SCHOOL OF INDUSTRIAL AND SYSTEMS ENGINEERING

A PROTOTYPE SUPPLY POINT LOCATOR FOR US ARMY DIVISIONS

Master's Thesis

By

CPT Larry L. Wheeler, USA

December 1988

GEORGIA INSTITUTE OF TECHNOLOGY
ATLANTA, GEORGIA 30332

A PROTOTYPE SUPPLY POINT LOCATOR
FOR U.S. ARMY DIVISIONS

A THESIS
Presented to
The Faculty of the Division of Graduate Study

By

CPT Larry L. Wheeler, USA

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in Systems Engineering

Georgia Institute of Technology
December, 1988

ACKNOWLEDGMENTS

This thesis was prepared under the supervision and direction of Professor Donovan Young. The generous giving of his time, ideas, and constructive criticism are greatly appreciated. Thanks also to Professor Leon McGinnis for providing his ideas and suggestions. I also acknowledge the substantial assistance of Walter White, a Computer Science major Co-op student, in the sizeable programming task of making the prototype SPL work. His technical assistance with the data structures of the program and portions of the graphics programming proved invaluable during the software development phase of the project.

Administrative support was provided by the U.S. Army Institute for Research in Management Information and Computer Science, Georgia Institute of Technology. I thank Dr. Mike Evans for his interest and support throughout the project.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

Figure Page

SUMMARY

A demonstration prototype Decision Support System (DSS) has been developed, documented, and validated for the interactive selection of supply point locations within a U.S. Army Division area of operations.

The DSS, called a Supply Point Locator (SPL), assumes a sophisticated software and hardware environment which uses a tactical terrain database as input for the analysis of terrain and selection of potential supply points. Locations of customers and potential supply point locations and the road network data connecting them are assumed to be in machine-usable form. The SPL first assists the user in providing and revising problem specific data on demands, supplies, vulnerability, risk guidance, and supported unit locations. Then the SPL assists the user in performing an interactive optimization procedure based on the location-allocation optimization model that balances "lag" (responsiveness) against "loss" (vulnerability). The SPL helps in identifying efficient solutions, in investigating the consequences of uncertainties or changes in data, and in integrating considerations not practically quantifiable in the location-allocation formulation such as mission, enemy, terrain, vulnerability, and commander's guidance. [illegible]

CHAPTER I

INTRODUCTION

The effective provision of combat service support (CSS) for the commander's tactical plan is a continuous and vital function of all CSS units. Logistics planners must insure that tactical and CSS plans are made concurrently and in coordination with operations planning, and that tactical schemes of maneuver and fire support are supportable by CSS elements. History has reminded us countless times of the absolute necessity of effective supply operations in the conduct of warfare.

There are no established quantitative procedures for planning CSS in support of tactical operations. Location of support areas to effectively support tactical operations is traditionally one of the more difficult decisions faced by logistics planners and CSS unit commanders. Analysis of the risks involved and the trade-offs necessary when planning support area locations in a combat zone must often be accomplished hurriedly and with incomplete information about both the friendly and enemy situation. Current Army doctrine specifies that these decisions are made through a risk-benefit analysis approach [7]. Any system to relieve the planner or decision maker of trivial and mundane tasks conducive to automation, such as basic terrain analysis and vulnerability analysis, could substantially enhance the efficiency of support area location decisions.

Battlefield information systems now planned will include an intelligence database, an operations database, a tactical terrain database, and a CSS database. Each of these data sources will provide essential information to an automated supply point locator. The intelligence database will provide data concerning the level of threat as a function of enemy location, capabilities, size, and intentions. A tactical terrain data base also maintained by the G2/S2 section will provide topographical information and relief data from which site suitability decisions are made. An automated situation map and operations database will provide information on friendly unit locations, sizes, missions, capabilities, and tactical priorities. The CSS database will provide demand data for operational requirements for supported units, supply capacity limitations, and supply status by category of supply.

The purpose of this research is to provide a prototype of a Decision Support System (DSS) or Tactical Decision Aid for supply point location for use by CSS planners in a U.S.

Army Division. It will assist planners in selecting, from among all potential support area locations, those locations that minimize total resupply travel distance and vulnerability consistent with mission requirements and the tactical situation.

The Decision Support System presented, subsequently referred to as the SPL (Supply Point Locator) would in no way be intended to replace the human decision maker, or as the ultimate tool for locating logistical facilities in a combat zone. The SPL should assist logistics planners at various echelons, particularly at Army Division level, to make use of operations research techniques and available battlefield data to solve the problem of locating supply facilities. Prior knowledge of operations research is not required for the user of this system. Interactive use of this system in the selection process permits use of the previous knowledge and experience of the logistician as well as the advantages of automation. The system would supplement what is now a subjective decision process by analyzing information from the proposed battlefield information systems to help quantify the tradeoffs involved in locating supply points.

1.1 Supply Point Location in U.S. Army Divisions

The "mission" is the primary consideration for the location of support areas and CSS units and facilities in the combat zone. All CSS units/elements must be far enough forward to provide rapid response to all supported units. A rapidly moving friendly offensive posture, for example, may dictate that support areas be far forward, very near combat units in order to provide a higher level of responsiveness for maintenance and medical support required in this situation, while accepting vulnerability to enemy artillery and other indirect fire systems and a higher level of threat. A defensive posture, requiring less maintenance and medical support but more Class V supplies (ammunition), could require location of support areas further to the rear with ammunition transfer points (ATP's) located forward.

In planning support for combat operations, CSS commanders and planners recognize that trade-offs between vulnerability and responsiveness are necessary in determining locations for support facilities. The question of whether the benefits of locating in a particular area and providing more responsive support outweigh the risks of locating in that area is often difficult. Unfortunately, there are no analytical rules to assist CSS planners in making these risk-benefit decisions. Circumstances must be assessed and risks measured, and a decision on the best course of action reached. CSS doctrine specifically cites the "location of support areas" decision in FM 63-2:

> The location of support areas is one example of CSS risk analysis. In order to provide the required support responsively, is it necessary to locate CSS activities within enemy direct support artillery range? There is clearly a risk involved, but it may be necessary to assume this risk if that is the only way in which critical support can be provided [8].

Since circumstances that require risk-benefit analysis are difficult to anticipate, since there is seldom time for systematic analysis, and since lives depend on these decisions, effective risk-benefit analysis is essential.

There are general principles of CSS planning and site selection which should be applied in the location of support areas whenever possible. Current U.S. Army doctrine [8] specifies that in the location of the DISCOM Command Post, which is the control center for the DSA, the particular site should:

- Be adaptable to the use of collective Nuclear, Biological, and Chemical (NBC) protection equipment.
- Provide sufficient area for dispersion.
- Be near subordinate units and installations.
- Provide adequate sites for communications.

- Provide adequate cover, concealment, drainage, hardstand, and roads.
- Be located away from probable enemy targets and other likely areas of enemy attack to reduce probability of damage and facilitate defense.
- Provide aircraft landing sites.
- Be beyond the range of enemy direct support artillery.

These criteria are offered as guidance. They involve tradeoffs. For example to be "near" supported units while "away" from probable enemy targets may not be possible, particularly if the supported units are combat units. The criteria are always subject to circumstances and the current tactical situation. It should also be noted that while requirements vary in degree, these principles apply to location of support areas at brigade and battalion level as well as division.

In summary, the objective of CSS planners in locating supply points is to balance between the most responsive support and the least vulnerability to enemy actions. Techniques for meeting this objective are also specified by doctrine [8]. They require planners at division and brigade levels to:

- Review the supported units' missions and available operational data.
- Review available intelligence data.

- Conduct a map reconnaissance to determine potential locations.
- Conduct an actual ground reconnaissance of the locations before a final location is determined.

Each potential location is examined to determine its suitability for use as a support facility location in terms of the doctrinal requirements. Depending on the mission, it may not be possible to satisfy all of these criteria, and tradeoffs are made in order to provide the best support.

After all of the required information is obtained, map and ground reconnaissance have been completed, and the enemy and friendly situations are updated, a subjective location decision is made. Subjective criteria for the decision are expressed in terms of the familiar Army acronym METTT (mission, enemy, troops available, terrain and weather, and time available).

This technique of supply point location, if done properly, is tedious, can be very time consuming, and is subjective. The terrain analysis task may require several hours to complete properly. After map reconnaissance and terrain analysis is completed, the ground reconnaissance task can easily take four to eight hours to complete. On the high-intensity battlefield of the future, time will be a scarce reso' rce. A timely decision method must also be a primary objective of supply point location. A more efficient and time sensitive method of supply point location is needed.

One method of improving the efficiency of supply point location in U.S. Army Divisions is to implement a Decision Support System, or Tactical Decision Aid, which would accomplish the required analysis through interactive computer usage by CSS planners. A DSS capable of performing this task requires three key elements: an efficient model, a computer capability, and the data required for the model to perform the analysis [22]. Suitable technology currently exists for each of these elements and will be available to Army CSS planners in the field within the next decade.

1.2 The Location-Allocation Model

1.2.a Model Description

The problem of locating supply facilities in a combat zone falls under a category known as location-allocation problems. Location-allocation problems involve the optimal placement of one or more sources with respect to various destinations requiring resupply (demand). The "location-allocation" name indicates the natural two-phase structure of this type of problem: if the locations of sources were fixed, there would be a best allocation of supplies to demands for the fixed set of source locations, so the best set of source locations is that which gives the best among these best allocations.

There is a modeling concept in operations research for the solution of the location-allocation problem. Both the location and the allocation aspects of the problem can be modeled using network analysis and location theory. This model has been used successfully to solve a wide variety of facilities location problems. Both Anderson [1] and Link [34] have successfully demonstrated the model's use and effectiveness in the location of military support facilities.

Doctrinal selection criteria described above can be grouped into three collective areas: site (node) suitability, road (arc) suitability, and vulnerability. The grouping of these criteria, required accuracy, and the time sensitivity of supply point location in combat, all suggest mathematical optimization techniques for the efficient solution of this problem.

In modeling this location-allocation problem each element described by doctrine above must be defined in network analysis terms. Each location on the battlefield is defined as a node. Consistent with military terminology, a node will normally be a unit location or an identifiable terrain feature such as a bridge or a road intersection. In this problem all supported units and candidate supply point locations will be located at nodes. Within an area of operations there will be a road or highway system. Each road which connects two nodes is a "link." The series of nodes connected by the links defines the "network" used in the analysis of the system.

All supply point or support area locations that are to be chosen are defined as "sources," i.e. these points are the source of supply for the supported units. Sites that are suitable (not necessarily optimal) for the location of a supply point are referred to as "potential sources" or "candidate locations" for a supply point.

Each supported unit will have requirements for each class of supply. These requirements, consistent with military terminology, are termed "demands." "Demand nodes" are those points from which supplies or services are demanded by supported units. Demand nodes and supported units' locations will normally be the same.

Quantities will be expressed in units for an arbitrary time span, normally one 24-hour day. Thus a unit such as "short tons" represents short tons per day or per other time unit.

"Capacities" will be defined as the maximum quantity of supplies which can be supported by either a node or a link in the system. All capacities in the supply point location problem will be in short tons per day.

A final term which must be defined in the context of supply point location using mathematical programming methods is "costs." In the transportation problem, costs normally refer to the price you pay for the distribution of commodities throughout a system. This will usually include either fixed or variable costs of the commodities and transportation. An important additional consideration for military operations in a combat zone is the effect of hostile activity on the system. In this problem, vulnerability to hostile activities is a necessary consideration and should weigh heavily in any solution. Responsiveness is also a key consideration and will normally be obtained at the expense of vulnerability. Responsiveness and vulnerability are the two key elements in the vulnerability analysis portion of the problem and can both be satisfactorily expressed in an objective function, as will be seen in the following chapter.

In this problem, "responsiveness" is a function of travel time, while "vulnerability" is a function of the characteristics of the sites (nodes) and the roads (links), and travel time. Distance is indirectly important because of its effect on travel time.

For the problem of locating support facilities, costs will be represented in terms of "expected losses of commodities" (in short tons) and "expected commodity lag time" rather than in dollars, travel times, or distances. Costs will be considered proportional to quantity.

To illustrate the concept of capacity and costs, an MSR is a good example. Military planners have traditionally attempted to choose a Main Supply Route which will accommodate high volumes of relatively high speed traffic, and minimize travel time as well as the effects of hostile activities and interdiction along the route. This would be an example of a "high capacity, low cost" supply route.

The supply points are selected from a finite set of acceptable locations (nodes) previously determined by a site suitability subroutine. This may be best visualized as selecting source nodes in a distribution network.

This model requires the use of two separate layers of decision variables. The first must decide which nodes to open, or where should the supply point be located (potential sources). The second level of decision variable in the model decides when certain source nodes are open, what are the best routes between the sources and the sinks (supported units)?

The location layer of the problem involves establishing a location for one or more distribution centers for the production or distribution of goods; these centers are supply points, which distribute military supplies of all classes, with the possible exception of water, and in some situations, ammunition. The allocation portion of the problem involves distribution of the supplies from these sources, through intermediate points or nodes such as road intersections, towns, or cities, to the demand points or supported units. When either the source location or the allocation of the distribution of commodities is considered separately the solution to the problem is simplified significantly. However, proper analysis of this problem requires simultaneous solution of both the location and the allocation aspects. The location of the supply points must be made at the same time as the determination of the most optimal allocation of commodities. Applying Anderson's analysis [1] to the problem of locating supply pcints, the problem can then be stated as the following:

Given:

The supported unit locations, $\{x_j, y_j\}$ and demands, $\{b_j\}$.

The set of candidate supply point locations, $\{x_i, y_i\}$.

Distances between points, $\{d_{ij}\}$.

Vulnerability data for links and nodes, $\{q_{ij}\}$ and $\{p_i\}$.

Responsiveness criteria, $\{w\}$.

The number of supply points being located, (r).

Select:

The location (x_i, y_i) of the supply point(s).

The required capacity of each source (k_i).

<u>1.2.b Mathematical Formulation</u>. A mathematical formulation of the Location-Allocation problem which best describes the supply point location problem is given by Ellwein [14].

$$\text{Minimize } Z = \sum_{i}^{m} \sum_{j}^{n} d_{ij} x_{ij} + \sum_{i}^{m} g_i \left(\sum_{j} x_{ij}\right) + \sum_{i}^{m} f_i y_i$$

Subject to:

$\sum_{i}^{m} x_{ij} \geq b_j, \; j = 1,2,\ldots n$ "demand at supported unit j"

$\sum_{j}^{n} x_{ij} \leq \alpha_i y_i \quad i = 1,2,\ldots m$ "supply at supply node i"

$\sum_{i}^{m} y_i \leq r$ "maximum number of supply points"

$$x_{ij} \geq 0$$

$$y_i = 0 \text{ or } 1$$

This formulation uses zero-one decision variables which indicate whether a potential source is "open" (a source) or "closed" (not a source). y_i=0 indicates a closed node while

$y_i=1$ indicates an open supply node. The x_{ij} are decision variables which represent the non-negative supply output that supply node i supplies to demand node j. The first constraint set indicates that demand at demand node j must be satisfied. The second constraint set states that supply capacity at each supply node cannot be exceeded. The third constraint is a system configuration constraint which states that there cannot be more supply points than open nodes.

The formulation presented above is more general than that used by the SPL. The fixed cost (f_i) of opening a military supply point is assumed to be the same for all sites; therefore, fixed costs may be assumed to be zero. This results in an objective function which may be expressed as:

$$\text{Minimize } Z = \sum_{i}^{m} \sum_{j}^{n} d_{ij} x_{ij}$$

where x_{ij} is the quantity (in short tons) required to satisfy the demand for a supported unit at demand point j traveling over the shortest route, d_{ij}. The third constraint may be deleted since the user is specifying the set of nodes to be considered as supply points.

A dummy demand point may be created to absorb all excess supply. This will be a necessary condition for the transportation algorithm. The cost of supplying the dummy demand point from any source is then zero.

The above formulation does not include vulnerability criteria. Following a rigorous discussion of these criteria, formulation of the completed objective function used by the SPL is presented in Chapter III. In this formulation, distance d_{ij} will be replaced by an effective cost c_{ij} that balances responsiveness against vulnerability.

1.3. Battlefield Information Environment

The battlefield information environment of the next decade will be much more sophisticated than that of today. This information environment will be significantly enhanced by extensive use of automated databases, secure data links and digital communications, tactical decision aids, and local area networks (LANS). Network communications systems currently being used and fielded, such as the Automated Joint Interoperability of Tactical Command and Control System (JINTACCS), will be followed in the next decade by more capable and more sophisticated systems such as the Joint Tactical Information Distribution System (JTIDS) and command control and communications systems (C^3I), employing some degree of artificial intelligence [32],[33]. Integrated automated systems at all levels using local area networks will assist in the gathering, storage, analysis, and reporting of battlefield information. Use of Tactical

Decision Support Systems or Tactical Decision Aids by planners and commanders at all levels and within all functional areas will continue to improve response times by automation of mundane yet critical tasks, which would otherwise be very time consuming but must be accomplished in order for well informed decisions to be made.

Tactical terrain data bases and digital maps will enable quick retrieval of all data needed for these systems to analyze terrain. Databases for each functional area of operations, intelligence, combat service support, fire support, air defense, etc. will be available to planners. An intelligence database will provide information about the enemy and the level of threat in terms of locations, capabilities, order of battle, and intentions. The operations database will provide more detailed data on the friendly situation including such information as mission, unit locations, posture, operations graphics for the current and future situations, controlled supply rates, and operational readiness. A CSS database will provide equipment status, supply status, unit demand data, personnel status, etc. similar to the systems in use today in Army Divisions, but in a network and much more accessible. These databases will employ standardized protocols within a local area network and should be accessible by staff planners at all levels of command down to and including battalion level.

The SPL assumes a hardware and software environment in which all combat, combat support, and combat service support units down to and including battalions and separate companies will have organic AT-class microcomputers with enhanced graphics capabilities and enhanced memory (greater than 100 Mb). Units will be equipped with assorted military applications software as well as a variety of tactical decision aids which interface with higher, lower, and adjacent headquarters in a system of local area networks.

1.4 Overview

The introduction is intended to familiarize the reader with the problems and the challenges of supply point location in a U.S. Army combat zone, and the basic design considerations of an interactive optimization technique used to aid logistics planners in this task. This chapter has discussed current U.S. Army doctrine and some of the problems of supply point location in an Army Division, a mathematical description of the location-allocation model used in this research to locate supply points, a description of the current and projected battlefield information environment, and a discussion of the hardware and software environments assumed by the Supply Point Locator.

Chapter II will present a description and discussion of Location-Allocation methods and prototypes as well as a discussion of previous research conducted in this area. Two prototype decision support systems with applications to the SPL will be discussed as well as doctrinal issues for supply point location and expected developments in map and roadway data. Chapter II also includes the mathematical formulation of the location-allocation methodologies used in this research and a review of solution methodologies as well as discussion of several network flow heuristics used in the solution to the supply point location problem.

The interactive optimization approach to supply point location used in this research is discussed and summarized in Chapter III. This includes discussion of node and link vulnerability, use of semiscaled networks in the network display, and the interactive interface used by the prototype DSS for supply point location. Chapter III also summarizes the computational methods used by the SPL.

