'data\demand.txt');
  reset(Data);
  i:=0;
  BNode:=strtointdef(Nodeedit.text,0);
  readln(Data);
  readln(Data);
  readln(Data);
  identified:=false;
  while not eof (Data) do
    begin
      readln(Data,Ref,Node[i],X[i],Y[i],Desc[i]);
      if (BNode=Node[i]) then
        begin
          i:=i-1;
          identified:=true;
        end;
      inc(i);
    end;
  if identified=false then
    begin
      showmessage('Node Not on List!');
    end;
  closefile(data);
  Finish:=i-1;
  demandgrid.rowcount:=Finish+1;
  for i:=0 to Finish do
    begin
      demandgrid.cells[0,i]:=inttostr(i+1);
      demandgrid.cells[1,i]:=inttostr(Node[i]);
      demandgrid.cells[2,i]:=inttostr(X[i]);
      demandgrid.cells[3,i]:=inttostr(Y[i]);
      demandgrid.cells[4,i]:=Desc[i];
    end;
  assignfile(Data,'data\demand.txt');
  rewrite(Data);
  writeln(Data,'Demand Node Report ');
  writeln(Data);
  writeln(Data,'Ref Number  Node     X       Y    Name');
  if finish<0 then
    begin
      demandgrid.rowcount:=1;
      demandgrid.cells[0,0]:=inttostr(0);
      demandgrid.cells[1,0]:=inttostr(0);
      demandgrid.cells[2,0]:=inttostr(0);
      demandgrid.cells[3,0]:=inttostr(0);
      demandgrid.cells[4,0]:='0';
      writeln(data,'0 0 0 0 0');
    end;
  for i:=0 to finish do
    begin
      writeln(Data,(i+1):8,demandgrid.cells[1,i]:8,
      demandgrid.cells[2,i]:8,demandgrid.cells[3,i]:8,demandgrid.cells[4,i]);
    end;
```

```
  closefile(data);
  end;


procedure TPATHMAIN.SPATHClick(Sender: TObject);
{Modified label correcting algorithm applied to the Graph.  Input is from source.txt
which is an emitter etc. Note use of variable T as a delimiter.}

  var Source:POSN;
     NOTDONE:boolean;
     i,j,ii,T,count,factor:integer;
     gauge:single;
     marked:longint;
     StartAdj,EndAdj,resolution:integer;
     Adj:array[1..8] of integer;

  begin

  Assignfile(Data,'data\Source.txt');    {file of source locations}
  reset(Data);
    read(Data,Source.Number);
    read(Data,Source.PositionX);
    read(Data,Source.PositionY);
    readln(Data);
  closefile(Data);
  spathgauge.update;
  spathgauge.visible:=true;
  resolution:=15000;
  gauge:=1;
  {get resolution of path algorithm}
  if radiobutton1.checked=true then
    begin
      resolution:=5000;
      gauge:=5.0;
    end;
  if radiobutton2.checked=true then
    begin
      resolution:=10000;
      gauge:=1.52;
    end;
  if radiobutton3.checked=true then
    begin
      resolution:=25000;
      gauge:=1.07;
    end;
  if radiobutton4.checked=true then
    begin
      resolution:=15000;
      gauge:=1.07;
    end;
  if radiobutton5.checked=true then
    begin
      resolution:=20000;
      gauge:=0.68;
    end;
```

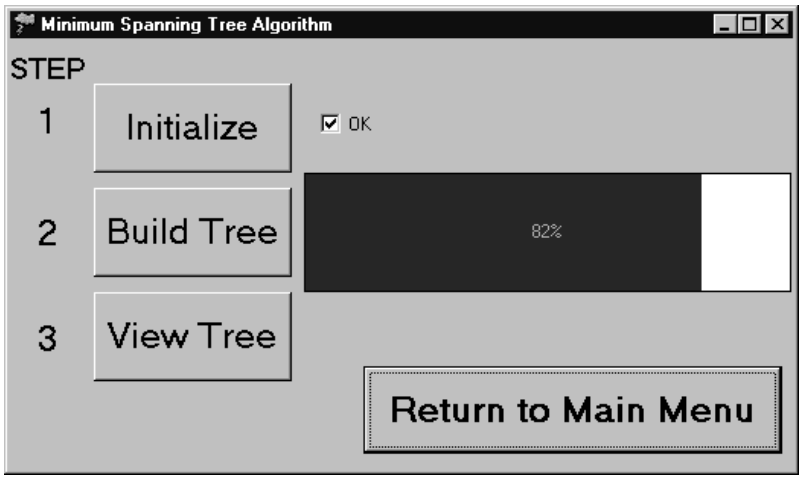
```
  {Modified Label Correcting }
  for i:=1 to Test.Size do
    begin
      if (Source.Number=Test.Element[i].Number) then
        begin
          Test.Element[i].Pred:=0;
          Test.Element[i].TotalCost:=0;
        end
      ELSE
        begin
          Test.Element[i].TotalCost:=3000000;     {hardwired - adjust for larger networks}
        end;
    end;
  NOTDONE:=TRUE;
  T:=1;
  count:=1;
  marked:=1;
  factor:=(resolution div 10)*3;
  if (Source.Number>(Test.Start+factor)) then
    begin
      StartLook:=Source.Number-factor;
    end
    else
    begin
      StartLook:=Test.Start;
    end;
  if (Source.Number<(Test.Finish-factor)) then
    begin
      EndLook:=Source.Number+factor;
    end
    else
    begin
      EndLook:=Test.Finish;        {look no farther than 10k south/north}
    end;
  while (NOTDONE=TRUE) do
    begin
      NOTDONE:=FALSE;
      for i:=StartLook to EndLook do
      begin   {boundary 1 to Test.Size for all,StartLook to EndLook Test.Start to Test.Finish for delimited
range}
        if (Test.Element[i].TotalCost<resolution) then  {resolution is a delimiter}
          begin
            {identify adjacent nodes to check}
            for j:= 1 to 8 do
              begin
                Adj[j]:=Test.Element[i].Arc[j].Number;
              end;
            for j:= 1 to 8 do
              begin
                if
(Test.Element[Adj[j]].TotalCost>(Test.Element[i].TotalCost+Test.Element[i].Arc[j].Cost)) then
                  begin
                    marked:=marked+1;
                    spathgauge.progress:=trunc((gauge*marked)/(EndLook-StartLook) *100);
```

```
Test.Element[Adj[j]].TotalCost:=Test.Element[i].TotalCost+Test.Element[i].Arc[j].Cost;
                        Test.Element[Adj[j]].Pred:=i;
                        NOTDONE:=TRUE;
                      end;
                  end;
              end;
          end;
      end;
  spathgauge.visible:=false;
  end;

procedure TPATHMAIN.RouteClick(Sender: TObject);
  {This procedure takes each demand sequentially - then determines optimal arcs to all
  other demands}

  var Demand:ARRAY[1..300] of DEMPOSN;              {hardwired - set to #demands}
     Buffer:Array[1..300] of integer;
     existingnodes,nullarc,produced,i,ii,count,counter,back,buffercount,maxbuffer,tail:integer;
     dummy:single;
     dummy1:string;
     DataE,temp,Outfile: textfile;       {Output File}

  begin
  if initializedbox.checked=false then
  begin
    showmessage('Initialization is Required!');
  end
  else
  begin
  if (checkbox1.checked=true) and (radiobutton1.checked=true) then
    begin
      showmessage('Minimum Resolution of 10km Required!');
      radiobutton2.checked:=true;
    end;
  pathgauge.update;
  pathgauge.visible:=true;
  pathgauge.progress:=0;
  edit1.visible:=true;
  label7.visible:=true;
  label8.visible:=true;
  ii:=0;
  existingnodes:=0;
  Test.Start:=Test.Size;
  Test.Finish:=0;
  if checkbox1.checked=true then      {existing Cable nodes}
    begin
    assignfile(DataE,'Data\DemandE.txt');
    reset(DataE);
    readln(DataE);
    readln(DataE);
    readln(DataE);
    while not eof (DataE) do
      begin
      inc(ii);
```