A detailed description of the implementation of the SPL will be presented in Chapter IV. This chapter will include a discussion of the development of the hardware and software used by the SPL and a description of the architecture of the SPL, including program listings and documentation.

Experimentation with the SPL using a simulated combat scenario and terrain data will be presented in Chapter V. Chapter VI contains the results and conclusions of this research, including a discussion of suggested improvements and further developmental possibilities for the prototype system.

CHAPTER II

LOCATION-ALLOCATION METHODS AND PROTOTYPES

2.1. Water Point Location

A prototype Decision Support System for the location of military water points developed by Anderson [1] uses an interactive heuristic approach to the solution of the location-allocation problem as a series of network flow problems. User input specifies potential water point locations, supported unit locations, demands, capacity constraints, and distances. All user and potential water point locations are initially identified as nodes, either sources or sinks, and sometimes both. For each user-selected set of open source nodes, there is a network flow problem that is solved to determine the minimum total resupply effort and the amounts and routings of resupply trips that minimize this effort. The network flow problem is solved by a two-step procedure. The shortest route problem is solved from each open node (water point) to the demand nodes. The shortest distance values are then used to form a cost matrix which is used in the solution of the transportation algorithm to best match demands to the open sources.

Sensitivity analysis is then presented to the user enabling him to analyze which points are not fully used, which points are used to maximum capacity, which points are not used at all, and shadow prices for opening or closing source nodes. The user then decides which sites should be opened and which ones should not, and then re-solves the network flow problem for a different set of open nodes. This interaction - choosing a set of open sources, seeing the network-flow consequences and sensitivity-analysis reports, revising the set of open sources accordingly, and repeating - continues until the user no longer wishes to try further sets of open sources.

A significant limitation of this procedure is that the effort-minimization objective function fails to represent important selection factors. For example, a Division Engineer would usually prefer water points further away from combat units if the points were more defensible. If the objective function could represent exposure to losses rather than distances, the analysis would be more realistic.

Another limitation of Anderson's prototype system is that the sensitivity analysis reports do not explicitly give the values of test quantities that can be computed from published heuristics for the location-allocation problem [1]. The Kuehn-Hamburger Heuristic [25] recommends sources to open and close; if these recommendations were reported, the performance of the interactive procedure in solving the location-allocation problem could be guaranteed; simply by including the recommended solutions among those tried, the user could be guaranteed of a solution at least as good as the solution that would be found by the heuristic.

2.2. Ammunition Transfer Point Location

In a prototype tactical decision aid for the location of ammunition transfer points (ATP's) in a US Army division area of operations based on field artillery role assignments, Link and Callahan [34] developed an interactive system that provides a procedure for the subjective evaluation of varying Field Artillery roles in the selection of ATP locations. The system includes an analysis of supported unit mission, battlefield posture, and targeting priorities which are used to weight contributions to the overall fire support mission of the division.

Similar to the Water Point Locator [1], this system allows input of the road network in the area of operations. Candidate ATP locations, firing unit locations, and road intersections are entered interactively, followed by the generation of a road (link) network connecting each of the locations. The system determines the "best" location for the ATP through an application of 1-median location theory. Through subsequent analysis of data determined by the system based upon the 1-median location theory, the user is able to determine the best location for the ATP.

In the solution of the location-allocation problem used in the Ammunition Transfer Point Locator, the Shortest Route Algorithm is first solved to determine the shortest distances from candidate locations to supported units. After the distances have been determined, 1-median calculations are accomplished using the weighted supported unit locations to determine a median value/location, with which the user then rank orders the candidate ATP locations for further consideration.

The ATP Locator also provides sensitivity analysis for each of the problem parameters and generates a "circle of effectiveness" within which supported units may reposition without compromising the ATP location or necessitating a change in the ATP location. Problem parameters which may be analyzed for sensitivity include target values, commander's priorities, and supported unit locations.

By including mission-related factors, the ATP location system avoids the main limitation of the Water Point Location system. The objective function can represent important factors such as battlefield posture and contribution to fire support missions, although in a subjective manner.

A severe limitation of both the Water Point and ATP location prototypes is their data demands. No decision aid can be practical if it requires the user to input entire networks by hand, or if it requires subjective factors to be input by the user for large numbers of nodes or links. It will be necessary to use routinely issued roadway network data sets in any practical successors to these prototypes. It will also be necessary to replace subjective data with data extracted from battlefield databases.

Further limitations of both the Water Point and ATP location prototypes stem from their implementation hardware. The Chromatics CG1999 computer has 512 x 512 color graphics and a Z-80 CPU, with 48,000 bytes of free storage. Graphics resolution, computing speed, and memory limits prevented the prototypes from handling problems covering more than about one-quarter of a typical division area of operations.

2.3. Doctrinal Issues for Supply Point Location

The purpose of the Supply Point Locator is to interactively select a set of candidate locations for supply facilities that will minimize exposure to loss where exposure depends on the current enemy situation and capabilities, friendly capability to provide security, and cover and concealment. The system will also seek to minimize travel times/distances and number of truckloads of supplies transported through the system.

Current Army doctrine on the location of support facilities in a tactical environment probably represents the major portion of the research conducted in this area. Although formal operations research techniques have not been used, the review and refinement process used in the development of doctrine and operating procedures represents large amounts of practical experience and experimentation and has evolved into current doctrine, which is considered to specify the best methods currently available. These methods are published in Army doctrinal manuals. Current U.S. Army doctrine for the location of supply facilities is contained in FM 100-10 and FM 63-2.

A significant portion of previous research for the development of the Supply Point Locator was directed at doctrinal issues that must be addressed preceding the development of a Tactical Decision Aid of this type. Among these doctrinal issues are node and link suitability, vulnerability, travel times, demand from supported units, and supply capacity.

A study group composed of four U.S. Army officer graduate students at Georgia Tech in 1986, during a special course offering in decision support systems, did a significant amount of research into doctrine and practice in the location of supply facilities [31]. In an effort to generalize the Water Point Location System developed by Anderson [1] to allow location of other types of supply points, the group determined that doctrine requires the conjunctive location of all supply facilities for all classes of supplies. Water point location is a special case of the larger problem of facility location because of the special water-source availability constraints (for other commodities, a supply point is not usually a primary source but can be replenished at any chosen rate). This group determined that the objective function of the Water Point Location DSS was inadequate for the Supply Point Locator. The primary finding of the group was that vulnerability is the underlying criterion and that it is best expressed quantitatively as "expected losses of commodities" per ton per unit time, which can be multiplied by the amount stored at a site and the amount shipped over a road, and multiplied by the storage time at a site and the travel time over a road to estimate total exposure to loss. It was also determined that the minimization of losses due to exposure leads to the same mathematical form as that used in the Water Point Location DSS. Doctrine and practice were found to require an initial screening of sites for suitability. An algorithm for determining node and link suitability was developed by one of the members of the Georgia Tech study group which provided a set of potential supply nodes in a way closely analogous to the water availability criterion used by Anderson [1].

2.4. Developments in Digital Terrain and Map Data

Knowledge of the terrain has always been critical to military planning and operations. Traditionally there have been several sources of this knowledge which include personal reconnaissance, terrain photographs, and maps. As Army systems have begun to become more sophisticated because of the technology bloom of the past decade, the need for advanced methods of terrain analysis and terrain data usage has become very obvious.

Maps and terrain data allow the same piece of ground to be represented many ways. These include conventional hardcopy representations, analog representations such as video signals, and digital representations of the terrain. Current technology provides videomaps, which are analog television images of maps, and "soft maps" which are described as digitized (pixelized) representations of the images of maps. These systems are useful, but offer only limited flexibility for manipulation, and are considered "dumb maps" in the sense that a human must read them and reassign meaning to their content [12].

Several systems currently in the US Army inventory are using advanced techniques for obtaining terrain information. The Pershing II Missile System uses terrain data in its terminal guidance system, and the FIREFINDER counter-mortar and counter-artillery location radar system uses digitalized terrain data in its terrain compensation calculations. Other initial applications have been used in training devices and simulators like the Combined Arms Tactical Training Simulator (CATTS) and the Army Training Battle Simulation System (ARTBASS). Each of these applications, however, use terrain data sets unique to the systems. In 1984, following a two-year study, the Army formally stated a basic operational support requirement for a multipurpose terrain data system. This system, currently being developed, is now known as Tactical Terrain Data (TTD) [12]. The Defense Mapping Agency (DMA) and all of the services are negotiating characteristics of TTD, which is being developed by DMA. The US Army Corps of Engineers and the Engineer Topographic Laboratories at Fort Belvoir, VA are also directly involved in research and development of TTD.

TTD is a basic operational digital set using attributed, unsymbolized data describing the geometry of the earth's surface (elevation data) and the composition and distribution of features at or near the earth's surface (feature data) [12]. TTD will be standardized and distributed through intelligence and supply channels much as hard copy maps are distributed today. The terrain data will be packaged on a standard compact disk and developers envision terrain data for an entire corps area of influence on a single disk. TTD will be used by tactical decision aids at all levels to assist military decision makers. A prototype TTD could be developed as early as 1988, and developers anticipate a system ready for fielding by 1992.

TTD or a similar system is a necessary pre-condition for most tactical decision aids being developed for use in the 1990's. The Supply Point Locator will interface directly with the Tactical Terrain Data Base to obtain necessary terrain information for the location of a supply points within a US Army Division area of operations.

2.5. Location-Allocation Methodologies

A review of the literature reveals a substantial amount of reported research on the location-allocation problem. The location-allocation problem was originally defined and examined by Cooper [9]. An abundance of research exists on the fixed capacity location-allocation problem [14, 17, 18, 19, 20, 21, 35]. More recent research by Rardin and Geoffrion [35, 20] has revealed exact algorithms and computational methods for the capacitated facility problem.

Current methods of obtaining exact solutions to the location-allocation class of problems rely heavily on integer programming methods and a sophisticated computer environment. Significant advances in the efficient solution of these problems have been offered by Rardin [35, 36] and Geoffrion [20]. Anderson summarizes the current state of computer use for obtaining optimal solutions to the location-allocation problem:

> All of these presently accepted methods of obtaining global optimal answers to the location-allocation class of problems with capacity constraints utilize heavily the techniques of integer programming. The codes developed for their implementation require large, high speed computers to produce satisfactory results (quote from [1]).

Computers have become more powerful and more efficient, but the complex and repetitive nature of exact solutions to location-allocation problems has led researchers to search for more efficient and less complex heuristic solutions.

The earliest heuristic approach to solving the plant location problem was done by Kuehn and Hamburger [25]. Approximate algorithms were later developed by Manne [28], and Feldmen, Lehrer, and Ray [15]. These heuristics were developed primarily as the result of a need to reduce the amount of computer time required to obtain reasonable solutions. An approach that has been used by the successful heuristics is to decompose the location-allocation problem into a series of network flow problems. The heuristics vary in the methods that they choose trial sets of open nodes, but are similar in their solution methodologies. As with any heuristic procedure, tradeoffs between exact, optimal solutions requiring large, high speed computers, and approximate solutions obtained by the heuristics much more efficiently, were necessary.

> Tradeoffs arise in this procedure (heuristics) because of the lack of a guarantee of reaching an optimal solution due to the heuristics, and because of the approximations that are made to the cost equations. The heuristics are used to assign customers to sources and to open or close sources, such that the number of alternatives evaluated is quite small. The result of applying these heuristics is that one can quickly determine a good set of sources in minimum computer time (quote from [1]).

The popular heuristic presented by Kuehn and Hamburger [25] for locating warehouses is a two-part process. A main program locates warehouses one at a time until no more warehouses can be added to the network without increasing total costs. A second "bump and shift" routine attempts to modify solutions obtained by the main program by evaluating the effects on profit of dropping individual warehouses or shifting them to a new location. The main program employs three principal heuristics. 1) Most geographical locations are not suitable; most suitable locations will be at or near concentrations of demand. 2) Near optimum warehousing systems can be developed by locating warehouses one at a time, adding at each stage of the analysis that warehouse which produces the greatest cost saving for the entire system. 3) Only a small subset of all possible warehouse locations need be evaluated in detail at each stage of the analysis to determine the next warehouse site to be added [25]. After all sites have been either eliminated or assigned a warehouse, the program enters the "bump and shift" routine. The bump and shift routine modifies solutions reached in the main program by first, eliminating any warehouse which is no longer economical because some of the original assigned customers are now being serviced by other warehouses, and then shifting each warehouse from its current location, if necessary, to insure that each of the territories established by the main program is served from a single warehouse within the territory in the most economical manner. This heuristic (Kuehn and Hamburger) is commonly referred to as the "add" heuristic.

A second popular heuristic for solving the warehouse location is the "drop" heuristic presented by Feldman, Lehrer, and Ray [15]. Rather than add warehouses one at a time, this heuristic begins by assuming that there is a warehouse assigned to every site, and then begins "dropping" warehouses one at a time until no more warehouses can be dropped without increasing total costs.

Several other heuristic procedures for solving the warehouse location problem have been developed. Among these are the procedures of Balinski and Mills [2], Ballou [3], and Baumol and Wolf [6]. Several of these heuristics obtain a solution by reducing the location-allocation problem to a series of network flow problems and differ only in their method of deciding which set of nodes to open next. Allowing a "user" to select the next set of open nodes based on viewing a series of sensitivity reports (interactive optimization) is one technique. This was the method employed by Anderson for the Water Point Locator [1].

The difficulties identified in optimally solving a realistic supply point location for Anderson's Water Point Locator [1] led him to search for an efficient and effective heuristic procedure using interactive computer graphics for the solution of the water point location problem. Anderson found that this type of solution was necessary because of the mathematical complexity of obtaining exact solutions due to the NP-complete nature of this class of problems. He also found that the use of interactive graphics allows the user to include his own experience and pattern recognition abilities in the solution process. This interactive process "eliminates many of the possible, but not optimal solutions from the set of those to be enumerated by the computer" [1].

Anderson found that there are generally three solution procedures for solving the network flow problems required in obtaining a solution to the location-allocation problem. These methods include primal network flow, the out-of-kilter algorithm, and the 2-stage approach using the shortest route problem combined with solving the transportation problem. The primal network solution and the out-of-kilter algorithm allowed a single step solution procedure, but made sensitivity analysis more difficult, i.e. the entire problem must be re-solved whenever changes are made to supply or demand. Anderson chose to use the transportation algorithm because of its relative simplicity and its ability to facilitate post-optimality analysis. This approach is generally considered to be a variation of those published by Kuehn and Hamburger [25], and Feldman, Lehrer, and Ray [15].

In their research on a DSS for the location of ammunition transfer points, Callahan and Link [34] used a similar approach. As with the water point locator, a shortest route algorithm was first used to determine the shortest distances from candidate locations to supported units to obtain distances used in the cost matrix. A series of 1-median calculations was then accomplished using the weighted supported unit locations to determine a median location, which was then used to rank order the candidate ATP locations for further consideration.

The transportation algorithm used by Callahan and Link [34], like Anderson's [1], required a two step solution process. An all possible shortest route problem was first solved for each of the candidate locations in the initial candidate set. The shortest distance values were then used to form a cost matrix which was used by the transportation algorithm to obtain a solution. This method was very efficient and very conducive to effective sensitivity analysis in that after an initial solution had been obtained, changes could be made and analyzed without re-solving the entire problem. Only the reduced transportation problem need be solved during subsequent iterations.

The complexity and size of the location-allocation class of problems has most recently led researchers to seek more efficient solutions. Exact or optimal solutions to these problems are often possible. However, even with the assistance of powerful computers, these solutions remain inefficient. Heuristic solutions which eliminate many of the non-optimal solutions early in the procedure have proven most efficient, and while no guarantee of optimality exists when using these heuristics, very good solutions have been obtained, as demonstrated by Anderson [1], Callahan and Link [34], Kuehn and Hamburger [25], Manne [28], and Feldman, et. al [15].

The literature reveals very little research in the area of vulnerability as a criterion in network optimization. Preston [30] investigated the problem of determining "optimum allocation of aircraft to an airstrike against a transportation network." Preston's objective function included a vulnerability parameter assigned to each arc in the transportation network. The values of the vulnerability parameters used in the analysis were arbitrarily assigned (apparently based on intelligence data) and remained constant for each arc/link in the network. Preston used an exponential damage function and a dynamic programming solution procedure which resulted in integer solutions.

CHAPTER III

INTERACTIVE OPTIMIZATION APPROACH TO SUPPLY POINT LOCATION

3.1 The Optimization Problem

This chapter describes proposed data, models, and methodology for interactive solution of the Supply Point Location problem as a location-allocation problem. The discussion includes some data and procedures that are emulated or bypassed in the implementation of the experimental SPL software. See Chapter IV for the program design.

3.1.a Suitability Constraints.

3.1.a.1 Node Suitability - Potential Sources.

Definition of the network is the first major function of the SPL. A "suitable" set of nodes/sites and links must be determined in order to define the network. Initially every node in the roadway network data, including demand nodes, is a potential source node or supply point location. If the user does not provide node suitability data, a site suitability routine queries the roadway network and the terrain data bases to eliminate all nodes that do not represent a viable area having the minimal terrain requirements to support a supply point. These minimum requirements for supporting a supply point reflect current doctrine. If there is not a sufficient area of traversable terrain around the node's defining location, and if the node does not possess suitable connectivity to other nodes, then the node is not one that can support a supply point and will be eliminated.

Define an area A (m^2) as the area required for a suitable site, and a dimension D (meters) so that the site suitability routine's terrain analysis subroutine will search to try to find at least A m^2 of suitable terrain within a D x D meter square centered on the node's defining location. A and D are both non-volatile parameters that are the same for all nodes.

When the site suitability routine is invoked, the D x D square is formed and a search is made through the points in a 10 x 10 meter grid of the terrain data base. For each grid intersection, the data base contains an "elevation" word, a "soil-type" word, a "vegetation" word, and a "structures' word. Other non-volatile parameters used by this routine are:

- maximum suitable slope (of terrain)
- a vector of suitable soil types
- a vector of suitable vegetation types
- a vector of suitable structures types

If at least A/100 of the points are determined to be suitable, then the site is suitable from a terrain analysis viewpoint. To be suitable, a point must have a small enough elevation difference from its surrounding points, a suitable soil type, a suitable vegetation type, and a suitable structures type. A site selected by the suitability routine as suitable contains all of the minimal requirements to support the logistics tasks performed at a support area or supply point.

An "access" subroutine of the site suitability routine will first check to see if the node has at least three links. If the node has only two or fewer links it will be eliminated as a potential supply point. If the node has three or more links its terrain analysis will be run and if it does not meet the minimum criterion of A/100 suitable points in its D x D hinterland it will be eliminated by the suitability routine as a potential supply point. The user may override or bybass site suitability determination.

3.1.a.2 Link Suitability - Reduced Network. The road network data base is assumed to contain link attributes that include endpoints and centerline distances in addition to enough detail to allow determination of trafficability by various types of high density wheel vehicles (e.g. 2 1/2 Ton and 5 Ton cargo trucks). "Trafficability" depends on road conditions which are a function of road characteristics, climate, weather, and "combat conditions," as well as any "arbitrary unsuitability" imposed by the user.

"Combat conditions" include temporary obstructions such as barriers, mines, craters, and temporary improvements, as well as any permanent construction or destruction recent enough not to be included in the roadway data. Combat conditions will be managed by requiring the user to take one of several actions which could include:

1. Delete the link from the DSS running copy of the roadway network data base.
2. Set the travel time to a very large value in the running data.
3. Update the DSS copy of the roadway network data base to set travel time large.

The user would normally take Action 1 (link deletion) if the road will be out of service for the duration of the operation, or if there is an "arbitrary unsuitability" expected to be permanent such as might be specified by a Host Nation Agreement or effective STANAG (Standardized NATO Agreement), and there is a negligible likelihood of having to restore the road (link) to the DSS copy of the data. The second action (travel time) is the easiest way to reflect weather or easily reversible damage, where the data are expected to be changed only temporarily. Action 3 (roadway data revision) is the best for relatively permanent changes, but it requires skills that the user may not possess.

Given the roadway data base and any manual revisions, the suitability of a link will be entirely determined by comparing the road type code (or equivalent combination of link attributes such as surface, shoulder, maximum slope, curvature, etc.) with a list of suitable road types.

When the link suitability routine is executed, the output is a reduced network, which excludes unsuitable links and all nodes whose links are all unsuitable. Note that a node that was eliminated as a potential supply point is not removed from the reduced network.

3.1.b Minimization of Lag and Loss

3.1.b.1 Lag and Loss. U.S. Army doctrine requires commanders and CSS planners to perform some type of risk-benefit analysis when locating support areas as described in Chapter I. This analysis requires a tradeoff between responsiveness to supported units and local security of the support area. In order for the DSS to consider the relative importance of each (responsiveness and security), parameters describing the relative values must be included in the objective function for consideration during optimization. In order to quantify these principles, the concepts of "commodity lag" and "commodity loss" are introduced.

Commodity lag describes the responsiveness of the system while commodity loss describes the vulnerability (or security) of the system. If parameters are assigned to describe the relative "value" of responsiveness and security (subjective) then with a given solution, they provide a way combining the estimated commodity loss in tons per day and a resulting commodity lag in ton-days per day. This allows for an overall performance measure of the system that balances responsiveness against expected losses. The goal of the DSS is to minimize the weighted average of commodity lag and commodity loss.

Key commodities are an entire class of supply, or items within a class of supply that the commander determines to be "mission essential." Examples of key commodities may include major end items such as tanks, howitzers, machine guns, etc. (pacing items), repair parts (CLASS IX), certain POL products such as diesel fuel, mogas, etc., or particular types of high consumption ammunition (CLASS V).

Lag for a key commodity is a surrogate for performance. For example, shorter supply routes will increase the rate of resupply while also saving time (responsiveness) and saving fuel. Travel time is a surrogate for reduction of delays to supported units and also for fuel costs. Loss for a key commodity is also a surrogate for cost of supplies in that commodities required by supported units destroyed by enemy action must be replaced; it is also a surrogate for loss of assets and personnel traveling with the commodities.

Weights for lag (w, for "wait") and loss (v, for "vulnerability") are specified by the user based upon the tactical situation and the commander's guidance and will subsequently be referred to as "risk guidance." There are essentially two approaches to specifying values for lag and loss. The first method is the use of "relative values." Using relative values for risk guidance is the simplest method and enables the user to specify a single value for risk guidance by normalizing the two parameters. An assumed relationship between w and v used by the SPL is

$$w = 1 - v.$$

The user must select a value for lag (w), and loss (v) will be automatically computed. w is limited to a range of values determined experimentally (see user documentation, sec 4.3). A small value of w corresponds to a greater importance of responsiveness and thus penalizes v (loss) and should tend to locate the supply point(s) farther forward. Likewise, a large value of w penalizes responsiveness and should locate the supply point(s) further back from the FLOT.

The second approach to specifying values for lag and loss is in terms of "absolute" weights. This approach represents an alternative method of using risk guidance in the solution procedure. Using this approach the user may estimate a value in "dollars," or any unit of value, for a ton of commodity loss and the losses associated with that loss such as accompanying personnel and materiel. The user also estimates the value of a lag (delay) of one day of a ton of commodity and the associated lag costs such as the cost to mission accomplishment caused by the delay in a similar manner.

Absolute values need not be between 0 and 1. For example, if the user estimates the value of one ton of loss to be \$300 and the value of one ton-day of lag to the mission as \$20, it would be the same as using $w = 300/320$ (lag) and $v = 20/320$ (loss) as in the relative approach. However, if the user has assigned "meaningful" values or magnitudes using the absolute approach, the values obtained as output from the solution will be meaningful rather than merely relative values which serve as multipliers in the objective function. Values between 0 and 1 may also be used with this method but need not sum to 1. For instance if the user feels the loss is "ten times more important" than lag, he may use 1 for loss and 0.1 for lag.

A special case of the "absolute" method of lag-loss weight specification is to make the objective function have units of equivalent tons per day of loss. Set v = 1. Then determine an appropriate lag weight w by applying the following consideration: if, say, 50 ton-days of lag (such as 50 tons traveling on roads for one full 24-hour day or 500 tons traveling for one tenth of a day) interferes with the mission approximately as much as losing one ton, then set w = 1/50 = 0.02.

As will be seen below, if true losses are proportional to estimated commodity losses and true performance is proportional to the negative of estimated lag, the network flow model solved by the DSS can optimize a weighted sum of losses and performance.

3.1.b.2 Node Vulnerability. Vulnerability is what determines losses on nodes. It is measured in units of expected proportional loss of the key commodity per day of exposure. For example, if the estimated vulnerability parameter (defined below) for node i is $p_i = 0.05$, this is equivalent to expecting 5% of the stored inventory of key commodity to be lost per day of exposure, or, approximately, to expecting 5% chance of total inventory loss per day of exposure. When p_i is multiplied by the average inventory the result is expected loss in tons per day.

Node vulnerability could presumably be modeled using node attributes such as distance from enemy locations, enemy capabilities, concealability, defensibility, line of sight, size of friendly security/reaction force, friendly electronic signatures, etc. The SPL could use data-based vulnerability estimation employing the terrain and roadway databases and a routine for comparing these attributes with acceptable values similar to the procedure used by the node suitability routine. If the vulnerability constant, no matter how estimated, exceeds an acceptable maximum value the node will be rejected as a potential supply point although it will remain as part of the network; otherwise the value helps determine commodity loss as explained below.

No one has yet defined a comprehensive method for such data-based vulnerability estimation. Simpler approaches include user entry of vulnerability estimates, or estimates based on distance from the forward line of troops (FLOT).

Even though a comprehensive method for data-based vulnerability estimation has not been defined, some method must be available to the SPL for estimating default values for vulnerability. A heuristic method for estimating default values for vulnerabilities based on general ranges of direct and indirect fire weapon systems and distance of the node from the FLOT is presented.

By assuming a straight line for the FLOT and given that x and y coordinates for all nodes are known, perpendicular distance from any node or the midpoint of any link may be computed using the normal form of a straight line and computing directed distance from the node or link to the line. If $Ax + By + C = 0$ represents the equation of a line, then this form may be reduced to the normal form by dividing each term by

$$\sqrt{A^2 + B^2}$$

and choosing the proper sign of the radical, i.e. if $C <> 0$, then the correct sign of the radical is opposite to that of C. The directed distance is then computed by substituting the coordinates of the point for x and y in the left member of the normal form of the equation of the line. Based on general ranges of direct fire, indirect fire, and tactical air support, and the straight line distance from the FLOT, an empirical distribution representing the probability of attack by enemy weapons systems has been assumed. The negative exponential closely approximates this distribution. The density function for this distribution may be written as

$$f(d) = e^{-gd}$$

where g is a single parameter and d is distance from the FLOT. Figure 3.1 illustrates the distribution function and shows the approximate range of the different types of weapons systems. The parameter (g) used by the vulnerability computation routine is assumed to be 0.06. The distribution function shown in Figure 3.1 uses 0.06 as a value for the scale parameter (g).

A vulnerability computation routine uses the distance previously computed from the node to the FLOT in the empirical formula for the negative exponential (g=0.06) and stores the computed vulnerability constants p_i and q_{ij} as vectors of node and link vulnerabilities.

The assumption of a straight line for the FLOT is not unreasonable in that the FLOT doctrinally follows an identifiable terrain feature such as a road, river, ridge line, etc, which generally can easily be simulated as straight lines. The outside boundaries of the Division or Brigade sector or zone along the FLOT are the endpoints for the line.

<u>3.1.b.3 Link Vulnerability</u>. Link vulnerability is what determines losses on links in the network and can be modeled using similar characteristics to those used in node vulnerability estimation. A link or a road, however, will seldom contain high values of attributes such as cover and concealment. Tactical priorities will determine how much

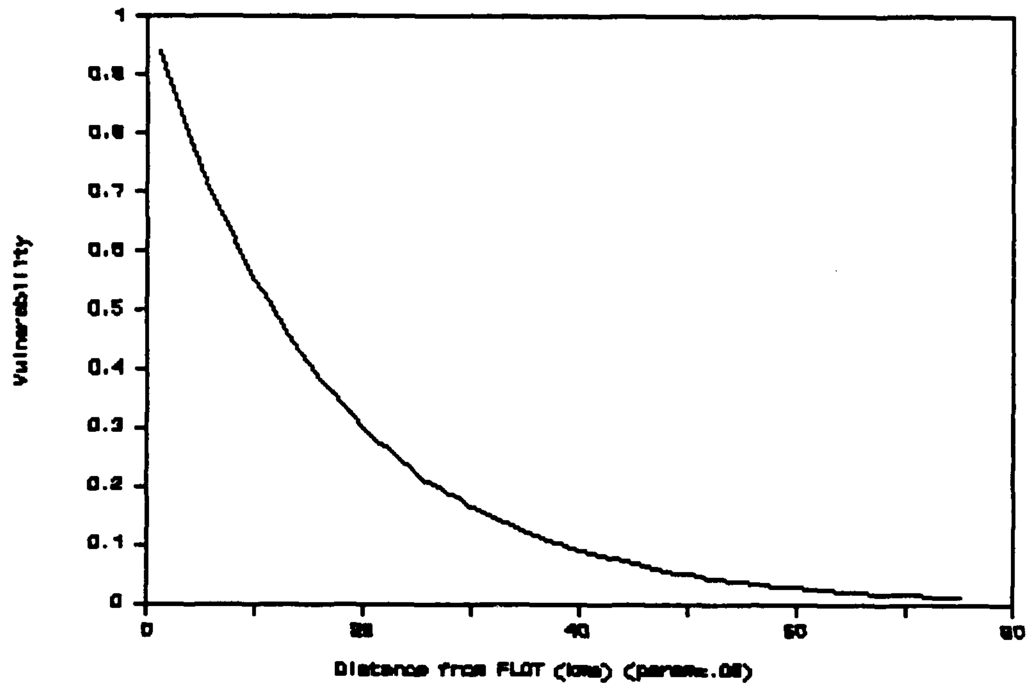

Figure 3.1: Vulnerability Gradient

friendly effort is expended in maintaining the security of a particular road. Doctrine normally affords high priority to the protection of supply routes (MSRs) whenever possible. Link vulnerability, however, remains difficult to quantify. With respect to most of the link vulnerability attributes it would be reasonable to assume an average of the two nodes that the link connects. For example, with respect to enemy capabilities, since the enemy's capabilities are mostly dependent on his ability to engage targets, it would be reasonable to measure this attribute by weighting the link as the arithmetic average of the two nodes that it connects. As with node vulnerability, if this value, or link vulnerability constant, exceeds an acceptable maximum value the link will be penalized with a very high vulnerability.

As with node vulnerability, simpler approaches to estimation include user entry of estimates, or estimates based on distance from the FLOT. Default link vulnerability constants may be computed as the arithmetic average of the two nodes at each end of the link.

The units of the link vulnerability parameter q_{ij} (defined below) are proportion of key commodity expected to be lost per day of travel on the link. When q_{ij} is multiplied by travel time and by shipment rate, the result is tons of expected loss.

3.1.b.4 Objective Function

Consider the reduced transportation network consisting of supply nodes, demand nodes, and links between them, where link [i,j] from supply node i to demand node j is a path of links in the roadway network. Figure 3.1.1 provides a small example of the relationship between the reduced network and the roadway network. Let q_{mn} be the vulnerability constant for link (m,n) in the roadway network. Let d_{mn} be the travel speed for the link, so that $q_{mn}d_{mn}/s_{mn}$ is the expected commodity loss per ton of commodity that flows along the link. Let the link be part of a path [i,j] from supply node i to demand node j; that path is a link in the reduced network. For a day of operation the commodity flow is x_{ij} tons along each link constituting the path, so the expected loss in tons is

$$Q_{ij}x_{ij} = \sum_{(m,n)\epsilon[i,j]} q_{mn}(d_{mn}/s_{mn})x_{ij}$$

where Q_{ij} is

$$Q_{ij} = \sum_{(m,n)\epsilon[i,j]} q_{mn}(d_{mn}/s_{mn})$$

In the transportation problem defined on the reduced network, the total "cost" for link vulnerability is

$$v \sum_{[i,j]} Q_{ij}x_{ij} = v \sum_{i} \sum_{j} Q_{ij}x_{ij}$$

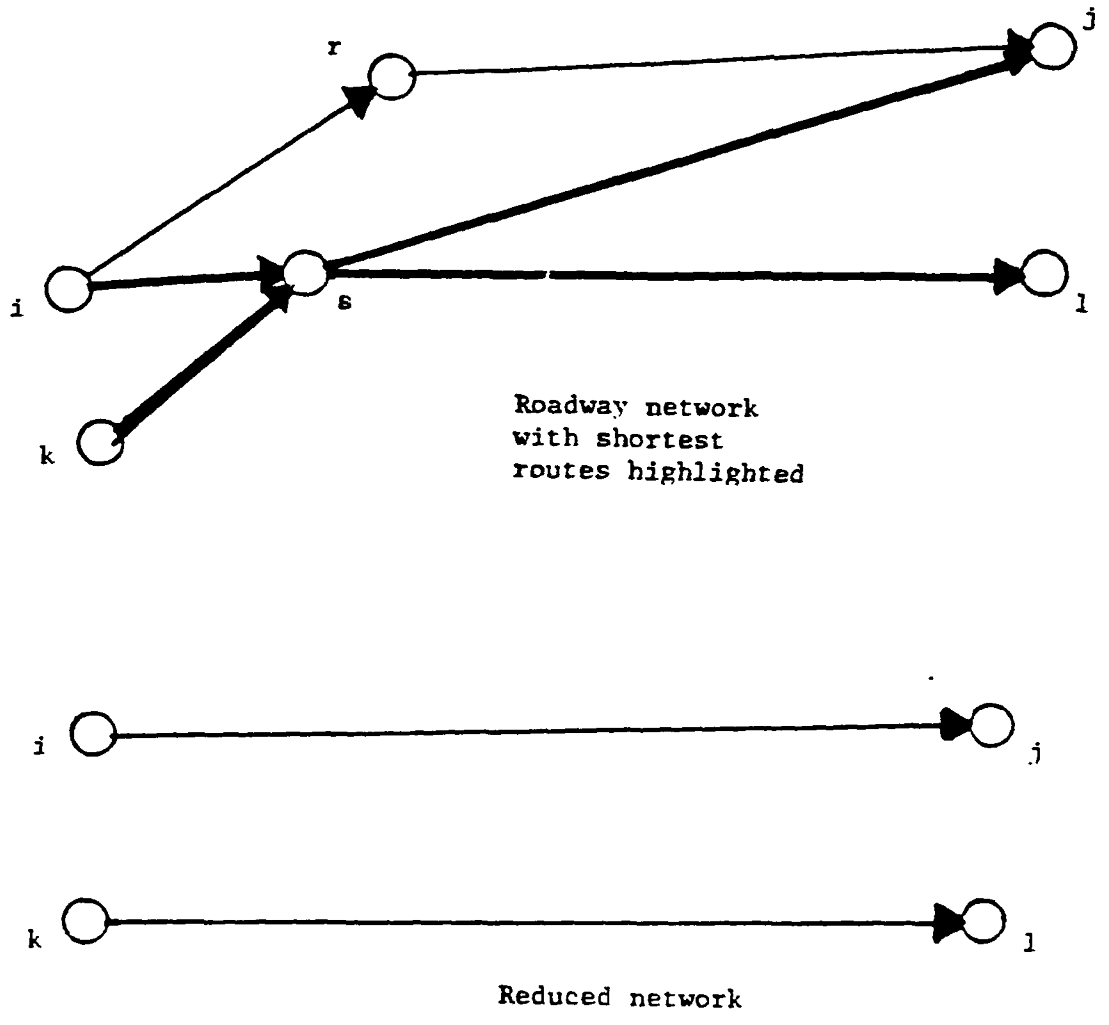

Figure 3.1.1: Reduced Network and Roadway Network

where v is the loss value as defined in Section 3.1.b.1. If $v = 1$ the sum is the expected tons of loss of traveling commodity in a day of operation.

While a commodity is being held at supply point i prior to issue to a supported unit, let its "vulnerability" be indicated by a quantity p_i, the expected proportion to be lost per day. Let the average inventory of the commodity at supply point i be $a_i \sum_j x_{ij}$ where a_i is the tones of inventory per ton-per-day shipped out of the supply point.

In the transportation problem, the total "cost" for node vulnerability is thus

$$v \sum_i p_i a_i \sum_j x_{ij} = v \sum_i \sum_j p_i a_i x_{ij}$$

Combining the terms for loss on links and at nodes, we have

$$v \sum_i \sum_j Q_{ij} x_{ij} + v \sum_i \sum_j p_i a_i x_{ij} = \sum_i \sum_j (v(Q_{ij} + p_i a_i)) x_{ij}$$

Note that because exposure to loss at a node is proportional to flow out of the node due to the inventory's being expressed as a "number of days of supply," the total losses are expressible as a sum of transportation-type cost expressions.

Lags, of course are also transportation-type quantities. The total lag or travel time in link (i,j) of the reduced transportation network is the sum of the travel times along the roadway-network links that comprise the path. Where w is the lag value as defined in Section 3.1.b.1, the "cost" for lag along a link is

$$w \sum_{[i,j]} G_{ij}x_{ij} = w \sum_i \sum_j G_{ij}x_{ij}$$

where

$$G_{ij} = \sum_{(m,n)\in[i,j]} d_{mn}/s_{mn}$$

A ton of commodity also spends a_i days at its supply point, but although this delay is influenced by the decision variables x_{ij} it will be ignored. (Without compromising the mathematical transportation structure, this delay could either be added to lag or given its own "value" in the objective function.) Note that if a_i is the same for all i this delay would be independent of $\{x_{ij}\}$; there appears to be no doctrinal reason for a_i to vary, or for the differences in inventory levels to influence the decision (other than through loss, which is already accounted).

The transportation objective function to be minimized is the sum of loss terms and lag terms:

$$\sum_i \sum_j (v(Q_{ij} + p_i a_i))x_{ij} + w \sum_i \sum_j G_{ij}x_{ij}$$

$$= \sum_i \sum_j [v(Q_{ij} + p_i a_i) + w\, G_{ij}]x_{ij}$$

$$= \sum_i \sum_j c_{ij}x_{ij}$$

where c_{ij} is defined as the quantity in brackets.