```
      read(DataE,dummy);
      read(DataE,Demand[ii].Number);
      read(DataE,Demand[ii].PositionX);
      read(DataE,Demand[ii].PositionY);
      Demand[ii].Status:='E';
      readln(DataE,dummy1);
      if Demand[ii].Number<Test.Start then
        begin
          Test.Start:=Demand[ii].Number;    {set boundarys of algorithm}
        end;
      if Demand[ii].Number>Test.Finish then
        begin
          Test.Finish:=Demand[ii].Number;
        end;
    end;
    existingnodes:=ii;
    closefile(DataE);
end;
Assignfile(Data,'Data\Demand.txt');    {file of demand locations}
reset(Data);
readln(Data);
readln(Data);
readln(Data);
while not eof (Data) do
  begin
    inc(ii);
    read(Data,dummy);
    read(Data,Demand[ii].Number);
    read(Data,Demand[ii].PositionX);
    read(Data,Demand[ii].PositionY);
    Demand[ii].Status:='N';
    readln(Data,dummy1);
    if Demand[ii].Number<Test.Start then
      begin
        Test.Start:=Demand[ii].Number;    {set boundarys of algorithm}
      end;
    if Demand[ii].Number>Test.Finish then
      begin
        Test.Finish:=Demand[ii].Number;
      end;
  end;
count:=ii;                  {number of demand locations}
closefile(Data);
assignfile(Outfile,'data\arcs.txt');
rewrite(Outfile);
writeln(Outfile,count,' nodes in network ',existingnodes,' linked nodes');
for counter:=(existingnodes+1) to count do
  begin {produce infinity arcs}
    writeln(Outfile,Demand[counter].Number:5,' ','     0',
   ' ',Demand[counter].PositionX:6,' ',Demand[counter].PositionY:6);
    writeln(Outfile,'3000000.00  total cost ');
    writeln(Outfile);
  end;
if checkbox1.checked=true then
begin
```

```
    for counter:=1 to existingnodes do
    begin {produce null arcs}
      writeln(Outfile,Demand[counter].Number:5,' ',Demand[counter].Number:5,
     ' ',Demand[counter].PositionX:6,' ',Demand[counter].PositionY:6);
      writeln(Outfile,Demand[counter].Number:5,' ','     0',
     ' ',Demand[counter].PositionX:6,' ',Demand[counter].PositionY:6);
      writeln(Outfile,'0.00  total cost ');
      writeln(Outfile);
    end;
  end;
  for counter:=1 to count do
    begin
      assignfile(temp,'data\Source.txt');
      rewrite(temp);
      edit1.text:=inttostr(Demand[counter].number);
      writeln(temp,Demand[counter].Number:6,Demand[counter].PositionX:6,
        Demand[counter].PositionY:6);
      closefile(temp);
      Spath.click;
      pathgauge.progress:=trunc(counter/count *100);
      for i:= Test.Start to Test.Finish do {boundary conditions}
        begin     {StartLook to EndLookchange counter to 1  to verify reverse arc length - slower -
Test.Start is faster but need reverse arc printing}
                  {also change output function at end - buffer items are redundant}
            for ii:=1 to count do
              begin
                if (Demand[ii].Number=Test.Element[i].Number) then
                  begin
                  {modify existing cable arcs}
                    if (Demand[ii].Status='E') and (Demand[counter].Status='E') then
                      begin
                        Test.Element[i].TotalCost:=0;
                      end;
                    if (Test.Element[i].TotalCost>0) and (Test.Element[i].TotalCost<>3000000)then
                      begin
                        buffercount:=1;
                        back:=i;
                        repeat
                          buffer[buffercount]:=back;
                          writeln(Outfile,Test.Element[back].Number:5,' ',Test.Element[back].Pred:5,
                          ' ',Test.Element[back].PositionX:6,' ',Test.Element[back].PositionY:6);
                          tail:=back;
                          back:=Test.Element[back].Pred; {regress your way through elements to source}
                          inc(buffercount);
                        until back=0;
                          maxbuffer:=buffercount-1;
                          writeln(Outfile,Test.Element[i].TotalCost:W:D,'  total cost ');
                          writeln(Outfile);
                      end;
                  end;
              end;
          end;
    end;
  edit1.visible:=false;
  pathgauge.visible:=false;
```

```
  label7.visible:=false;
  label8.visible:=false;
  closefile(Outfile);
  end;
  end;

end.
```




```
{$M 16384, 16777216}

unit TREE;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, Gauges;

  {POSN is just a simple position, ANODE is an adjacent NODE, NODE is self explanatory,
Graph encompasses the network}

const W = 4;   {Precision on output}
      D = 1;   {Decimal precision}


type POSN  = OBJECT

               private
                 Number    : Integer;
                 PositionX : LongInt;
                 PositionY : Integer;
             end;

type ANode = OBJECT

               private
                 Number    : Integer;
                 PositionX : LongInt;
                 PositionY : Integer;
                 Pred      : Integer;
             end;

type Node = OBJECT

              private
                Number    : Integer;
                PositionX : Integer;
```

```
            PositionY : LongInt;
            Arc       : Array[1..100] of ANode;   {adjacent nodes in path - hardwired for 30km}
            Pred      : Integer;
            length    : Integer; {number of arcs in path}
            marked    : boolean;
            USED      : boolean;
            Status    : string[1]; {Existing or New}
            TotalCost : Single;
          end;


type Graph = OBJECT

            private          {Limits RAM use - 3800 Elements ~8 meg}
              Element: Array[0..3800] of Node;  {number of nodes in total network}
              Paths  : Integer;              {Number of Paths in network n(n-1)}
              Size   : Integer;              {Number of demand nodes n in network}
          end;

type
  TMSTMAIN = class(TForm)
    ReturnB2: TButton;
    buildtree: TButton;
    TreeEdit1: TEdit;
    ViewTree: TButton;
    Initialize: TButton;
    CheckBox1: TCheckBox;
    Gauge1: TGauge;
    CheckBox2: TCheckBox;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    procedure ReturnB2Click(Sender: TObject);
    procedure buildtreeClick(Sender: TObject);
    procedure ViewTreeClick(Sender: TObject);
    procedure InitializeClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  MSTMAIN: TMSTMAIN;
  Data: textfile;
  Test:Graph;

implementation

uses ntc, graphUnit;

{$R *.DFM}
```

```
procedure TMSTMAIN.ReturnB2Click(Sender: TObject);
begin
  treeedit1.text:=' ';
  treeedit1.visible:=false;
  checkbox1.visible:=false;
  Main.show;
  MSTMAIN.close;
end;

procedure TMSTMAIN.buildtreeClick(Sender: TObject);

{algorithm for min tree - arcs considered are source to all demands,
demands to all other demands}

  var Source:POSN;
     NOTDONE:boolean;
     Total,min: single;
     dummy,i,j,ii,iii,T,count,subtrees,markedArc,markedNode,Nodes:Integer;
     minmarker,tail,start,bogey,ArcAdd:integer;
     buffer,report:textfile;
     Subtree,Subtreeadd:integer;

  begin
    if checkbox1.checked=false then
    begin
      showmessage('Initialization Required!');
    end
    else
    begin
    checkbox1.visible:=false;
    gauge1.update;
    gauge1.visible:=true;
    treeedit1.text:=' ';
    {mark the node with smallest cost arc }
    MarkedArc:=1;
    MarkedNode:=Test.Size;
    Nodes:=Test.Size;
    minmarker:=0;
    Subtree:=0;
    total:=0;
    start:=0;
    ArcAdd:=1;
    T:=1;                          {number of paths used}
    min:=3000000;                                {hardwired}
    for i:=1 to Test.Paths do
    begin
      if (Test.Element[i].TotalCost<min) then
        begin
          min:=Test.Element[i].TotalCost;
          start:=i;
        end;
    end;
    if min>2000000 then start:=1;
    Test.Element[start].Marked:=TRUE;       {mark head }
    Test.Element[start].Used:=TRUE;         {path is used}
```

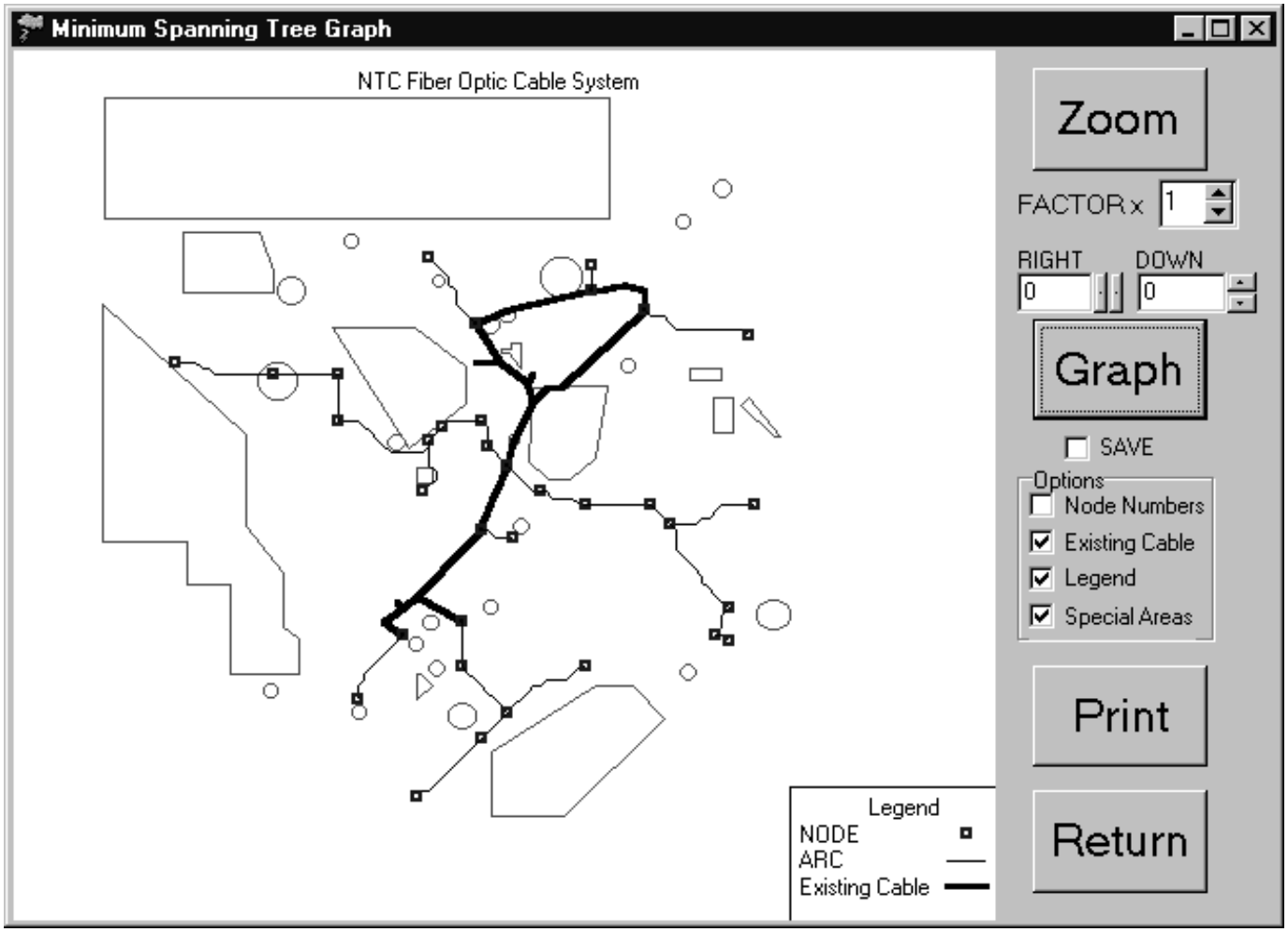
```
tail:=Test.Element[Start].Arc[Test.Element[Start].length].Number;
for i:=1 to Test.Paths do              {mark other heads - destroy reverse arc}
  begin
    dummy:=Test.Element[i].Arc[Test.Element[i].length].Number;
    if (Test.Element[i].Number=tail) AND          {H=T}  {get ride of reverse arc - mark tail}
      (dummy=Test.Element[start].Number) or        {T=H}
      (Test.Element[i].Number=Test.Element[start].Number) or
      (Test.Element[i].Number=tail)  then   {other heads}
      begin
        Test.Element[i].Marked:=TRUE;
      end;
  end;
while T<(Test.Size-1) do              {need n-1 paths for connected graph}
begin
 min:=3000000;
{find min arc emanating from marked node tails}
 for i:=1 to Test.Paths do
    begin
      If (Test.Element[i].Marked=TRUE) then   {new comment}
        begin
          for ii:=1 to Test.Paths do
            begin                       {check heads with unmarked tails}
              if (Test.Element[i].Number=Test.Element[ii].Number) then  {new comment}
                  begin                 {get shortest unused path with unmarked tail}
                  tail:=Test.Element[ii].Arc[Test.Element[ii].length].Number;
                    if (Test.Element[ii].TotalCost<min) AND
                      (Test.Element[ii].Used=FALSE) then
                      begin
                        for iii:=1 to Test.Paths do
                          begin
                            if (Test.Element[iii].Number=tail) AND
                              (Test.Element[iii].Marked=False) then
                              begin
                                min:=Test.Element[ii].TotalCost;
                                minmarker:=ii;
                              end;
                          end;
                      end;
                  end;
            end;
        end;
    end;
  if (min=3000000) then      {no available arc from marked nodes}
    begin
      min:=4000000;                         {need to find cheapest unmarked arc}
      subtreeadd:=1;
      for iii:=1 to Test.Paths do
        begin
          if (Test.Element[iii].Marked=False) and (Test.Element[iii].TotalCost<min) then
            begin
              min:=Test.Element[iii].TotalCost;
              Subtree:=Subtree+subtreeadd;
              minmarker:=iii;
              subtreeadd:=0;
            end;
```