The basis for computing Q_{ij} and G_{ij} is evident from their definitions. First, note that

$$c_{ij} = v(Q_{ij} + p_i a_i) + w\, G_{ij}$$

is independent of x_{ij} and must be minimized for the transportation objective to be minimized. Next, note that c_{ij} can be minimized by minimizing the quantity

$$vQ_{ij} + wG_{ij} = \sum_{(m,n)\epsilon[i,j]} (vq_{mn} + w)d_{mn}/s_{mn}$$

over all sets of links that comprise path (i,j). (Here $p_i a_i$ was omitted since it is constant.)

But this minimization is exactly that of the standard shortest-path problem with link costs as given in the above sum.

Hence the optimal solution to the network flow problem is given by first solving the defined shortest-path problem to obtain c_{ij} for each path from a supply node to a demand node, and then solving the defined transportation problem.

3.2 Network Display - Semiscaled Networks

When a road network is displayed to scale, nodes are represented by small figures (such as circles or triangles) centered on scaled locations, and roads are represented by line segments whose ends are at the scaled locations. Figure 3.2.1 gives an example: There are five locations and four roads, represented by five circles and four line segments. Although this display straightens the roads, it preserves relative locations of nodes and thus preserves some degree of recognizability, allowing the user to maintain awareness of which displayed entity corresponds to which real entity. This awareness is desirable in interactive work with a network display.

However, ability to view large segments of a network, without the necessity for panning and zooming, is also desirable in interactive work, and it is desirable to have a sufficient hinterland around each node to allow display of numerical and text information around each node without interfering with display of other nodes.

With limited-resolution graphics, a scaled-location display can interfere with these desiderata: the rigid scaled locations limit the number of displayable nodes and limit the node hinterland sizes. Where nodes are too far apart, display space is wasted; where nodes are too close together, hinterlands are too small.

The semiscaled network allows some adjustment among the desiderata of location integrity, number of displayed nodes and hinterland size. A semiscaled network distorts the locations of nodes to spread them more evenly over the screen and provide bigger hinterlands. The amount of distortion or spreading is controlled by a <u>spreading parameter</u>, ALPHA, that varies from 0 for complete preservation of scale to 1 for maximum spread, and by a <u>ranking parameter</u>, BETA, that varies from 0 for complete preservation of relative horizontal and vertical node order to 1 for no preservation.

We define an unscaled display in which the window is partitioned into n x m rectangular cells with each node at the center of a cell. The ranking parameter BETA is the minimal proportional x or y separation for which the unscaled display will separate two node's x or y cell indices. The unscaled network will have a different row (column) of cells for every node whose y (x) coordinate differs from that of another node by at least BETA. Figure 3.2.1 shows a scaled network; Figure 3.2.3 shows the corresponding unscaled network for BETA = 0.04, which has nodes 4 and 5 in the same column because their x coordinates differ by 3/250 = 1.2% < BETA, but has nodes 2 and 3 in different columns because their x coordinates differ by 10/250 = 4% = BETA. The corresponding unscaled network for high BETA is shown in Figure 3.2.5.

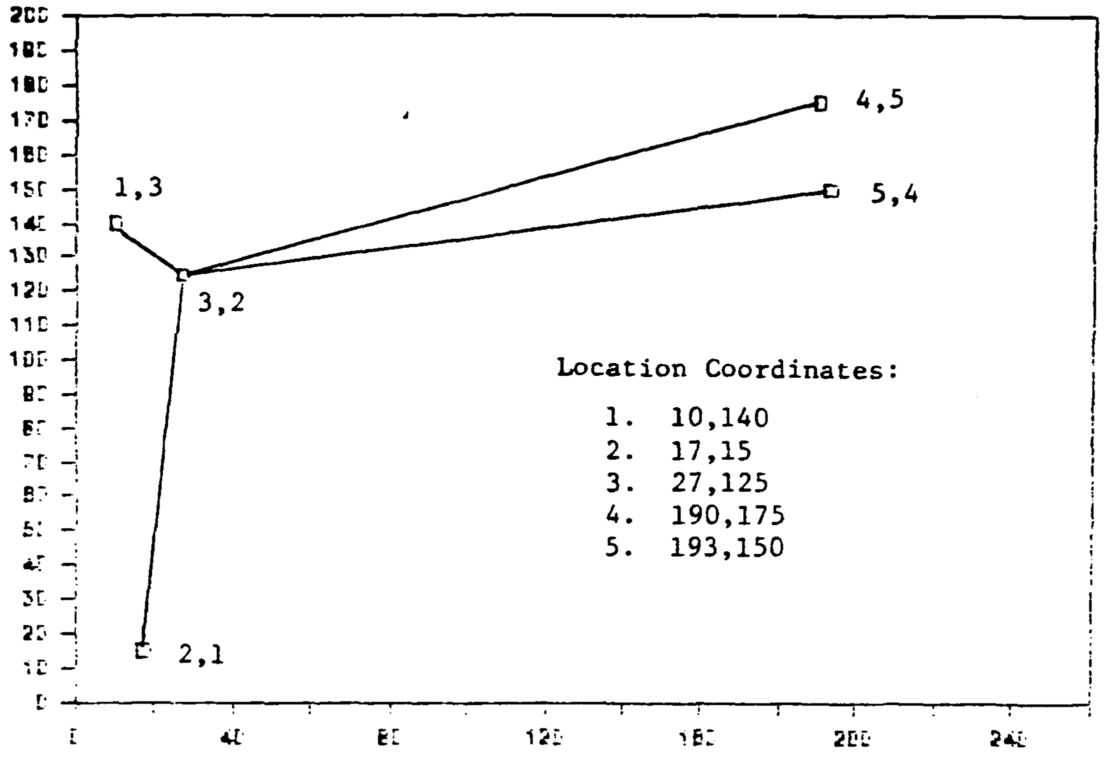

Figure 3.2.1: A Scaled Network
(ALPHA = 0, any BETA)

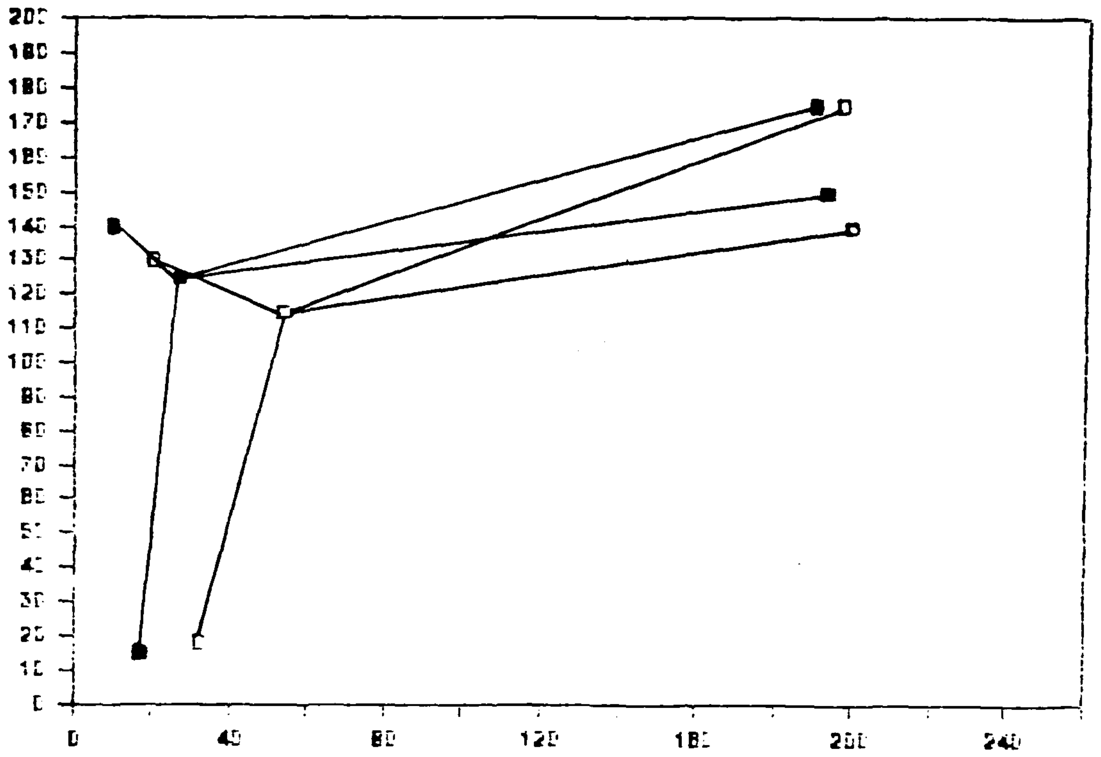

Figure 3.2.2: A Semiscaled Network
(ALPHA = 0.2, BETA = 0.04)

The first step in semiscaling is to define the unscaled network, which may be done by any set of rules that uniquely assigns nodes to rectangular cells.

The final step is to define the semiscaled network. This is done by placing each node a proportion ALPHA of the distance from its scaled to unscaled location. If x_s is the scaled (true) x-coordinate of a node, and x_u is its true unscaled x-coordinate, then its semiscaled x-coordinate, x, is computed as $x = x_s + ALPHA(x_u - x_s)$; similarly , $y = y_s + ALPHA(y_u - y_s)$.

Figure 3.2.2 gives a representative example, with BETA= 0.04 and ALPHA = 0.20. Note that the network retains the general appearance of the truly-scaled network (Figure 3.2.1), but gives enhanced separation of nodes, thus allowing larger hinterlands for unambiguous display of node-specific information surrounding each node.
Table 1 shows the numerical results for semi-scaling that converts Figure 3.2.1 to Figure 3.2.2. Figure 3.2.6, with BETA = 0.8 and ALPHA = 0.5, is more distorted but allows larger hinterlands. Figures 3.2.4 and 3.2.5 are extremely distorted but allow very large hinterlands. With sophisticated rules for semiscaling it is possible for an N-node network to be displayed (as in Figure 3.2.5) in a number of cells not much greater than N, each with a

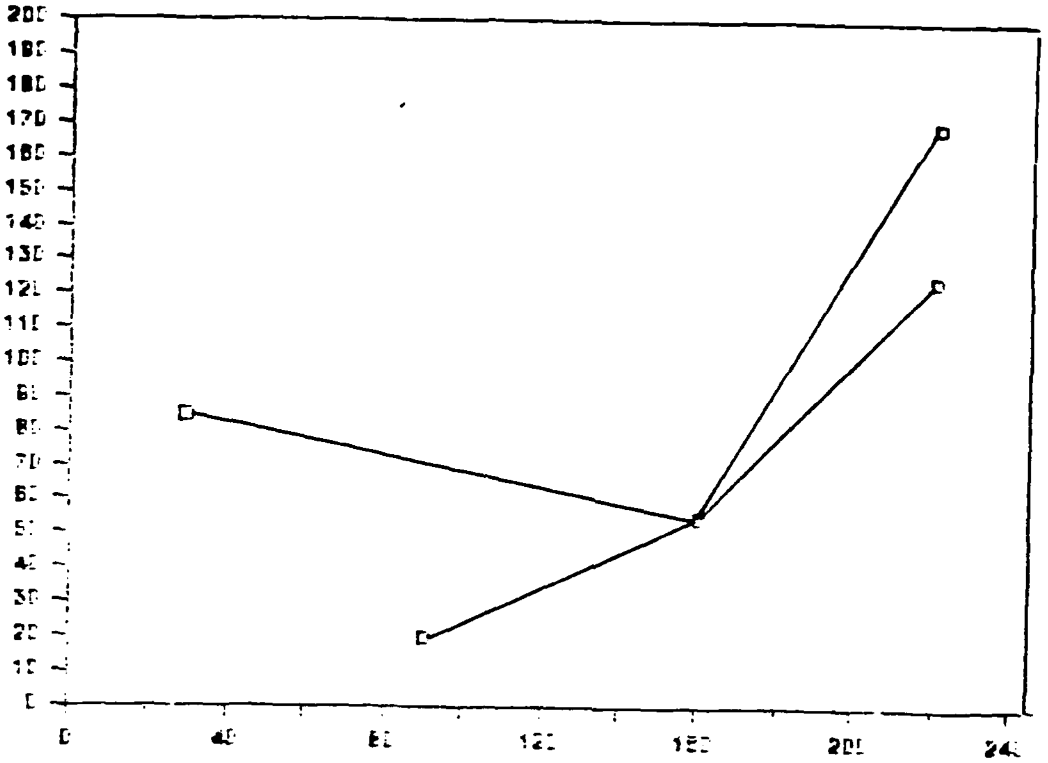
Figure 3.2.3: An Unscaled Network
(ALPHA = 1.0, BETA = 0.04)

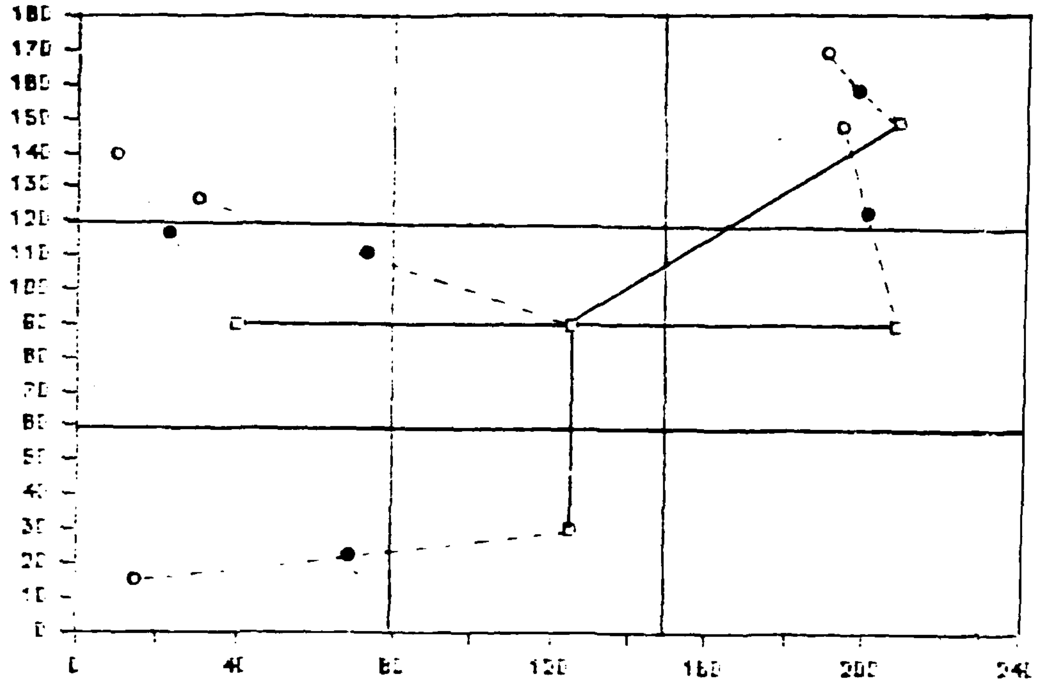
Figure 3.2.4: An Unscaled Network
(ALPHA = 1.0, BETA = 0.8)

Table 1: Semiscaling Example - BETA=0.04, ALPHA=0.20

Node	Scaled Coordinates	Ranks	B=0.04 Cells	Unscaled Coordinates	ALPHA=0.20 Semiscaled Coordinates
1	10,140	1,3	1,3	31.25,90	14.25,130
2	17,15	2,1	1,2	93.75,18	32.25,15.6
3	27,125	3,2	3,2	156.25,54	52.85,110.8
4	190,170	4,5	4,5	218.75,162	195.75,168.4
5	193,150	5,4	4,4	218.74,126	198.15,145.2

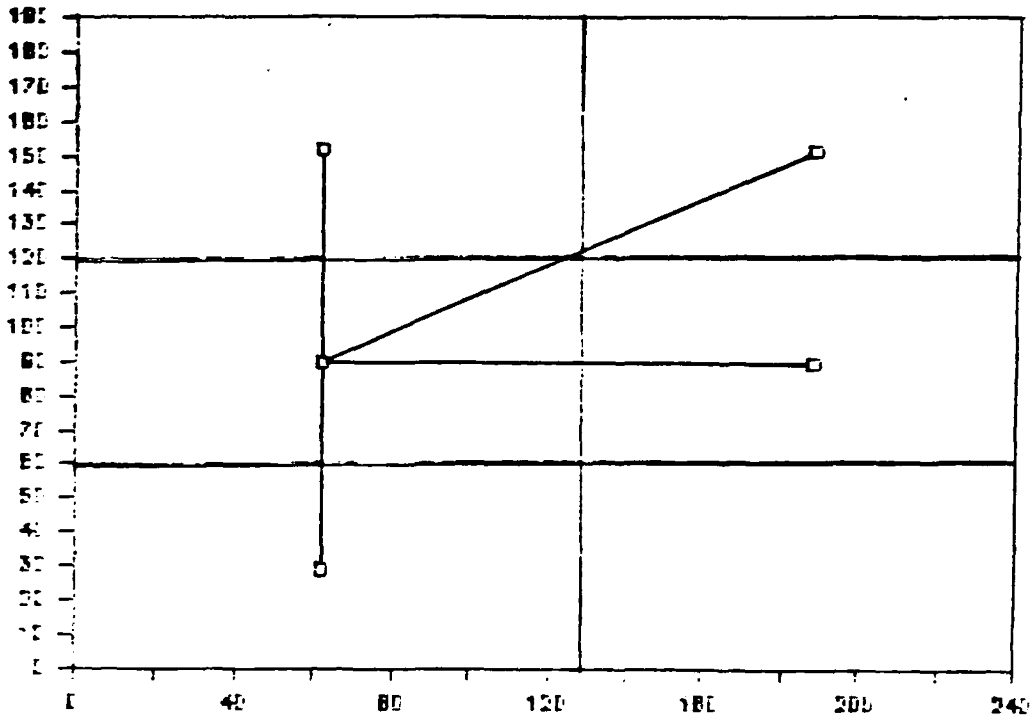

Figure 3.2.5: An Unscaled Network
(ALPHA = 1.0, BETA = 1.0)

hinterland whose proportional area approaches 1/N.

At large BETA, there are many ties for cell membership in formation of the unscaled network. Let Q different nodes be in the same cell by the tentative assignment of rows and columns; these nodes all differ in both x and y coordinates by less than BETA. Let there be at least Q available cells surrounding the crowded one. The assignment algorithm can be used to assign each of the Q nodes to a unique cell. Each assignment cost is the distance (either rectilinear or Euclidean) from the true coordinate of the node to the center of a cell.

For this prototype the assignment algorithm is not used. Instead, semiscaling is done only when BETA is small enough to provide a unique cell for each node. Thus the prototype will demonstrate the ability of semiscaling to provide enhanced separation in the display, but not to the maximum extent possible.