```
            end;
          end;
        Test.Element[minmarker].Marked:=TRUE;   {mark the head of the new arc}
        Test.Element[minmarker].Used:=TRUE;     {show path as used}
        tail:=Test.Element[minmarker].Arc[Test.Element[minmarker].length].Number;
        for i:=1 to Test.Paths do               {mark other heads - destroy reverse arc}
          begin
            dummy:=Test.Element[i].Arc[Test.Element[i].length].Number;
            if (Test.Element[i].Number=tail) AND          {H=T}  {get ride of reverse arc - mark tail}
              (dummy=Test.Element[minmarker].Number) or        {T=H}
              (Test.Element[i].Number=Test.Element[minmarker].Number) or
              (Test.Element[i].Number=tail)  then   {other heads}
              begin
                Test.Element[i].Marked:=TRUE;
              end;
          end;
        inc(T);
        gauge1.progress:=trunc((T/Test.Size)*100);
      end;
      assignfile(buffer,'data\buffer.txt');
      rewrite(buffer);
      writeln(buffer,(Test.Size-1));
      for count:=1 to Test.Paths do
        begin
          if (Test.Element[count].used=TRUE) and (Test.Element[count].TotalCost<>0) then
            begin
              write(buffer,Test.Element[count].Number:8,' ');
              write(buffer,Test.Element[count].PRED:8,' ');
              write(buffer,Test.Element[count].PositionX:8,' ');
              writeln(buffer,Test.Element[count].PositionY:8,' ');
              for i:=1 to Test.Element[count].length do
                begin           {Arcs}
                  write(buffer,Test.Element[count].Arc[i].Number:8,' ');
                  write(buffer,Test.Element[count].Arc[i].Pred:8,' ');
                  write(buffer,Test.Element[count].Arc[i].PositionX:8,' ');
                  writeln(buffer,Test.Element[count].Arc[i].PositionY:8,' ');
                end;
            end;
        end;
      writeln(buffer,'0 0 0 0');
      closefile(buffer);
      assignfile(report,'data\Tree.txt');               {create report}
      rewrite(report);
      writeln(report,'Network Report');
      writeln(report);
      writeln(report,'Connected Nodes');
      writeln(report);
      for count:=1 to Test.Paths do
        begin
          if (Test.Element[count].used=TRUE) and (Test.Element[count].TotalCost<>0) and
(Test.Element[count].totalcost<>3000000) then
            begin
              write(report,Test.Element[count].Number:8,' ');
              write(report,Test.Element[count].PRED:8,' ');
              write(report,Test.Element[count].PositionX:8,' ');
```

```
          writeln(report,Test.Element[count].PositionY:8,' ');
          for i:=1 to Test.Element[count].length do
            begin            {Arcs}
              write(report,Test.Element[count].Arc[i].Number:8,' ');
              write(report,Test.Element[count].Arc[i].Pred:8,' ');
              write(report,Test.Element[count].Arc[i].PositionX:8,' ');
              writeln(report,Test.Element[count].Arc[i].PositionY:8,' ');
            end;
          writeln(report,'Arc Cost ',Test.Element[count].TotalCost:8:2);
          Total:=Total+Test.Element[count].TotalCost;
          writeln(report);
      end;
  end;
writeln(report);
writeln(report,'Unconnected Nodes');
writeln(report);
for count:=1 to Test.Paths do
  begin
    if (Test.Element[count].used=TRUE) and (Test.Element[count].totalcost=3000000)then
      begin
        write(report,Test.Element[count].Number:8,' ');
        write(report,Test.Element[count].PositionX:8,' ');
        writeln(report,Test.Element[count].PositionY:8,' ');
        writeln(report,'Arc Cost ',Test.Element[count].TotalCost:8:2);
        markednode:=markednode-1;
        writeln(report);
      end;
  end;
if (markednode<Test.Size) then
 begin
   showmessage('Warning Disjoint Graph!!');
 end;
  {boundary conditions}
if subtree<>0 then Subtree:=subtree+1-(Test.Size-markedNode);
if markednode=1 then markednode:=markednode-1;
if markednode=0 then subtree:=0;
if markednode=0 then Total:=0;
if markednode=Test.Size then subtree:=0;
writeln(report);
writeln(report,'SUMMARY');
writeln(report);
writeln(report,'  ',Nodes,' Nodes');
writeln(report);
writeln(report,'  ',markedNode,' Connected Nodes');
writeln(report);
writeln(report,'  ',(Test.Size-markedNode),' Disconnected Nodes');
writeln(report);
if subtree<>1 then writeln(report,'   ',(subtree),' Disconnected Trees');
writeln(report);
writeln(report,'Total Network Cost ',Total:10:2);
closefile(report);
treeedit1.text:='       Tree Complete';
gauge1.visible:=false;
checkbox2.visible:=true;
checkbox2.checked:=true;
```

```
    end;
    end;

procedure TMSTMAIN.ViewTreeClick(Sender: TObject);
begin
  TreeGraph.show;
  TreeEdit1.text:=' ';
  TreeEdit1.visible:=false;
  checkbox1.checked:=false;
  checkbox1.visible:=false;
end;

procedure TMSTMAIN.InitializeClick(Sender: TObject);
{Read in all graph data from file arcs.txt - arc x,y combinations
destination to source (head to tail), Pred , then total cost -initializes boolean values}

  var i,count,Bumper,dummy:Integer;

  begin
  checkbox2.visible:=false;
  treeedit1.visible:=true;
  Assignfile(Data,'data\arcs.txt');
  reset(Data);
  readln(Data,Test.Size);
  count:=0;
  while not eof (Data) do begin
    i:=1;
    count:=count+1;
    Test.Element[count].Marked:=FALSE;
    Test.Element[count].Used:=FALSE;
    read(Data,Test.Element[count].Number);
    read(Data,Test.Element[count].PRED);
    read(Data,Test.Element[count].PositionX);
    read(Data,Test.Element[count].PositionY);
    Bumper:=Test.Element[count].PRED;          {Bumper ensures a node precedes}
    while Bumper <> 0 do begin
      read(Data,Test.Element[count].Arc[i].Number);
      read(Data,Test.Element[count].Arc[i].PRED);
      read(Data,Test.Element[count].Arc[i].PositionX);
      read(Data,Test.Element[count].Arc[i].PositionY);
      Bumper:=Test.Element[count].Arc[i].PRED;
      inc(i);
    end;
    if Bumper = 0 then
      begin
        Test.Element[count].Arc[i].Number:=Test.Element[count].Number;
        Test.Element[count].Arc[i].PRED:=Test.Element[count].PRED;
        Test.Element[count].Arc[i].PositionX:=Test.Element[count].PositionX;
        Test.Element[count].Arc[i].PositionY:=Test.Element[count].PositionY;
      end;
    Test.Element[count].length:=i-1;
    readln(Data,Test.Element[count].TotalCost);
    if Test.Element[count].TotalCost=0 then
      begin
        Test.Element[count].marked:=true;
```

```
          Test.Element[Test.Element[count].length].marked:=true;
        end;
      readln(Data);
    end;
    Test.Paths:=count;
    closefile(Data);
    treeedit1.text:='  Initialization Complete';
    checkbox1.visible:=true;
    checkbox1.checked:=true;
end;

procedure TMSTMAIN.FormCreate(Sender: TObject);
begin
    treeedit1.visible:=false;
end;

end.
```




```
unit graphUnit;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, ExtCtrls, Printers, MMsystem, ComCtrls, Spin;

type
  Ttreegraph = class(TForm)
    dgraph: TButton;
    Return: TButton;
    XZoomEdit: TEdit;
    YZoomEdit: TEdit;
    Zoom: TButton;
    Print: TButton;
    NodeN: TCheckBox;
    existc: TCheckBox;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    GroupBox1: TGroupBox;
```

```
    legendBox: TCheckBox;
    zoomEdit: TSpinEdit;
    RL: TUpDown;
    UpDown: TUpDown;
    offlimit: TCheckBox;
    savebox: TCheckBox;
    procedure dgraphClick(Sender: TObject);
    procedure ReturnClick(Sender: TObject);
    procedure ZoomClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure PrintClick(Sender: TObject);
    procedure UpDownClick(Sender: TObject; Button: TUDBtnType);
    procedure RLClick(Sender: TObject; Button: TUDBtnType);
    procedure zoomEditChange(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  treegraph: Ttreegraph;
  XZoom,YZoom,FZoom:longint;

implementation

uses print;

{$R *.DFM}

procedure Ttreegraph.dgraphClick(Sender: TObject);

var PX,PY,PRX,PRY,radius,startX,StartY,FinishX,FinishY: real;
    RadiusX,RadiusY,PGX,PGY:longint;
    i,ii,count,head,counter,NumberofArcs,dummy,dummy1: integer;
    buffer:textfile;
    cost:real;
    BoxSize,TextSize,previous:integer;
    drawtype:string[1];
    dummy2:string;
    mybitmap:Tbitmap;

begin
   mybitmap:=tbitmap.create;
   mybitmap.canvas.pen.width:=0;
   mybitmap.height:=450;
   mybitmap.width:=500;
   BoxSize:=2;
   count:=5;
      {Draw offlimit/restricted areas}
   if offlimit.checked=true then
   begin
   assignfile(buffer,'data\offlimit.txt');
   reset(buffer);
   mybitmap.canvas.pen.width:=1;
```

```
  mybitmap.canvas.pen.color:=clRed;
  while not eof (buffer) do
    begin
      readln(buffer,drawtype,startX,StartY,FinishX,FinishY,dummy2);
      PX:=((StartX-(5000+XZoom))/755)*mybitmap.width*FZoom;       {scaling factors}
      PY:=((((39460-YZoom)-StartY)/675)*mybitmap.height)*FZoom;{distortion is possible}
      PRX:=((FinishX-(5000+XZoom))/755)*mybitmap.width*FZoom;
      PRY:=((((39460-YZoom)-FinishY)/675)*mybitmap.height)*FZoom;
      PGX:=trunc(PX);
      PGY:=trunc(PY);
      radiusX:=trunc(PRX);
      radiusY:=trunc(PRY);
      if drawtype='M' then mybitmap.Canvas.moveto(PGX,PGY);
      if drawtype='L' then mybitmap.canvas.lineto(PGX,PGY);
      if drawtype='C' then mybitmap.canvas.ellipse(PGX,PGY,radiusX,radiusY);
    end;
  closefile(buffer);
  end;
 {draw existing cable}
  assignfile(buffer,'data\exist.txt');
  reset(buffer);
  mybitmap.canvas.pen.color:=rgb(count,count,count);
  mybitmap.canvas.pen.style:=psSolid; {dash,dot,dashdot etc p301}
  mybitmap.canvas.pen.width:=3;
  if existc.checked=true then
  begin
  while not eof (buffer) do
    begin
      readln(buffer,drawtype,FinishX,FinishY,dummy2);
      PX:=((FinishX-(5000+XZoom))/755)*mybitmap.width*FZoom;       {scaling factors}
      PY:=((((39460-YZoom)-FinishY)/675)*mybitmap.height) *FZoom; {distortion is possible}
      PGX:=trunc(PX);
      PGY:=trunc(PY);
      PGX:=trunc(PX);
      PGY:=trunc(PY);
      if drawtype='L' then mybitmap.canvas.lineto(PGX,PGY) else mybitmap.canvas.moveto(PGX,PGY);
    end;
  end;
  closefile(buffer);
  {Draw Legend/Text Data}
  if Legendbox.checked=true then
  begin
    mybitmap.canvas.pen.width:=1;
    mybitmap.canvas.moveto((mybitmap.width div 2 +
trunc(0.29*mybitmap.width)),trunc(0.84*mybitmap.height));
    mybitmap.canvas.lineto((mybitmap.width div 2
+trunc(0.29*mybitmap.width)),trunc(mybitmap.height));
    mybitmap.canvas.moveto((mybitmap.width div 2
+trunc(0.29*mybitmap.width)),trunc(0.84*mybitmap.height));
    mybitmap.canvas.lineto((mybitmap.width),trunc(0.84*mybitmap.height));
    mybitmap.canvas.textout((mybitmap.width div 2 -
trunc(0.15*mybitmap.width)),trunc(0.02*mybitmap.height),'NTC Fiber Optic Cable System');
    mybitmap.canvas.textout((mybitmap.width div 2
+trunc(0.37*mybitmap.width)),trunc(0.85*mybitmap.height),'Legend');
```