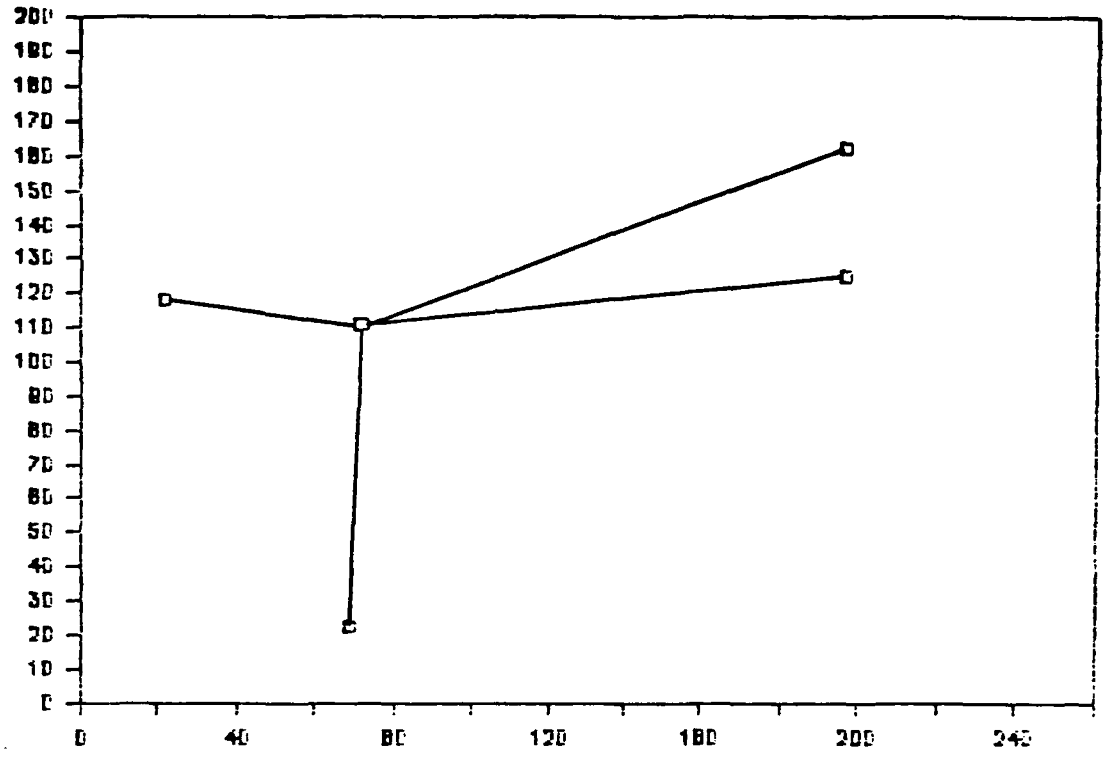

Figure 3.2.6: A Semiscaled Network
(ALPHA = 0.5, BETA = 0.8)

3.3 Interactive Optimization

As a tool to assist Army commanders and CSS planners in the field, the SPL uses an interactive optimization technique for the solution of the supply point location problem. Use of the SPL requires little or no operations research experience or prior knowledge of network optimization. Interactive use of the system, however, is a key element in the selection of a supply point location. The interactive optimization technique employed by the SPL combines the speed and accuracy of automation with the experience, skill, and knowledge of the user.

The SPL requires two broad categories of data. These two data types will be referred to as "problem data" and "plan data." Both data types may be entered by the user or obtained through database extraction from a parent headquarters tactical database. Problem data is a very broad category and is defined as all of the data required to describe the network and the location-allocation problem and includes supply data (availability), demand data (from supported units), link and node costs, and the set of potential source nodes. There are two levels of problem data. The first level is "very non-volatile data" and consists of all terrain and road data including all attributes of nodes, links, and terrain, for example, grid coordinates, altitude, attitude, centerline distances, etc. The second level of problem data is "problem definition" data and includes operational information such as supply availability, demands, locations of demand sites (supported units), intelligence data, vulnerability criteria, and lag and loss objective weights (w and v). The second level of problem data is much more volatile than the first but will normally remain relatively constant for the duration of a given operation (i.e. throughout any single solution of the supply point location problem).

The second major category of data used by the SPL is "plan data" and consists only of decision variables and their consequences: a given set of open nodes to be used in the current solution of the supply point location problem, and the resulting routes, flows and objective values (In Decision Support Systems generally, "problem data" defines a problem and "plan" data defines alternative solutions to the problem. The set of open nodes may be designated by the user in the initial solution or may be designated by the user or the computer during subsequent analysis or while conducting sensitivity analysis. Plan data is very volatile and may change numerous times throughout a supply point location session. Each set of open nodes is a trial solution.

The solution structure for the interactive optimization solution technique is illustrated in Figure 3.3. Note that the structure includes three major phases; a problem definition phase, solution phase, and a decision phase. Each phase represents a critical element of the interactive optimization.

The initial phase of the interactive optimization provides the SPL with all data required to define the problem. The solution phase represents the two-phase solution procedure employed by the SPL to solve the location-allocation problem. The statement of the problem is "to select a set of open nodes so as to minimize the weighted lag and loss, subject to satisfying all demands and obeying all supply restrictions." The location-allocation problem (as defined in Chapter I) is solved by reducing the larger problem to a series of network flow problems. Each network flow problem is then solved by the solution of a transportation problem. Before the transportation problem can be solved for each network flow problem a set of shortest route problems is solved in order to provide cost data in terms of lags and losses to the transportation algorithm; there is one shortest-route problem for each open supply node. This technique provides a solution which is then analyzed in the Decision Phase of the solution

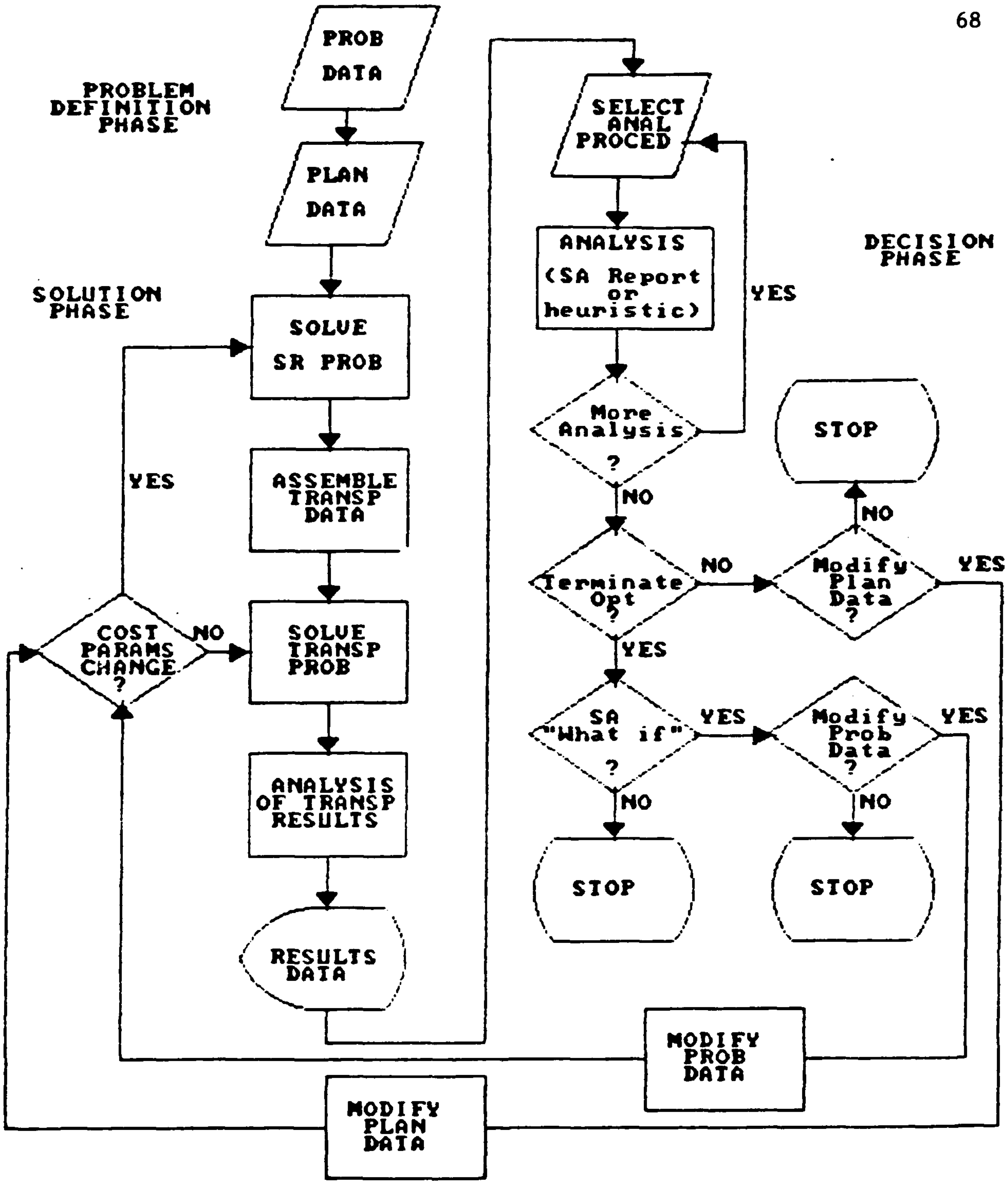

Figure 3.3
Interactive Optimization Solution Structure

structure.

During the decision phase the user analyzes the solution by viewing a series of sensitivity reports, or by requesting one or more heuristic solutions provided by the analysis function for comparison with the current solution and provides modifications as necessary to attempt to obtain a better solution.

The problem definition data requirements include all problem data as described above. Default values are available for many of these data requirements.

Plan data must be present for an initial solution and may be updated as necessary throughout the Decision Phase. The user may try as many different combinations of open nodes as desired, but the number of designated open nodes must not exceed the number of supply points to be located (r). The DSS may be used to locate a single supply facility or multiple facilities in a Division or Brigade sector/zone. The SPL has the capability to locate as many facilities as desired allowing the user to locate several supply points within a Division or Brigade area of operations each to be used for a different class of supply or group of supplies. For example, the division logistics planner may wish to locate an ammunition transfer point (ATP; CL V) and a ration breakdown point (CL I) separate from the Main Support

Battalion, i.e. closer to the support units than the MSB.

After all required data is present, solution of the shortest route problem is the next step in the solution structure for the problem. Each link in the network has its own cost because of the differences in both distances and vulnerability along the various paths. By assuming that all losses during transport are instantaneously replenished a linear relationship may be maintained. The solution of the shortest route problem yields a cost matrix representing the sum of the minimum costs of lag and loss along each path. This data is then assembled for subsequent use by the transportation algorithm. The data structure for the cost matrix is key to the proper solution of the transportation problem.

The final step in the initial solution is the transportation algorithm. Once the transportation problem is solved and results are obtained the user enters the Decision Phase of the interactive optimization. In this phase the user decides whether to accept the current plan or make modifications. To support this decision the user may select one or more of several available analysis procedures. The user may request one or more heuristic solutions for comparison or he may request any of a series of sensitivity analysis reports for further evaluation of the solution.

Heuristic solutions and analysis procedures are discussed in greater detail in the following chapter. The user may use any one or any combination of these techniques. Several different sets of plan data may be used with any one problem data set. The user may desire to modify one or more of the problem parameters or modify the network (problem data) or may wish to change the set of open nodes based on a heuristic recommendation or the sensitivity analysis. These heuristics will not necessarily provide an optimum solution but the heuristic recommendation will always be at least as good as the previous solution. When modifications are made the solution technique enters a loop (see Fig. 3.3) to obtain a new solution and provide updated analysis. This procedure may be repeated until the user is satisfied with a solution and wishes to terminate the optimization.

3.4 Computational Method

The computational method used by the SPL is very similar to that demonstrated by Anderson [1] with the Water Point Locator. This method includes a 2-stage solution approach using a shortest route algorithm combined with a transportation algorithm for solving the location-allocation problem.

Once all problem definition data has been received by the SPL and an initial set of open nodes has been designated, the shortest route problem will be solved for the reduced network in order to provide cost data (matrix) to the transportation algorithm which will subsequently be solved. All information necessary for the system to define the network is included in this data. Vulnerability parameters will be imputed from intelligence data. Demand and supply data, as well as the friendly tactical situation, are also included in this data.

Additional data that must be provided includes condition specifications for such things as convoy speed (s), number of supply points being located (r), and number of days of supply maintained at supply points (a_i). Value specifications representing the value of a ton of commodity loss (v), or vulnerability, and the value of a ton-day of commodity lag (w), or responsiveness, as defined above, will also be provided by the user.

Each path or route in the network has a unique cost combining the cost of commodity lag and loss with distance to be travelled and vulnerability on the links. The cost of each individual link on the path is expressed as

$$(w + vq_{kl})d_{kl}/s_{kl}$$

where q_{kl} is the vulnerability constant for the intermediate link (k,l) and d_{kl} represents the length or distance traveled on link (k,l). Solution of the shortest path problem will result in a cost matrix L representing the minimum loss and lag along each path. L_{ij} is defined as the link-related cost along the best path between i and j, i.e. the minimum sum of the cost of lag and loss along the path:

$$L_{ij} = vQ_{ij} + wG_{ij}$$

Once the link cost data has been computed by the shortest route algorithm, a transportation algorithm is used to complete the solution of the network flow problem by selecting the r best of the designated set of open nodes as a supply point location(s). Using the cost data provided by the shortest route solution the objective function used by the transportation algorithm now becomes

$$\sum_i \sum_j L_{ij}x_{ij} + \sum_i vp_i a_i \; j \; x_{ij}$$

from which it can be seen that

$$c_{ij} = L_{ij} + vp_i a_i.$$

After an initial solution is obtained, one or more location-allocation heuristics may be selected by the user to be executed by the SPL using the reduced network to obtain a heuristic solution(s) to the network flow problem. The user may also select from a series of sensitivity analysis reports to analyze the performance of the current solution. Consequences and performance will be displayed as a series of reports to the user.

The two stage solution of the network flow problem makes sensitivity analysis very useful and efficient. The user may make changes to the network or to any of the condition or value specifications initially provided to the SPL in an attempt to improve the solution. Modifications of the network, i.e. adding or deleting nodes or links, changes to the value specifications for lag and loss, or modification of the link vulnerability parameter (q) will require resolving the shortest route problem <u>and</u> the transportation problem in order to obtain the new solution. However, any other modifications made by the user will only require re-solving the transportation problem. The user may continue with sensitivity analysis and/or modification of the problem until satisfied that a "best" solution has been obtained.

CHAPTER IV

IMPLEMENTATION OF THE SUPPLY POINT LOCATOR PROTOTYPE

4.1 Development of Hardware and Software

4.1.a Hardware

The SPL assumes a sophisticated battlefield information environment in which data requirements for the system will be downloaded from parent unit databases to using units equipped with microcomputers and secure data communication capabilities. All hardware used in the implementation of the Supply Point Locator is current technology and should be available to Army units within five years.

The Supply Point Locator DSS employs two monitors and a printer capability as output devices and a standard IBM XT/AT or equivalent keyboard and a "mouse" or similar pointing device as input devices. All I/O will be echoed on to one of the two screens used by the system. The mouse is optional with the SPL Prototype.

A standard CGA or EGA monitor designated as the "text monitor" provides the user with menus, messages (including error messages), and a workspace for text input. A second monitor with a 1K x 1K color graphics display capability serves as a "graphics monitor." All network displays and sensitivity reports are displayed on the graphics monitor. The high resolution graphics monitor enables the user to view the each element of the network (semiscaled network), view text or numerical data associated with each of the network elements, and view sensitivity reports generated by the system. The pointer device (mouse) is used with the graphics monitor to enable the user to provide changes to the network as desired for the original designation of open nodes or for subsequent changes to the network during sensitivity analysis such as opening or closing a node or deleting a link. The mouse may also be used for selecting menu items from the text screen or designating text to be modified in the problem data.

The Supply Point Locator has been implemented using the IBM AT microcomputer with an 80286 processor, dual floppy disk drives, 20Mb hard disk and standard CGA color monitor. A minimum of 640 Kb RAM is required by the SPL Program. Additional equipment includes a Phillips 15 inch, 50kHz RGB analog monitor and a 2048 x 4 Number Nine Interrupt Operating System color graphics board. The graphics board is configured for 1024 x 768 pixels of non-interlaced display. The additional monitor is used as a graphics monitor and provides high resolution graphics display necessary for displaying large networks.

4.1.b Software

All programming has been accomplished using MICROSOFT QUICKBASIC Language, Version 4.00b [5] supplemented by a HALO Graphics library [24]. HALO is a graphics library of high performance subroutines which allows the implementation of sophisticated high resolution graphics through interface with the QUICKBASIC compiler.

4.2 Supply Point Locator Architecture

The Supply Point Locator is designed as a menu driven modular program which seeks to solve the location-allocation supply point location problem interactively, display results, and assemble reports. The architecture of the SPL employs a multi-modal design which enables the solution structure described in Section 3.3 above. The SPL uses a text monitor and a graphics monitor to provide output to the user as well as a printer capability for reports.

4.2.a Supply Point Locator Modes.

The SPL program includes five different modes as illustrated in the Functional Breakdown in Figure 4.1. Each of the modes represents one or more major functions of the program. Each major function consists of one or more functions which perform the logic for a given step in the solution structure. The five modes include:

- Network Mode
- Problem Data Mode
- Solve and Consequences Mode
- Sensitivity Mode
- Report Mode

Each of the five modes interacts with the other modes and plays a key role in the solution of the supply point location problem.

The Network Mode consists of only one major function - to provide network definition to the SPL. The key element of network definition is the roadway network data. Proper definition of the network is a critical task that must be performed by this function in order to obtain a good solution.

The Problem Data Mode serves to provide and process input data which is unique to a particular operation and which changes from mission to mission. The purpose of this mode is to specify the problem. Problem data includes CSS data, tactical terrain data, enemy and friendly situations and locations, vulnerability data, commander's risk guidance, and user input. This mode assembles problem data and defines parameters and variables required by the solution structure.

The Solve and Consequences Mode serves as the "brain" of the SPL and is the mode in which all computations are made and solutions are obtained. This mode contains the shortest route and transportation algorithms as well as the location-allocation heuristics used by the interactive optimization solution technique.

The purpose of the Sensitivity Mode is to provide the user with the ability to conduct sensitivity analysis for any given solution and provide a "what-if" capability to the SPL. Sensitivity reports are provided to the user in this mode for further analysis in the form of graphical comparisons between solutions.

The Report Mode provides no computations or analysis for the SPL but serves primarily as a utility to allow the user to view results and analysis and recommended solutions. This mode also performs some data assembly and filing tasks for reports generated and stored during the solution process.

In the "display architecture" of the SPL, each mode represents a "screen" or a series of screens requesting or providing input or providing output to the user during the interactive optimization process. Major functions in each mode are represented by a menu, menu item, or a query to the user in the message space of the text monitor.