```
    mybitmap.canvas.textout((mybitmap.width div 2
+trunc(0.3*mybitmap.width)),trunc(0.88*mybitmap.height),'NODE');
    mybitmap.canvas.pen.color:=clRed;
    mybitmap.canvas.pen.width:=2;
    mybitmap.canvas.polyline([Point(((mybitmap.width div 2) +trunc(0.47*mybitmap.width)-
boxsize),(trunc(0.895*mybitmap.height)+BoxSize)),    {1st head}
      Point((mybitmap.width div 2
+trunc(0.47*mybitmap.width)+BoxSize),(trunc(0.895*mybitmap.height)+BoxSize)),
      Point((mybitmap.width div 2
+trunc(0.47*mybitmap.width)+BoxSize),(trunc(0.895*mybitmap.height)-BoxSize)),
      Point((mybitmap.width div 2 +trunc(0.47*mybitmap.width)-
BoxSize),(trunc(0.895*mybitmap.height)-BoxSize)),
      Point((mybitmap.width div 2 +trunc(0.47*mybitmap.width)-
BoxSize),(trunc(0.895*mybitmap.height)+BoxSize))]);
    mybitmap.canvas.pen.color:=rgb(5,5,5);
    mybitmap.canvas.pen.width:=1;
    mybitmap.canvas.textout((mybitmap.width div 2
+trunc(0.3*mybitmap.width)),trunc(0.91*mybitmap.height),'ARC');
    mybitmap.canvas.moveto((mybitmap.width div 2
+trunc(0.45*mybitmap.width)),trunc(0.925*mybitmap.height));
    mybitmap.canvas.lineto((mybitmap.width div 2
+trunc(0.49*mybitmap.width)),trunc(0.925*mybitmap.height));
    mybitmap.canvas.textout((mybitmap.width div 2
+trunc(0.3*mybitmap.width)),trunc(0.94*mybitmap.height),'Existing Cable');
    mybitmap.canvas.pen.width:=3;
    mybitmap.canvas.moveto((mybitmap.width div 2
+trunc(0.45*mybitmap.width)),trunc(0.955*mybitmap.height));
    mybitmap.canvas.lineto((mybitmap.width div 2
+trunc(0.49*mybitmap.width)),trunc(0.955*mybitmap.height));
    mybitmap.canvas.pen.width:=1;
  end;
  assignfile(buffer,'data\buffer.txt');
  reset(buffer);
  readln(buffer,NumberofArcs);
  NumberofArcs:=NumberofArcs;
  counter:=0;
  readln(buffer,dummy,dummy1,PX,PY);
  PX:=((PX-(5000+XZoom))/755)*mybitmap.width*FZoom;       {scaling factors}
  PY:=((((39460-YZoom)-PY)/675)*mybitmap.height)*FZoom; {distortion is possible}
  PGX:=trunc(PX);
  PGY:=trunc(PY);
  TextSize:=12;
  mybitmap.canvas.pen.style:=psSolid; {dash,dot,dashdot etc p301}
  mybitmap.canvas.pen.color:=clBlue;
  mybitmap.canvas.pen.width:=2;
  mybitmap.canvas.polyline([Point((PGX-BoxSize),(PGY+BoxSize)),    {1st head}
    Point((PGX+BoxSize),(PGY+BoxSize)),Point((PGX+BoxSize),(PGY-BoxSize)),
    Point((PGX-BoxSize),(PGY-BoxSize)),Point((PGX-BoxSize),(PGY+BoxSize))]);
  if nodeN.checked=true then mybitmap.canvas.textout((PGX),(PGY-TextSize),inttostr(dummy));
  count:=5;
  i:=1;
  mybitmap.canvas.pen.width:=i;
  mybitmap.Canvas.moveto(PGX,PGY);
  head:=0;
  previous:=dummy1;
```

```
  while counter<>NumberofArcs do
    begin
      readln(buffer,dummy,dummy1,PX,PY);
      PX:=((PX-(5000+XZoom))/755)*mybitmap.width*FZoom;
      PY:=((((39460-YZoom)-PY)/675)*mybitmap.height)*FZoom;
      PGX:=trunc(PX);
      PGY:=trunc(PY);
      if previous=0 then
        begin
          mybitmap.canvas.pen.color:=clBlue;
          mybitmap.canvas.pen.width:=2;
          mybitmap.canvas.polyline([Point((PGX-BoxSize),(PGY+BoxSize)),
            Point((PGX+BoxSize),(PGY+BoxSize)),Point((PGX+BoxSize),(PGY-BoxSize)),
            Point((PGX-BoxSize),(PGY-BoxSize)),Point((PGX-BoxSize),(PGY+BoxSize))]);
        if nodeN.checked=true then mybitmap.canvas.textout((PGX),(PGY-TextSize),inttostr(dummy));
        end;
      if (dummy1=0) then
        begin                          {tail}
          if (dummy<>0) and (previous<>0) then
            begin
              mybitmap.canvas.pen.color:=rgb(count,count,count);
              mybitmap.canvas.pen.width:=1;
              mybitmap.Canvas.lineto(PGX,PGY);     {need for null arcs}
            end
            else
            begin
              mybitmap.Canvas.moveto(PGX,PGY);     {need for null arcs}
            end;
          mybitmap.canvas.pen.color:=clBlue;
          mybitmap.canvas.pen.width:=2;
          mybitmap.canvas.polyline([Point((PGX-BoxSize),(PGY+BoxSize)),
            Point((PGX+BoxSize),(PGY+BoxSize)),Point((PGX+BoxSize),(PGY-BoxSize)),
            Point((PGX-BoxSize),(PGY-BoxSize)),Point((PGX-BoxSize),(PGY+BoxSize))]);
          if nodeN.checked=true then mybitmap.canvas.textout((PGX),(PGY-TextSize),inttostr(dummy));
          if dummy<>0 then readln(buffer,dummy,dummy1,PX,PY);
          PX:=((PX-(5000+XZoom))/755)*mybitmap.width*FZoom;
          PY:=((((39460-YZoom)-PY)/675)*mybitmap.height)*FZoom;
          PGX:=trunc(PX);
          PGY:=trunc(PY);
          previous:=dummy1;
          mybitmap.Canvas.moveto(PGX,PGY);
          mybitmap.canvas.polyline([Point((PGX-BoxSize),(PGY+BoxSize)),
            Point((PGX+BoxSize),(PGY+BoxSize)),Point((PGX+BoxSize),(PGY-BoxSize)),
            Point((PGX-BoxSize),(PGY-BoxSize)),Point((PGX-BoxSize),(PGY+BoxSize))]);
          if (nodeN.checked=true) then mybitmap.canvas.textout((PGX),(PGY-
TextSize),inttostr(dummy));
          mybitmap.Canvas.moveto(PGX,PGY);
          inc(i);
         {  count:=count+25;  }
          inc(counter);
        end
        else
        begin
          if (previous=0) and (dummy1<>0) then
            begin
```

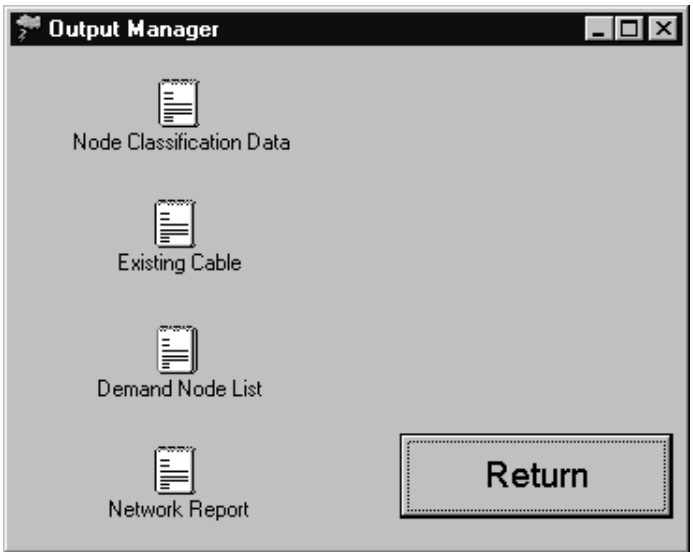
```
              mybitmap.Canvas.moveto(PGX,PGY);
            end
            else
            begin
              mybitmap.canvas.pen.color:=rgb(count,count,count);
              mybitmap.canvas.pen.width:=1;
              mybitmap.Canvas.lineto(PGX,PGY);
            end;
            previous:=dummy1;
          end;
      end;
  closefile(buffer);
  treegraph.canvas.draw(1,1,mybitmap);
  if savebox.checked=true then Mybitmap.SaveToFile('data\graphic.bmp');
  mybitmap.free;
end;

procedure Ttreegraph.ReturnClick(Sender: TObject);
begin
  treegraph.close;
end;

procedure Ttreegraph.ZoomClick(Sender: TObject);
begin
  XZoom:=10*strtoint(XZoomEdit.text);
  YZoom:=10*strtoint(YZoomEdit.text);
  FZoom:=strtoint(ZoomEdit.text);
  dgraph.click;
end;

procedure Ttreegraph.FormCreate(Sender: TObject);
begin
  FZoom:=1;
  XZoom:=0;
  YZoom:=0;

end;

procedure Ttreegraph.PrintClick(Sender: TObject);

var PX,PY,PRX,PRY,radius,startX,StartY,FinishX,FinishY: real;
   PHeight,PWidth:integer;
   RadiusX,RadiusY,PGX,PGY:longint;
   i,ii,count,head,counter,NumberofArcs,dummy,dummy1: integer;
   buffer:textfile;
   cost:real;
   BoxSize,TextSize,previous:integer;
   drawtype:string[1];
   dummy2:string;
   mybitmap:Tbitmap;

begin
  print.caption:='Working';
  printer.begindoc;
  PHeight:=trunc((9/10)*printer.pagewidth);   {scaling for data}
```