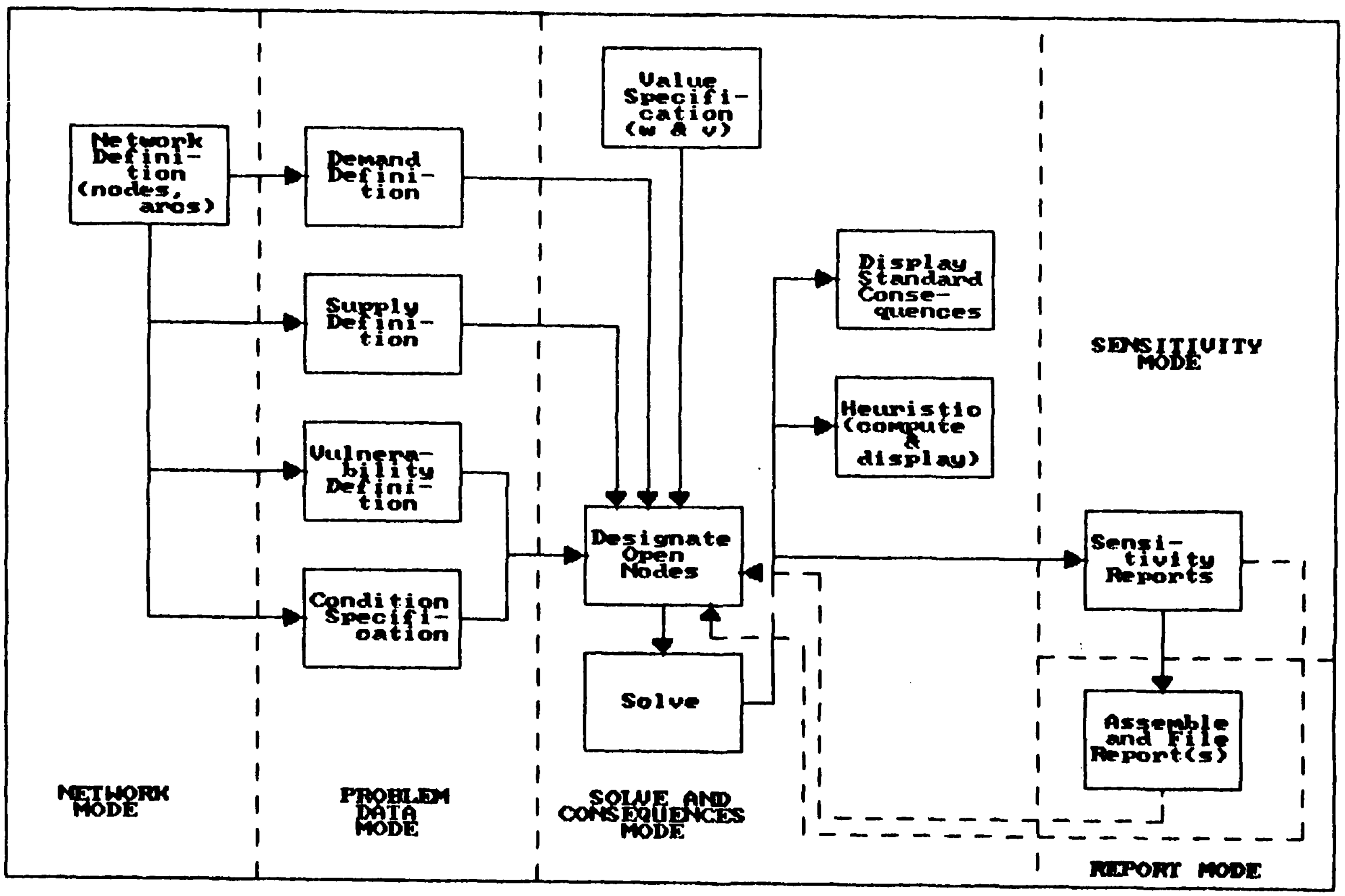
Network Defini- tion (nodes, arcs)
Demand Defini- tion
Supply Defini- tion
Vulnera- bility Defini- tion
Condition Specifi- cation
Value Specifi- cation (w & v)
Designate Open Nodes
Solve
Display Standard Conse- quences
Heuristic (compute & display)
Sensi- tivity Reports
Assemble and File Report(s)
NETWORK MODE
PROBLEM DATA MODE
SOLVE AND CONSEQUENCES MODE
SENSITIVITY MODE
REPORT MODE

FIGURE 4.1

SPL FUNCTIONAL BREAKDOWN (5 MODES)

The standard CGA monitor serves as a text monitor for the SPL. The text monitor contains two windows which will appear at all times. The windows are a "workspace window" for data echo and user input, and a "menu window" for display and selection of menu items. Messages to the user, instructions, data queries, etc. will appear at the lower portion of the screen below the workspace window. The high resolution RGB monitor is used to display network graphics, network text and data, and sensitivity analysis reports.

Note that the arrows in Figure 4.1 represent major data streams only and not any particular sequence of activities. Network definition and problem data are combined in the Problem Data Mode to provide all problem definition data to the Solve and Consequences Mode. Specification of lag and loss values is required in the Solve and Consequences Mode rather than the problem definition mode because this expresses the values placed by the user on various desiderata, controlling how the consequences of a solution are interpreted, not the computation of the consequences themselves. The Sensitivity Mode requires output data from the Solve and Consequences Mode in order to generate and display sensitivity reports. This data combined with sensitivity report data is provided to the Report Mode for assembly and storage, and printing if necessary. User input provided in the Sensitivity Mode and/or Report Mode may be returned for further analysis and computation to the Solve and Consequences Mode.

The interactive optimization solution structure as described in Section 3.3 requires a combination of modes and the major functions within the modes. Problem data is read in the Network Mode and the Problem Data Mode. Value specifications for lag and loss are read and an initial set of open nodes is designated in the Solve and Consequences Mode. Once all of these data are available to the "Solve" function, the function can solve a shortest route problem, can reassemble the data, and can solve a transportation problem for each open node. The system can then enter a "decision phase" (see Figure 3.3) and results are provided to the user who may then select an analysis procedure using a series of sensitivity analysis reports, obtain a heuristic solution by solving one or more of the available heuristics, or by doing both. At the end of a single iteration of the process the user may opt for more analysis, terminate, or modify problem data or plan data. If the user desires to modify any of the problem data, for example changing supply or demand at any node or nodes, modifying vulnerability criteria, adding or deleting a link in the network, etc., the system will enter the Network Mode and/or the Problem

Data Mode to assemble and process the new data before reentering the Solve and Consequences Mode to obtain a new solution. Because of the flexible structure of the interactive optimization solution technique, all modes can be employed during any given solution process.

4.2.b Major Functions Within Modes.

Each mode in the SPL architecture is composed of one or more major functions. Each mode may be composed of up to four levels of functions. A major function may include several functions which may in turn be composed of several subfunctions. At the lowest level subfunctions are accomplished by a series of tasks.

The "Network" Mode consists of one major function which is composed of three subordinate functions:

- Node Creation
- Link Creation
- Network Semiscaling

Each of these functions includes several tasks or subfunctions which must be performed. Subfunctions completed by the node creation function include selection of a location (x,y coordinates from grid coordinates), specification of attributes such as identification, location, elevation, etc. (stored as a vector of attributes), movement to a new location, and deletion of the node. Note that not all of these tasks are required each time the function is executed.

The user may add nodes to the current network by modifying node attributes, which include the new location. Nodes may also be deleted. All other attributes of a deleted node are deleted when its identification attribute is deleted.

Subfunctions completed by the link creation function are similar to node creation except that when a link is created endpoints (in terms of tail and head node identifiers) must be specified rather than a location, and length and width (road classification) are included as attributes.

Tasks performed by the semiscaled network function include the selection of ranking and spreading parameters (or default values), performing the rescaling, and printing the network to the graphics screen.

The second mode in the SPL architecture is a "Problem Data Mode." Major functions in this mode include:

- Demand Definition
- Supply Definition
- Vulnerability Definition
- Condition Specifications

Each major function in the Problem Data Mode defines problem data and assembles this data for use during the solution phase.

The "Demand Definition" function reads demand data for each of the demand nodes (sinks) that represent supported units. These data $\{b_j\}$ are received as input and stored as a vector of unit demands by the SPL. Demand associated with a single supported unit node may be modified by changing the appropriate demand node attribute. A second value required in the solution is the inventory kept at a supply point i $\{a_i\}$. This value may be modified as desired as a node attribute and under some circumstances may vary between supply points. However, current doctrine specifies three days of supply (DOS) at brigade and division support areas.

"Supply Definition" is the next function performed in the Problem Data Mode. As with demand definition, supply will be received as input by data extraction from a parent database or entered by the user for the associated supply node i and is stored as a vector of support area supply data (or node capacities) by the SPL. Modifications may be made to the node attributes file if the user desires to change supply data. Storage requirements for both demand and supply data are relatively small because in an Army Division area of operations the number of supply and demand nodes will be fairly small (less than 20).

Vulnerability definition is the next major function that must be accomplished in the problem data mode. All nodes and links must contain vulnerability attributes, p_i and q_{ij} respectively. These vulnerability attributes are provided as problem data and must be available to the Solve and Consequences Mode for solution of the problem.

Generally there are three methods by which the vulnerability attributes may be provided to the SPL. The first method is direct extraction from a CSSCS or tactical database. A second method is some type of approximation method or estimate such as the exponential decay method discussed in Section 3.1 which uses a gradient scale for determining default vulnerabilities based on general ranges of enemy weapon systems and distance of the node or link from the FLOT. A final method for obtaining the vulnerability attributes is direct entry by the user.

All values for node and link vulnerability may also be modified by the user by changing the appropriate vulnerability attributes for the respective nodes and links. This capability is important to the user in the case where there is more recent information available that does not appear in the data files such as the mining of a road or a known ambush site. In this situation the user may wish to delete a link or impose a very high vulnerability on the link.

Condition specification is a fourth major function in the Problem Data Mode. This function is used to provide information to the SPL such as average convoy speed, scale parameter for vulnerability computations, number of days of supply (inventory) maintained at supply points, etc. The user is provided the opportunity to input these values or modify them once provided. Default values are available for each of these parameters if no values are specified.

The third mode in the architecture of the SPL is the "Solve and Consequences Mode." In this mode value specifications for lag and loss are entered, open nodes are designated, and the actual computations for the solution of the location-allocation problem are made. Portions of the analysis during the decision phase are made in this mode by the solution of user defined heuristics. Major functions in the Solve and Consequences mode include:

- Value Specification
- Designation of Open Nodes
- Solve
- Display Standard Consequences
- Location-Allocation Heuristics

Value specifications (weights) for loss and lag are the first major function in this mode. Weights for lag (w) and loss (v) are specified by the user based upon the tactical situation and the commander's guidance. Two options for specifying values for lag and loss are available to the user. The user may choose to use either absolute weights or relative weights of lag and loss as discussed in Section 3.1. Default values for lag and loss ("value of 1 day lag and value of 1 ton-day loss) are available based on experimentally determined results which tend to give equal influence on the result to w and v. These values may be modified by the user during the sensitivity analysis or "what if" activity of the Decision Phase.

Designation of open nodes is also a function which must be accomplished in the Solve and Consequences mode. Initially, the user must designate a set of open nodes to be used during the initial solution of the problem. During the decision phase and subsequent solutions, the user may designate a new set of open nodes, modify the existing set, or use a set of open nodes recommended by the heuristic solution. This function also allows the user to close nodes as appropriate. If a set of open nodes is not initially designated by the user, an initial solution cannot be computed, resulting in an error message.

The third and most elaborate major function in this mode is the "Solve" function. This major function includes several subordinate functions each of which performs numerous tasks during the Solution Phase of the problem. Recall that the two-phase solution procedure of the location-allocation problem used by the SPL reduces the problem to a series of network flow problems. Each network flow problem is solved by first solving a shortest route problem and then a transportation problem. Subordinate functions to the Solve function include the shortest route algorithm which generates a cost matrix, an assembly routine which assembles the data used by the transportation algorithm, and a transportation algorithm for solving the transportation problem.

The "Solve" function consists of five blocks or subfunctions as illustrated in Figure 4.2.1. The network definition data is received as input to the first block which converts the information to shortest route input data. This data is dynamically dimensioned and stored as a matrix of arc lengths. A shortest route algorithm then solves the shortest route problem which outputs and stores a second matrix of "path lengths" or cost matrix L. The values in this cost matrix (L_{ij}'s) represent the minimum costs of lag and loss along all paths as described in sec. 3.4. A transportation algorithm is then solved using the cost matrix as input and the objective function:

$$\text{Min } Z = \sum_i \sum_j L_{ij}x_{ij} + \sum_i vp_i\alpha_i \sum_j x_{ij}$$

The output from the transportation solution is then converted to flow and the objective value representing the sum of the minimum costs of lag and loss along the selected path is received as output and provided to the Display Standard Consequences function for display. Values for the two components of the objective function, lag and loss, are also provided to the Display Standard Consequences function for display as output. These values may be either relative or absolute values, depending upon which method the user designated in the Value Specification function.

The next major function in the Solve and Consequences mode is the "Display of Standard Consequences." This function echoes back the current solution and information required by the user for further analysis. This information includes the objective value from the solved transportation problem and the two components, estimated lag and estimated loss as stated above. Data showing estimated lag and loss and their combination (Z) for each of the open nodes is also echoed to the text screen by this function for relative comparison.

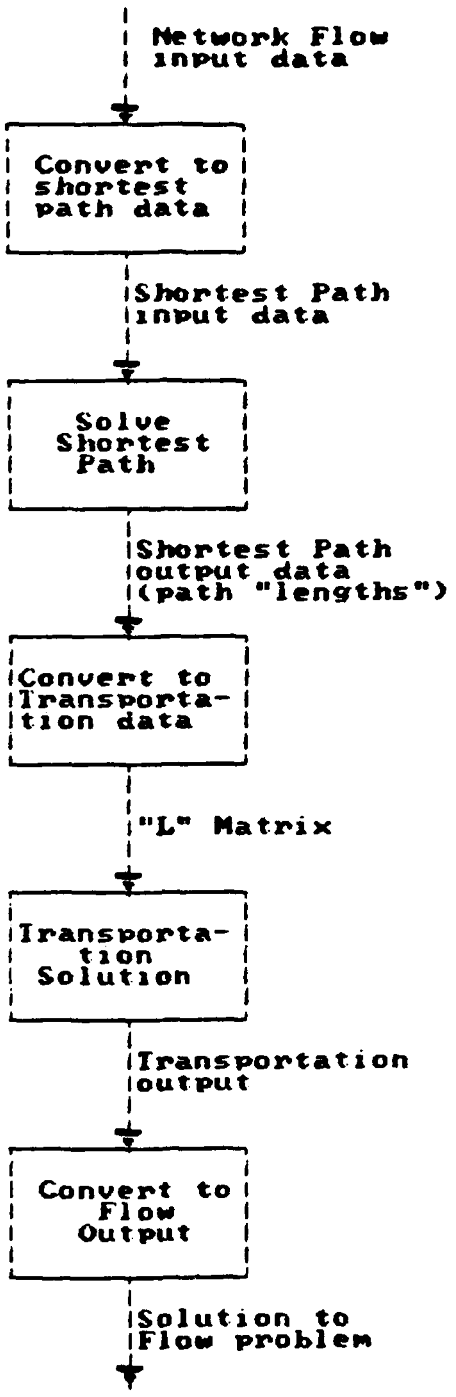

Figure 4.2.1
Solve Functions

The final major function in the Solve and Consequences mode is the Heuristic function. This function is included in the design as an added feature which will solve the location-allocation problem heuristically, by one or more of several available heuristics, and offer the user a recommended set of open nodes. There is no guarantee of obtaining an optimal solution using the heuristics, but a reasonably "good" solution may be expected. The solution of the heuristic also provides the user results with which to compare a solution obtained through the primary solution technique using interactive optimization.

The heuristics use the same problem formulation as the primary solution technique in solving the location-allocation problem. The SPL could use as many different heuristics in this function as desired by the designer. One heuristic is designated as the "primary" heuristic and all others will be "secondary" heuristics.

The fourth mode in the SPL architecture is the Sensitivity Mode and includes only one major function with six subfunctions all of which generate sensitivity reports for the user. These reports include:

- Transportation Schedule
- Most Efficient Supply Point
- Plan Comparison

- Most Costly Supported Unit
- Expected Loss vs Supply Node
- Expected Lag vs Supply Node

The user must select which reports to see. These reports, except for the Transportation Schedule, will appear on the graphics monitor when requested by the user. This mode also allows the user to do substantial "what iffing" by modifying problem data and re-solving the problem using new parameters in the objective function.

The final mode in the SPL architecture is the Assemble and File Report mode. The primary functions of this mode include filing solutions under a problem name and printing a report from the Display Consequences Mode when the user desires a hardcopy of the output. A "problem data" set is assigned a name and may include several different solutions for the different trial sets of open nodes. Each subsequent solution using different "plan data" is filed sequentially under the problem name (problem data set). If the problem data is modified, i.e. any changes other than the set of open nodes, a new problem name is assigned and the solution filed under the new problem name.

4.2.c Unique Characteristics of the Prototype SPL.

Throughout the duration of this research every attempt has been made to design a decision support system which could eventually be implemented by the U.S. Army. The design is based on both current technology and anticipated developments in computer systems, data communications, and databases available to U.S. Army units by the mid 1990's. Because some of these systems have not yet been fully developed, such as the Tactical Terrain Database and the Combat Service Support Computer System, modifications to the Prototype SPL are necessary in order to provide a "working system." None of the modifications to the Prototype SPL have violated the integrity of the "theoretical SPL" presented in Section 4.2.a and 4.2.b, and which should evolve from this research, the completed prototype, and future improvements.

The design architecture of the Prototype is identical to that illustrated in Figure 4.1 and discussed previously. The Prototype SPL employs the same five modes and the same major functions discussed, however, some of the functions have been modified to use simulated databases, files, and unique methods of determining default values.

The data structure of the SPL prototype is the same except that it uses data provided by files that simulate data extraction from a parent headquarters' database and/or by the user rather than data actually extracted from a parent database. The preponderance of the data requirements are comprised of node and link attributes. Node and link attributes are stored as vectors of data, for example, x and y coordinates for node locations are each stored as a vector of attributes. Other attributes stored by the SPL in a similar manner include demand data, supply data, node and link vulnerabilities, node identification number, etc. Rather than extract this data from a parent database and file it, the data for these files in the prototype SPL have been provided separately.

The Network Definition major function of the SPL prototype performs the same three functions: Node Creation, Link Creation, and Network Semiscaling. The Network Definition major function draws the network on the graphics screen and allows the user to add and delete nodes and links as necessary during the Decision Phase of the interactive optimization solution technique. The node locations and center line distances for the connecting links are read from files in the prototype SPL which simulate the tactical terrain and road network databases. Default values for the semiscaling parameters are set to "no semiscaling" (ALPHA=0) and will result in the display of the scaled network with no distortion.

The major functions in the Problem Data Mode of the prototype SPL are the same as those discussed above. The

"Demand Definition" function reads demand data $\{b_j\}$ for the supported units from the simulated data file. These values are stored as a vector of unit demands by the SPL prototype. Demand associated with a single supported unit may be modified during the interactive solution procedure. The value for the inventory maintained at a supply point $\{a_i\}$ is also stored as a vector and may be modified as necessary. Current doctrine specifies three days of supply at Brigade and Division support areas. This value should be specified, but the SPL prototype uses a default value of "3" which may modified later as a node attribute if necessary in the solution process.

"Supply Definition" is the next function performed in the Problem Data Mode. As with demand definition, supply could be read from the simulated data file for the associated supply node i, but will normally be provided by the user, and is stored as a vector of support-area supply data by the SPL. These data must be provided to the SPL prototype as there is no default. A reasonable default value could be extracted internally from a standard supply table for different types of combat units, e.g. Mechanized Infantry Division, Light Infantry Division, Armored Brigade, Mechanized Infantry Brigade, etc.