```
 PWidth:=printer.pagewidth;
 BoxSize:=12;
 count:=5;
   {Draw offlimit/restricted areas}
 if offlimit.checked=true then
 begin
 assignfile(buffer,'data\offlimit.txt');
 reset(buffer);
 printer.canvas.pen.width:=1;
 printer.canvas.pen.color:=clRed;
 while not eof (buffer) do
   begin
     readln(buffer,drawtype,startX,StartY,FinishX,FinishY,dummy2);
     PX:=((StartX-(5000+XZoom))/755)*PWidth*FZoom;       {scaling factors}
     PY:=((((39460-YZoom)-StartY)/675)*Pheight)*FZoom;{distortion is possible}
     PRX:=((FinishX-(5000+XZoom))/755)*PWidth*FZoom;
     PRY:=((((39460-YZoom)-FinishY)/675)*Pheight)*FZoom;
     PGX:=trunc(PX);
     PGY:=trunc(PY);
     radiusX:=trunc(PRX);
     radiusY:=trunc(PRY);
     if drawtype='M' then printer.Canvas.moveto(PGX,PGY);
     if drawtype='L' then printer.canvas.lineto(PGX,PGY);
     if drawtype='C' then printer.canvas.ellipse(PGX,PGY,radiusX,radiusY);
   end;
 closefile(buffer);
 end;
{draw existing cable}
 assignfile(buffer,'data\exist.txt');
 reset(buffer);
 printer.canvas.pen.color:=rgb(count,count,count);
 printer.canvas.pen.style:=psSolid; {dash,dot,dashdot etc p301}
 printer.canvas.pen.width:=3;
 if existc.checked=true then
 begin
 while not eof (buffer) do
   begin
     readln(buffer,drawtype,FinishX,FinishY,dummy2);
     PX:=((FinishX-(5000+XZoom))/755)*PWidth*FZoom;       {scaling factors}
     PY:=((((39460-YZoom)-FinishY)/675)*Pheight) *FZoom; {distortion is possible}
     PGX:=trunc(PX);
     PGY:=trunc(PY);
     PGX:=trunc(PX);
     PGY:=trunc(PY);
     if drawtype='L' then printer.canvas.lineto(PGX,PGY) else printer.canvas.moveto(PGX,PGY);
   end;
 end;
 closefile(buffer);
 {Draw Legend/Text Data}
 if Legendbox.checked=true then
 begin
   printer.canvas.pen.width:=1;
   printer.canvas.moveto((PWidth div 2 + trunc(0.29*PWidth)),trunc(0.84*Pheight));
   printer.canvas.lineto((PWidth div 2 +trunc(0.29*PWidth)),trunc(Pheight));
   printer.canvas.moveto((PWidth div 2 +trunc(0.29*PWidth)),trunc(0.84*Pheight));
```

```
    printer.canvas.lineto((PWidth),trunc(0.84*Pheight));
    printer.canvas.textout((PWidth div 2 -trunc(0.15*PWidth)),trunc(0.02*Pheight),'NTC Fiber Optic
Cable System');
    printer.canvas.textout((PWidth div 2 +trunc(0.37*PWidth)),trunc(0.85*Pheight),'Legend');
    printer.canvas.textout((PWidth div 2 +trunc(0.3*PWidth)),trunc(0.88*Pheight),'NODE');
    printer.canvas.pen.color:=clRed;
    printer.canvas.pen.width:=2;
    printer.canvas.polyline([Point(((PWidth div 2) +trunc(0.47*PWidth)-
boxsize),(trunc(0.895*Pheight)+BoxSize)),    {1st head}
      Point((PWidth div 2 +trunc(0.47*PWidth)+BoxSize),(trunc(0.895*Pheight)+BoxSize)),
      Point((PWidth div 2 +trunc(0.47*PWidth)+BoxSize),(trunc(0.895*Pheight)-BoxSize)),
      Point((PWidth div 2 +trunc(0.47*PWidth)-BoxSize),(trunc(0.895*Pheight)-BoxSize)),
      Point((PWidth div 2 +trunc(0.47*PWidth)-BoxSize),(trunc(0.895*Pheight)+BoxSize))]);
    printer.canvas.pen.color:=rgb(5,5,5);
    printer.canvas.pen.width:=1;
    printer.canvas.textout((PWidth div 2 +trunc(0.3*PWidth)),trunc(0.91*Pheight),'ARC');
    printer.canvas.moveto((PWidth div 2 +trunc(0.45*PWidth)),trunc(0.925*Pheight));
    printer.canvas.lineto((PWidth div 2 +trunc(0.49*PWidth)),trunc(0.925*Pheight));
    printer.canvas.textout((PWidth div 2 +trunc(0.3*PWidth)),trunc(0.94*Pheight),'Existing Cable');
    printer.canvas.pen.width:=3;
    printer.canvas.moveto((PWidth div 2 +trunc(0.45*PWidth)),trunc(0.955*Pheight));
    printer.canvas.lineto((PWidth div 2 +trunc(0.49*PWidth)),trunc(0.955*Pheight));
    printer.canvas.pen.width:=1;
  end;
  assignfile(buffer,'data\buffer.txt');
  reset(buffer);
  readln(buffer,NumberofArcs);
  NumberofArcs:=NumberofArcs;
  counter:=0;
  readln(buffer,dummy,dummy1,PX,PY);
  PX:=((PX-(5000+XZoom))/755)*PWidth*FZoom;       {scaling factors}
  PY:=((((39460-YZoom)-PY)/675)*Pheight)*FZoom; {distortion is possible}
  PGX:=trunc(PX);
  PGY:=trunc(PY);
  TextSize:=12;
  printer.canvas.pen.style:=psSolid; {dash,dot,dashdot etc p301}
  printer.canvas.pen.color:=clBlue;
  printer.canvas.pen.width:=2;
  printer.canvas.polyline([Point((PGX-BoxSize),(PGY+BoxSize)),    {1st head}
    Point((PGX+BoxSize),(PGY+BoxSize)),Point((PGX+BoxSize),(PGY-BoxSize)),
    Point((PGX-BoxSize),(PGY-BoxSize)),Point((PGX-BoxSize),(PGY+BoxSize))]);
  if nodeN.checked=true then printer.canvas.textout((PGX),(PGY-TextSize),inttostr(dummy));
  count:=5;
  i:=1;
  printer.canvas.pen.width:=i;
  printer.Canvas.moveto(PGX,PGY);
  head:=0;
  previous:=dummy1;
  while counter<>NumberofArcs do
    begin
      readln(buffer,dummy,dummy1,PX,PY);
      PX:=((PX-(5000+XZoom))/755)*PWidth*FZoom;
      PY:=((((39460-YZoom)-PY)/675)*Pheight)*FZoom;
      PGX:=trunc(PX);
      PGY:=trunc(PY);
```

```
if previous=0 then
  begin
    printer.canvas.pen.color:=clBlue;
    printer.canvas.pen.width:=2;
    printer.canvas.polyline([Point((PGX-BoxSize),(PGY+BoxSize)),
      Point((PGX+BoxSize),(PGY+BoxSize)),Point((PGX+BoxSize),(PGY-BoxSize)),
      Point((PGX-BoxSize),(PGY-BoxSize)),Point((PGX-BoxSize),(PGY+BoxSize))]);
  if nodeN.checked=true then printer.canvas.textout((PGX),(PGY-TextSize),inttostr(dummy));
  end;
if (dummy1=0) then
  begin                          {tail}
    if (dummy<>0) and (previous<>0) then
      begin
        printer.canvas.pen.color:=rgb(count,count,count);
        printer.canvas.pen.width:=1;
        printer.Canvas.lineto(PGX,PGY);     {need for null arcs}
      end
      else
      begin
        printer.Canvas.moveto(PGX,PGY);     {need for null arcs}
      end;
    printer.canvas.pen.color:=clBlue;
    printer.canvas.pen.width:=2;
    printer.canvas.polyline([Point((PGX-BoxSize),(PGY+BoxSize)),
      Point((PGX+BoxSize),(PGY+BoxSize)),Point((PGX+BoxSize),(PGY-BoxSize)),
      Point((PGX-BoxSize),(PGY-BoxSize)),Point((PGX-BoxSize),(PGY+BoxSize))]);
    if nodeN.checked=true then printer.canvas.textout((PGX),(PGY-TextSize),inttostr(dummy));
    if dummy<>0 then readln(buffer,dummy,dummy1,PX,PY);
    PX:=((PX-(5000+XZoom))/755)*PWidth*FZoom;
    PY:=((((39460-YZoom)-PY)/675)*Pheight)*FZoom;
    PGX:=trunc(PX);
    PGY:=trunc(PY);
    previous:=dummy1;
    printer.Canvas.moveto(PGX,PGY);
    printer.canvas.polyline([Point((PGX-BoxSize),(PGY+BoxSize)),
      Point((PGX+BoxSize),(PGY+BoxSize)),Point((PGX+BoxSize),(PGY-BoxSize)),
      Point((PGX-BoxSize),(PGY-BoxSize)),Point((PGX-BoxSize),(PGY+BoxSize))]);
    if (nodeN.checked=true) then printer.canvas.textout((PGX),(PGY-TextSize),inttostr(dummy));
    printer.Canvas.moveto(PGX,PGY);
    inc(i);
   {  count:=count+25;  }
    inc(counter);
  end
  else
  begin
    if (previous=0) and (dummy1<>0) then
      begin
        printer.Canvas.moveto(PGX,PGY);
      end
      else
      begin
        printer.canvas.pen.color:=rgb(count,count,count);
        printer.canvas.pen.width:=1;
        printer.Canvas.lineto(PGX,PGY);
      end;
```

```
                previous:=dummy1;
            end;
        end;
      closefile(buffer);
      printer.enddoc;
      print.caption:='Print';
end;

procedure Ttreegraph.UpDownClick(Sender: TObject; Button: TUDBtnType);
begin
  zoom.click;
end;

procedure Ttreegraph.RLClick(Sender: TObject; Button: TUDBtnType);
begin
  zoom.click;

end;

procedure Ttreegraph.zoomEditChange(Sender: TObject);
begin
  zoom.click;
end;

end.
```




```
unit Output;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, OleCtnrs, FileCtrl;

type
  Toutputform = class(TForm)
    Button1: TButton;
    OleContainer1: TOleContainer;
    OleContainer4: TOleContainer;
    OleContainer5: TOleContainer;
    OleContainer2: TOleContainer;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  outputform: Toutputform;

implementation
uses ntc;
{$R *.DFM}
procedure Toutputform.Button1Click(Sender: TObject);
begin
   main.show;
   outputform.close;
end;

end.
```

LIST OF REFERENCES

1. Goulette, Dana, " A Methodology for Modeling Subjective Data Encapsulation at the National Training Center, Fort Irwin, CA.," Master's Thesis, Naval Postgraduate School, Monterey, Califiornia, March 1997.

2. Project Manager for Training Devices, "Systems Specification for the National Training Center Range Data Measurement Subsystem Upgrade," Orlando, Florida, 20 December 1991.

3. Maciaszek, L.A., *Database Design and Implementation*, Prentice-Hall, Inc, Englewood Cliffs, New Jersey, 1990.

4. Kroenke, David M., *Database Processing: Fundamentals, Design, and Implementation 5th edition*, Prentice-Hall, Inc, Englewood Cliffs, New Jersey, 1995.

5. Petho, Frank C., *Human Factors in System Design - Research Methodology and Performance Measurement*, Operations Research Department, Naval Postgraduate School, Monterey, California, 1995.

6. National Training Center Vulture Team, "Report Generator Briefing," obtained from LTC Dwight Raymond, Barstow, California, June 1996.

7. Ahuja, Ravindra K., Magnanti, Thomas L., and Orlin, James B., *Network Flows: Theory, Algorithms, and Applications*, Prentice-Hall, Inc, Upper Saddle River, New Jersey, 1993.

8. Lawphongpanich, Siriphong, *OA4204: Network Flows and Graphs*, Operations Research Department, Naval Postgraduate School, Monterey, California, 1995.

9. Osier, Dan, Grobman, Steve, and Batson, Steve, *Teach Yourself Delphi 2 in 21 Days*, SAMS Publishing, Indianapolis, Indiana, 1996.

10. Buck, Stephen D., "A Database Management System to Manipulate Data Collected at the National Training Center, Ft. Irwin Ca.," Master's Thesis, Naval Postgraduate School, Monterey, Califiornia, June 1987.

11. Litwin, Paul, "Fundamentals of Relational Database Design," adapted from Microsoft Access 2 Developer's Handbook, Sybex, 1994. Downloaded 2 August 1996 from http://www.shareware.com

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   8725 John J. Kingman Road, Ste 0944
   Fr. Belvior, VA 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, California 93943-5101

3. Professor Samuel Parry 2
   Code OR/Py
   Naval Postgraduate School
   Monterey, California 93943-5002

4. LTC Charles H. Shaw III 1
   Code OR/Sc
   Naval Postgraduate School
   Monterey, California 93943-5002

5. LTC A. Dwight Raymond 4
   Chief, NTC Tactical Analysis and Computer Support Division
   National Training Center
   Fort Irwin, California 92310

6. Director, TRAC 1
   ATTN: ATRC - FA
   Ft. Leavenworth, Kansas 66027

7. CPT Kirk C. Benson 2
   25 Anderson Road
   Cheney, Washington 99004

8. Director, Us Army Tradoc Analysis Center - Monterey 1
   ATTN: ATRC-RDM (LTC Wood)
   Monterey, California 93943