Vulnerability definition is the next major function that must be accomplished in the Problem Data Mode. All nodes and links must contain a vulnerability attribute, p_i and q_{ij} respectively. As discussed in Section 4.2.b, there are generally three methods by which vulnerability data may be estimated or computed. These include: direct extraction from a parent database, approximation or estimation by some heuristic method, and by being provided directly by the user.

The prototype SPL accommodates each of these methods of vulnerability definition to varying degrees. The first method is emulated by the provision in the data structure for vulnerability data as node and link attributes. The second method of vulnerability "estimation" has also been implemented by providing a heuristic which uses a gradient scale for determining default vulnerabilities based on general ranges of weapon systems and distance of the node or link from the FLOT as described in Sec. 3.1.b.2. This method is used as the default by the SPL prototype. The user-provided method of vulnerability definition has also been implemented by the SPL prototype by allowing the user to modify vulnerability data as node and link attributes.

The first major function in the Solve and Consequences Mode is the Value Specification function which provides risk guidance values (w and v) to the Solve function. The SPL prototype enables the user to specify risk guidance by either of the two methods described in Chapter III, by absolute values of lag and loss or by using relative values of lag and loss to specify risk guidance. Default values for lag and loss (value of 1 day lag/value of 1 ton-day loss) are available to the prototype SPL based on experimentally determined results which locate the supply point according to current doctrinal standards. These values may be modified by the user during the sensitivity analysis or "what if" activity of the Decision Phase.

The SPL prototype employs the same method for the next major function of Designation of Open Nodes as discussed in Section 4.2.b, except that all entries will be made from the keyboard rather than with the mouse. The solution technique employed by the Solve major function of the SPL prototype is also exactly the same as that described in Section 4.2.b and illustrated in Figure 4.2.1.

The final major function in this mode is the Heuristic function. The prototype SPL includes only one location-allocation heuristic, which may be selected by the user during the Decision Phase of the solution procedure. The heuristic used by the prototype is based on Cooper's location-allocation solution technique [9]. Hooks are available within the program structure for additional heuristics.

The Cooper heuristic is the primary heuristic for the prototype SPL and also serves as the default. If the user has not designated an initial set of open nodes, but asks for a solution, an error message will result. During the Decision Phase if the user selects the heuristic solution from the menu, the problem will be solved by the heuristic and results echoed to the Display Consequences function. The results of the heuristic solutions are recommended sets of open nodes and may or may not be accepted by the user for further investigation.

The Prototype SPL includes only one heuristic location-allocation procedure in the Analysis function. This procedure uses an uncapacitated location-allocation heuristic method proposed by Cooper [9], as cited by Banks [4]. The heuristic did not yield satisfactory results in final testing. The results produced by the heuristic procedure are inconsistent with the primary solution technique, because the primary solution technique does not allow supply nodes to be collocated with demand nodes. The heuristic often locates a supply node at the same location as a demand node with large demand. The heuristic also does not necessarily select a "suitable" node when locating the supply points because it does not distinguish between a suitable (candidate supply point) or an unsuitable node. A capacitated heuristic would probably work better in that capacities could be set to zero for unsuitable nodes.

The functions performed by the final two modes, Sensitivity Mode and Reports Mode, in the SPL prototype are exactly the same as those described in Section 4.2.b. The same choice of sensitivity reports are offered to the user by the Sensitivity Reports function and the reports are generated in the same manner. The filing and assembly tasks performed by the Assemble and File Reports function are also the same.

4.3 User Documentation

The SPL interface is designed to perform a logical sequence of activities and assist the user throughout the interactive solution procedure. The main menu presents the tasks which must be performed in a sequential manner allowing the user to progress through the menu during the solution procedure and returning to a previous task when required. The main menu is visible at all times except in some situations where computations are being made and when in the submenu during the Analysis phase of the solution process. The main menu returns immediately upon completion of computations or when returned to from the ANALYSIS submenu. The menu design allows the user to progress through the solution structure providing input to the program as it is required either interactively or from data files. Figure 4.3.1 shows a general method of problem solution using the SPL.

In order for the SPL program to run successfully there are four files which must be present in the default drive to run the SPL:

1. SPL.EXE
2. BRUN41.LIB
3. HALO9T.DEV
4. HALO104.FNT

SPL.EXE is the main Supply Point Locator Program. BRUN41.LIB is a QUICKBASIC (version 4.00b) library file that contains routines used by QUICKBASIC. HALO9T.DEV is the HALO graphics device driver for the selected hardware (see Section 4.1.a). HALO104.FNT is the HALO graphics character font used by the SPL to write text on the graphics screen. These files should already be included with the SPL Program Disk, however, if they are not, they must be added in order for the program to execute successfully.

The default drive for the SPL Program Disk is the "A" drive. Output files from the SPL will be written to this drive. The SPL may also be run from a hard disk drive if

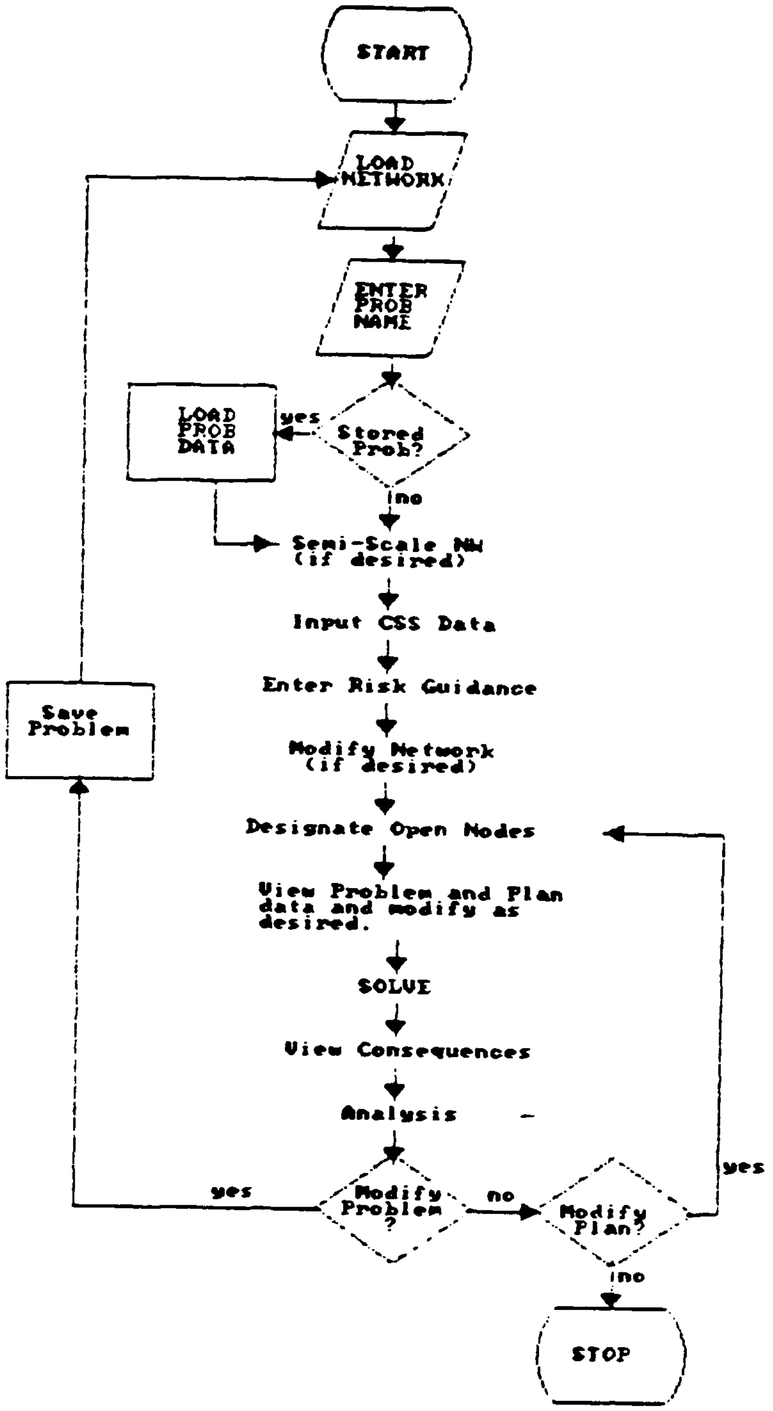

Figure 4.3.1
Flow Chart of User Actions

the above listed files are present (default drive would be the "C" drive). The default drive for the SPL <u>data</u> disk is the "B" drive. This cannot be changed without modification of the SPL source code. All network and terrain and roadway data should be included on this file unless it is to be entered interactively. The "B" drive must contain a data disk in order for the program to execute.

Begin a session by placing the SPL program disk in the "A" drive (or enter the directory containing the SPL.EXE file and the above listed files if using a hard disk drive) and the SPL data disk in the "B" drive and typing "SPL" at the DOS prompt.

4.3.a Main Menu

The "Main Menu" for the SPL Prototype appears upon execution of the program. This menu offers twelve different choices for functions. The choices are numbered F1 through SF3. Menu items may be selected using the optional mouse or from the keyboard. To select a menu item using the mouse, move the mouse cursor over the number of the item to be selected and "click on" the mouse to make the selection. To select a menu item from the keyboard, press the "function key" corresponding to the appropriate menu item. The numbers that include an "S" before the function key number indicate that the "shift" key must be pressed simultaneously with the function key. For example, to select "SOLVE," press the F1 (function key) while holding down either of the two "shift" keys (press <Shift>F1). There are times during the solution process when the main menu is not visible. If the menu is not visible a message to the user should be visible on the text screen indicating current status and an instruction to the user (e.g. "computing cost matrix ... please wait"). When the main menu is not visible, the user should not attempt to enter commands or menu selections. When the Main Menu window has reappeared and the mouse cursor is visible in the menu, control has been returned to the user and selections may again be made from the menu. Each menu item from the main menu is discussed in the following paragraphs.

4.3.b LOAD NETWORK FILES (F1)

4.3.b.1 Loading the Network. After executing the program and the main menu appears, the first item on the menu is "LOAD NETWORK FILES." This function must be executed in order to load or enter the network and terrain data. There are no default values for these data. Press "F1" (function key on the IBM keyboard) or move the mouse to this function and press to execute the "LOAD NETWORK FILES" function. Several important tasks are performed by this function. The user will first be asked to "Enter problem name." Enter a problem name using any character or numbers subject to DOS file naming requirements to assign a problem name. Do not include a "." after the file name or a file extension such as ".bas." After the problem has been named, press <Enter> to continue. The SPL then begins to load the network and terrain data. This data is contained in two files on the SPL Data Disk: NODE.DAT and LINK.DAT. If the program does not find these files on the data disk, the user will be prompted to enter the data from the keyboard.

If the program finds the two files it begins by reading the node data. Node data is in a special format on the data disk which contains a record containing 6 words or attributes for each node. These attributes include: a sequence number, easting coordinate, northing coordinate, elevation, terrain type, terrain description, and a "suitability" code. The NODE.DAT file must be written in this format. The user has two options as the data is being read by the computer; he may view each record as it is being read or he may press "Q" and the data will be read without allowing the user to view it. The latter option is much faster.

After the node data has been read the program reads the LINK.DAT file (if found) exactly as it did the node data file. The LINK.DAT file is in a similar format and includes: an identification number, head node number, tail node number, length (in kilometers, not meters), and road classification, as link attributes. The user again has the same options of viewing each record as it is read or pressing "Q" to continue without waiting.

The function next looks for map corner data. These coordinates define the map area covered in the data file by identifying the lower left hand corner and the upper right hand corner of the map area. This data is contained in the file CORNERS.MAP. If the file is not found on the data disk the user will be prompted to enter the data in the format

"max easting,min easting,max northing,min northing."

Once the map and terrain data has been read or entered and the map area has been defined, the network should appear to scale on the graphics screen for view and use by the user.

4.3.b.2 Network Display. When the network is displayed throughout a solution session, nodes and links may appear in several different colors.

Blue nodes indicate "suitable" nodes for supply point location. Black nodes indicate unsuitability and should not be designated by the user as supply point locations. Orange nodes indicate supported unit locations (demand nodes) and are not allowed to be designated as supply points. Nodes designated as "open," or supply point locations appear as blue "unfilled" circles, when the network display is replenished after open nodes are designated.

All links in the network will appear as green lines on the screen. During the "Solve" phase, the selected path from a supply point to a demand node will be traced in purple. When the network display is redrawn after a solution, the previously selected path will be erased and all links in the network will again appear drawn in green.

4.3.c LOAD PROBLEM FILE (F2)

The "Load Problem File" function is the second item on the main menu. This item is selected when the user wishes to perform more analysis on a previous problem that has been filed under a specific problem name. If the problem data from a previous problem is not desired the user should not select this function. If this function is selected, the user must first place the SPL Data Disk containing the desired problem data and the appropriate network files in the "B" drive, select the function by pressing F2 or selecting it with the mouse, then enter the problem name of the previous problem when prompted. The "LOAD PROBLEM FILE" function is optional and should only be used when the user desires to continue working on a previously entered problem data set.

4.3.d SEMI-SCALING (F3)

The SEMI-SCALING function is used when desired by the user as discussed in Section 3.2. If the user desires to semi-scale the current network he does so by selecting "F3" (Semi-Scaling) from the keyboard or with the mouse. After selecting this function the user will be prompted to specify values for the Spreading Parameter "Alpha" and the Ranking Parameter "Beta." Both values must be between 0 and 1. Hardware limitations limit this prototype to relatively small values of Beta; however significant improvement in the quality of the display is obtained using allowable values for the ranking parameter. If the ranking parameter entered is too large to be accommodated by the system, the user will be prompted to enter a smaller value for Beta. Experimentation indicates that acceptable values for Beta for problems of the size encountered in this implementation should normally be in the range: 0 < Beta <= .05. An acceptable value for Beta is obtained by trial and error by continuing to enter a smaller value or "bracketing" until the user has determined a satisfactory value that will be accepted. Normally the user would use the largest acceptable value for Beta and a value of 1 for Alpha to obtain the maximum amount of semi-scaling. A value of 0 for Alpha will return the network to its original scale. To try different values for Alpha and Beta the user re-selects the SEMI-SCALE function from the menu and then may enter new values for the two parameters. Note that each time this function is selected the network is redrawn on the graphics screen.

The current values for the parameters are indicated in the lower left hand corner of the graphics screen if the network has been semi-scaled. If the network has not been semi-scaled, or if the spreading parameter (Alpha) has been set equal to 0, these values will not appear.

The SEMI-SCALING function should be used throughout a session to redraw the network display when the network is not visible because the user has previously selected to view a report with the ANALYSIS function. For example, after a solution cycle the user has selected to view several of the analysis reports and then desires to designate a new set of open nodes. If the user wishes to view the network while designating the new set of open nodes, SEMI-SCALING should be selected from the main menu using the same values for the spreading and ranking parameters as used previously or selecting "0" for the spreading parameter (Alpha) if no semi-scaling is desired. The user may then continue with the selection of a new set of open nodes with the network visible on the graphics screen.

4.3.e ENTER CSS DATA (F4)

The next function to be executed is the "READ CSS DATA" function. CSS data which must be read from file or entered from the keyboard includes: locations of supported units, supported unit demands (in short tons), and the end points of the FLOT for use in vulnerability calculations. Locations of supply points, capacities, and average days of supply maintained at each potential supply point are provided as the supply points are designated in the "DESIGNATE OPEN NODES" function. The function looks for a file on the SPL Data Disk named DEMAND.DAT and reads the data if the file is present. If the DEMAND.DAT file is not found the user is prompted to enter the data from the keyboard. The function first asks for the total number of supported units and then asks for the node number (supported unit location) and the supported unit demand. The user is provided the opportunity to change this data as it is being entered by responding "yes" to the question "Do you want to make changes?" After this data is read or entered it is stored in a "demand" array.

The user may be asked to enter coordinates of the "endpoints" of the FLOT. If the SPL is using the default method of vulnerability estimation the endpoints of the FLOT must be used by the vulnerability routine to estimate distance from the FLOT. These points are entered as

X1,Y1,X2,Y2

where X1,Y1 is the "easting,northing" of the first endpoint and X2,Y2 is the easting,northing of the second endpoint. These coordinates must not exceed the area defined by the CORNERS.MAP file or an error statement will result, telling the user that one or both of the coordinates is outside of the defined map area. The northing and easting for both endpoints must be entered as a four place easting and four place northing coordinates (e.g. 9150,3842,0654,3826).

A hidden feature is installed in the SPL to allow the user to modify the current value of the vulnerability parameter (g) used by the default method of computing vulnerabilities. The user may modify, or view the current value of the parameter by selecting "<ALT>P" and answering the prompt. The default value for g used by the SPL is 0.06. This value is very sensitive and normally should not be modified during the solution process unless the user is certain that the modification is desired.

4.3.f RISK GUIDANCE (F5)

Risk guidance is entered by selecting "F5" from the keyboard or may be selected using the mouse. Risk guidance includes the value specifications for lag and loss entered by the user based on the tactical situation and commander's guidance. When "RISK GUIDANCE" is selected from the menu the user is prompted to select either "absolute" or "relative" values. Recall from Section 3.1.b.1 that there are two methods of providing risk guidance: the absolute approach where the user specifies values for both w and v, and the relative approach where the terms are normalized ($w = 1 - v$) and the user selects only one value for risk guidance (w).

The absolute approach is highly recommended over the relative approach. When using the absolute approach, the recommended method of selecting w and v is as follows. The user should determine how much more valuable 1 ton of loss is to the mission than 1 ton-day of lag, if at all. 1 ton-day of lag could be thought of as the delay of 1 ton of supply for 1 day, or 24 tons of supply delayed for 1 hour. The value of a ton of loss should include the loss of accompanying personnel and equipment as well as the ton of commodity. Always enter a value of "1" for loss (v), then if loss is 1, lag should be considered the reciprocal of the relative value between loss and lag. For example, if the user feels that the value of 1 ton of supplies and the accompanying personnel and equipment is worth 25 ton-days of lag, then the value specification for lag (w) should be equal to 1/25 = .04 (the value for lag is 1). This is a general rule, but should be followed in order to maintain proper scale and a basic understanding between lag and loss.

If the "relative method" of risk guidance is selected, then the user should insure that the sum of w + v = 1, and that "reasonable" values are used. Recommended values for w and v when using this method are as follows:

$$.90 < v < .99$$
$$.01 < w < .10$$

When the RISK GUIDANCE function is selected the user must first respond to a query to determine the desired method (absolute or relative) of selecting w and v. After responding to the first prompt the user is then asked to enter the values for w and v. The values for w and v should be entered together on the same line separated by a comma.

example:

```
        Enter the values for w and v:  .1,1
```

Note that the two values must be entered in the correct sequence (w,v).

Default values for w and v are available which assign approximately equal influence to lag and loss, if the user is unsure of the correct values to use. Default values are selected by pressing <RETURN> at the first prompt. It should be emphasized that the "absolute method" is preferred and provides the best results using the algorithm stated above for selecting w and v. This method will provide output in the form of "tons of loss" and "ton-days of lag" where the other methods provide only relative weights for the two values.

4.3.g MODIFY NETWORK (F6)

The "MODIFY NETWORK" function allows the user to modify network data during the solution procedure by allowing the user to change node and link attributes for the network. Nodes are identified by number and links are identified by their head and tail node numbers. Either node may be designated as the "head node" when identifying a link, while using the other as the tail node (links are non-directional). When a node or link is identified after the user has been prompted to identify the node (link), then node attributes are cycled through and new values may be entered. Because each set of attributes must be "cycled" through, the current values are displayed and if no modification of a particular attribute is desired the user may hit <RETURN> without changing the value of that particular attribute. This is the only method of modifying node/link attributes. Nodes/links are added and deleted by this function and this is also the method by which node/link vulnerabilities are modified. Vulnerabilities assigned by the default vulnerability estimation heuristic or read from a data file are modified by changing the appropriate node or link attribute for a particular node/link.

Whenever nodes are deleted from the network, the user must insure that all links associated with that node are also deleted. Likewise, when a node is added to the network, the appropriate links must also be added, and all node and link attributes must be entered using the procedure describe above.

If the user desires to view the current values of the node and link attributes (without making changes), this function may be use for this purpose. Select "modify" node (link) and view the current values as the appear in the workspace window (press <RETURN> to advance to the next attribute without making changes).

4.3.h DESIGNATE OPEN NODES (F7)

"DESIGNATE OPEN NODES" is selected from the main menu by pressing F7 or selection with the mouse. A set of open nodes must be entered by the user before the SOLVE function can be selected to obtain a solution and before the heuristic solution method can be selected in the ANALYSIS function. When "DESIGNATE OPEN NODES" is selected, the user will first be prompted to enter the "number of supply nodes being entered." Upon entering the appropriate number of supply nodes and pressing RETURN the user will be asked to enter the node number of the first supply point. After the node number is entered the user will be asked to enter the supply available at the supply point. The user will then be asked to verify the node number and available supply by answering yes or no (Y/N) to the prompt, and the sequence will be repeated for the remaining number of supply points being opened. Care should be taken during the entry of data in this function because there is no "escape" if values are entered in the wrong sequence resulting in having to restart the program/problem from the beginning by reentering the network and following the sequence as listed above. Since the set of open nodes is designated anew for each "plan," the only way to open a node, close and node or keep a node open is to redesignate the set.

4.3.i SOLVE (F8)

After all data and parameters have been provided to the SPL, a current solution may be obtained by selecting the "SOLVE" function. After this function has been selected and computations are made by the SPL, a list of standard consequences of the current solution are output to the text screen for the user to view the current value of the objective function (combined lag and loss), the two components lag and loss, total flow, unmet demand, and unused supply. The units of each of these quantities is short tons (S/T) (except that the units of lag are "ton-days"). Note that this solution is for the current set of problem and plan data only (i.e. for the current set of open supply nodes).

Output from the SOLVE function will include a Problem/Plan name and number, e.g. EAGLE / 2, indicating "Plan 2" of Problem "Eagle." A military date time group indicating the time that the current Problem/Plan was solved will also appear on the output and on all reports in the ANALYSIS function.

4.3.j ANALYSIS (SF1)

After a current solution is obtained by the "SOLVE" function, the user may select the "ANALYSIS" function in order to conduct more detailed analysis on the current problem as described in Chapters III and IV. When the user selects "ANALYSIS" (<shift>F1) from the main menu a submenu will appear on the text screen in place of the main. The ANALYSIS submenu appears in "blue" to avoid confusion by the user as to which menu is current (the Main Menu is green). Selections are made from this submenu in the same manner as from the Main Menu. The user may select any of these menu items which include a heuristic solution to the current location-allocation to obtain a recommended set of open nodes. The user may select any or all of the six reports available in this functior by selecting the appropriate menu item. All reports except the "Transportation schedule" will appear on the graphics screen as a graph. The Transportation schedule is printed on the text screen as a supplement to the "View Consequences" data output from the "SOLVE" function.

4.3.j.1 Transportation Schedule. This report as illustrated in Figure 5.3.2 provides a recommended transportation schedule for the current solution. When the user selects this report a prompt will appear asking the user if he desires the output be sent to the screen or to the printer. The selection is made by pressing "P" (for printer) or "S" (for screen). The user will be provided an opportunity to print output by the "PRINT" function as well as here, and may desire to have the output sent only to the screen and delay printing until later in the solution process. Values for total cost (total expected lag and loss), loss, lag, total flow, unused supply, and unmet demand, losses at supply points, losses in transit, and lag in transit are provided with this output just as in the view consequences output. The report also shows which supply nodes support which supported units, the quantities provided, and the costs. Current values for w and v are also printed at the end of this report.

4.3.j.2 Plan Comparison. This report is an XY graph, and is used to compare the current plan against previous plans under the same set of problem data. Expected lag and loss for each plan are plotted on the same axis to allow the user to graphically compare the consequences of all of the different plans (different sets of open nodes). This report should assist the user in selecting the best set of open nodes for the given problem data set.

4.3.j.3 Most Efficient Supply Point. This report is an XY graph which plots the values of lag and loss against each of the open supply nodes. Both lag and loss are plotted for each supply node, and each of the supply nodes is on the same axis allowing the user to graphically compare the values of lag and loss for each supply point.

4.3.j.4 Most Costly Supported Unit. Similar to the Most Efficient Supply Point Report above, this report is also an XY graph which plots the values of lag and loss against each of the demand nodes. Lag and loss are plotted for each of the demand nodes, and each of the demand nodes is on the same axis allowing the user once again to graphically compare the values of lag and loss for each supported unit.

<u>4.3.j.5 Expected Loss vs Supply Node</u>. This XY graph plots only the value of the "expected loss" component against each of the open supply nodes for quick comparison by the user.

<u>4.3.j.6 Expected Lag vs Supply Node</u>. Like the previous graph, this report plots only the value of the "expected lag" component of total cost against each of the open supply nodes.

<u>4.3.j.7 Modify Data</u>. This option is not a report but assists in the decision process by prompting the user and providing instructions in order to continue. The user is first required by prompt to decide if he desires to modify the problem data. If the user desires to change the problem data or the plan data, or both, he is instructed on where to proceed for the next activity, i.e. if the user desires to change the problem data, he must go to "LOAD NETWORK FILES," specify a new problem name, etc. and continue through the solution procedure entering new or modified data as appropriate. If the user desires to modify only plan data, he is instructed to return to the "DESIGNATE OPEN NODES" function and continue.

The ANALYSIS menu is exited by selecting "RETURN MAIN MENU." This takes the user out of the ANALYSIS menu and returns to the MAIN menu.

4.3.k PRINT (SF4)

The "PRINT" function allows the user to print output from the current solution if this was not accomplished during analysis. The user may select to either print the output to the screen or to the printer.

4.3.1 QUIT (SF5)

The "QUIT" function causes the SPL to end and returns control to DOS. This function prompts the user to verify that leaving the SPL program is the desired action. If the QUIT key (SF5) was pressed inadvertently, the user may press "N" and return to the main menu.

CHAPTER V

EXPERIMENTATION

5.1 Verification Testing

Prior to performing experimentation on a complete test problem it was necessary to test several of the individual algorithms within the SPL in order to verify their performance and accuracy.

The first algorithm tested was the heuristic method of computing vulnerabilities used by the SPL. Verification of the performance of this algorithm after the computer code was completed was done by running the vulnerability routine on the computer for a large test problem and comparing manually calculated values with those of the output from the vulnerability routine. All values that were checked (approximately 10) were very close (plus or minus .0001) to those calculated by the vulnerability routine. The distances computed by the vulnerability algorithms used to compute the final values were compared with those measured from the map used for the test problem and were also very accurate.

The shortest route algorithm used by the "SOLVE" function of the SPL was tested several times while the computer code was being written and upon completion of the program. "Wagner's Shortest Route" problem [38] was used to test the routine after its completion. The results (distance and path) were compared with the primal problem solution and were exactly correct due to all integer distances used in the problem and no round-off error. The Wagner problem is a simple shortest route problem using one source, one sink, and six intermediate nodes and is easily solved manually.

The transportation algorithm used in the "SOLVE" function of the SPL was tested independently outside of the SPL by solving several well documented transportation problems, both balanced and unbalanced. The problems used in the testing were taken from an operations research text book by Hamdy A. Taha [37]. As with the shortest route algorithm, all integer input was used (demands, capacities, and transportation costs) in testing the transportation algorithm. Results for all test problems agreed exactly with the solution as presented in the cited reference.

The cost matrix provided by the shortest route algorithm to the transportation routine in the SPL provides a key function in the solution process. The interaction between these algorithms is the is the core of the SPL Program. The best method determined for testing the interaction of the two routines and their output was with a series of small problems which could be manually solved and compared with the output of the SPL. Only minor differences in the solutions to these problems were encountered (less than .01 difference) and the differences were attributed to computer and calculator round-off error. The problems solved manually consisted of one source and one or two sources connected by a single link. Larger problems would be time consuming to solve manually, and results obtained with the series of small problems was very encouraging.

The location-allocation heuristic used in the "ANALYSIS" function was tested outside of the SPL using an example problem by Banks [4] presented with the basic algorithm.

5.2 Test Problem

The example problem used in testing the Supply Point Locator is based on a modification of a scenario used by students in the Combined Arms Services Staff School at Fort Leavenworth, Kansas (see Appendix II). The problem uses a defensive scenario in the midwestern United States.

The map used for the exercise and in constructing the terrain and road network data bases is the same as that referenced in the operations order, Kansas-Missouri, 1:50,000, USACGSC 50-310 (see operations map, Appendix II, Special Situation).

Terrain data and road network data were extracted from the map and input files were constructed for use by the SPL. In actual practice, this data would be received through intelligence and operations channels and would be provided to the user.

Demand data and consumption rates have been approximated based on consumption rates of a mechanized infantry division provided in the above scenario. Exact demand data is not critical to the effective solution of the problem. Personnel and equipment strengths are based on the ALO 2 organization of the division as stated in the operations order.

This problem is designed to test the upper limit of the capability of the SPL. The network resulting from this problem contains 249 nodes and 446 links. Since 250 nodes is the maximum size problem for the SPL, problems solved using this network should serve as "extremes" tests for the SPL.

5.3 Solution Procedure

5.3.a The "Division" Problem

The main menu of the SPL is designed to facilitate an efficient, methodical, and systematic solution procedure by the user. For the initial problem the user should sequentially follow the menu choices offered by the main menu.

The network files are loaded and examined on the graphics screen. The display appears somewhat congested in the vicinity of node 61 (Nortonville), so semi-scaling should be desirable. A spreading parameter (Alpha) of "1" is first attempted for maximum spreading and an initial value of 0.03 for the ranking parameter. An error message is received stating that 0.03 is too large for the ranking parameter. The ranking parameter is reduced until an acceptable value is found. 0.018 is found to be the largest value that works for Beta. The network is semi-scaled, and although the difference is subtle, a significant improvement in the "viewability" of the area around node 61 can be detected.

Most of the CSS data including demand data, supported unit locations, etc. are provided by data files and no modifications are made during the initial solution attempt. This simulates the situation that all subordinate headquarters and support areas have been deployed and have notified the division tactical operations center (TOC) of their locations. In accordance with the operations order there are ten units (brigade size and separate battalions) which must be supported by the Division Support Command (DISCOM). Subordinate unit support areas which must be supplied from the DSA are located at nodes 46, 87, 93, 113, 120, 134, 174, 191, 194, and 215. Figure 5.3.1 shows the network display after entry of the demand data.

After entering the network and upon selecting "ENTER CSS DATA," the SPL queries the user for more CSS information including the endpoints for the FLOT to be used for default vulnerability calculations and days of supply maintained at the supply points (DOS). The doctrinal standard of 3 days of supply is used for this value (3 is also the default value). Value specifications for lag and loss are then asked for by the SPL and entered by the user.

Since no security force has been assigned to protect the Division Support Area, which means that the position will be defended only by organic security forces, and the division is in a defensive posture, command guidance is interpreted to mean that security of the DSA is more important than responsiveness. A value for "w" is selected which will locate the support areas further to the rear.

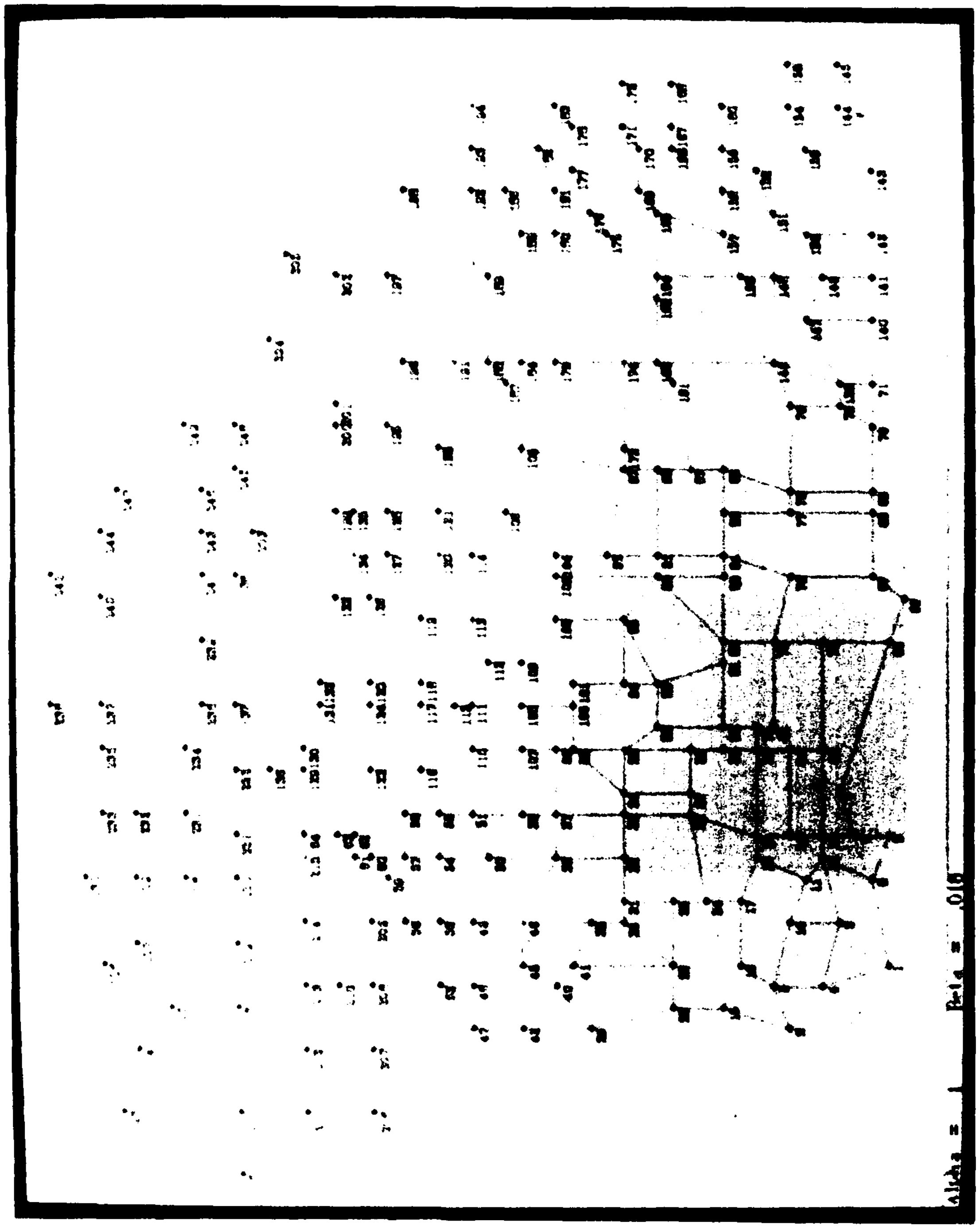

Figure 5.3.1: Network Display - Division Test Problem

Using the "absolute value" technique for specifying risk guidance, the initial values selected are 0.1 and 1.0 for lag and loss respectively. This reflects a value judgement that loss is approximately ten times as important as lag to the user. No nodes or links are deleted or added in the initial solution attempt.

All problem data has now been entered and a set of open nodes must be entered for the current plan. No firm decision has yet been made or guidance received on the number of supply points to open. The user decides to open three supply points for the initial solution attempt with one supply node located to the rear of each of the three brigade sectors. Nodes 209, 136, and 245 appear to be good candidates from a map reconnaissance and are designated as the open nodes. Note that these nodes are fairly far forward in the Division rear area, in accordance with the user's preference to start forward and work back in selecting subsequent sets of open nodes. A systematic approach should be used to save time and provide a more efficient solution procedure. Each of the supply nodes will be able to provide 170 short tons of supply per day to the supported units. The set of open nodes is designated and the network is redrawn with the open nodes appearing as unfilled circles.

All required data for the solution of the problem has now been entered or read from files. Once all data has been verified and the user is satisfied, the first solution attempt may be made by selecting "SOLVE" from the main menu.

Upon selecting the "SOLVE" function the SPL gives an initial solution with output as shown in Figure 5.3.2. As anticipated, from the initial selection of the absolute values for lag and loss specification and the relative difference between them, loss is less desired and has been weighted as such. Note that only a small portion of the lag and loss are sustained on the links, or while in transit, while a much greater portion is sustained at the supply points.

The results are reasonable for an initial attempt, but total loss seems fairly high and is probably a result of being at the forward edge of the division rear area. Better results should be obtained by locating the supply points farther to the rear. A new set of open nodes should now be designated. From the sensitivity reports resulting from the first set of open nodes it is determined that loss and lag are very high for node 209. The user may decide to move the supply point on the division right from node 209 to node 218, approximately a 7 kilometer shift in the direction away from the FLOT. The other two nodes are also moved back in

```
                          BEAR1     / 1
                          091153 DEC 88
                      Transportation Solution
                      -----------------------
        Schedule                                     Resupply
        --------                                     --------
  Source        Destination            Quantity (S/T)         Costs
 ------------------------------    ----------------------------------
  Node   209     Node   215                 20               2.929034E-02
  Node   209     Node   46                  90               .2068135
  Node   209     Node   93                  25               .1392472
  Node   209     Node   113                 26               .1415128
  Node   136     Node   113                 82               .3524601
  Node   136     Node   120                 27               .1195674
  Node   136     Node   134                 20               6.568207E-02
  Node   136     Node   87                  20               .2134298
  Node   136     Node   174                 20               .1797894
  Node   136     Node   194                 1                9.878021E-03
  Node   245     Node   191                 90               .2517819
  Node   245     Node   194                 80               .5347947
 --------------------------------------------------------------------
        Unused Supply =  9          Total Costs   =  255.5545
        Unmet Demand  =  0          Loss  (S/T)   =  254.8286
        Total Flow    =  501        Lag   (S/T)   =  .7259317

           Loss at supply point 209  =  103.6244
           Loss at supply point 136  =  79.17612
           Loss at supply point 245  =  70.50979

           Loss in transit to node 215  =  .0191982
           Loss in transit to node 46  =  .1486104
           Loss in transit to node 93  =  .1025642
           Loss in transit to node 113  =  .3307958
           Loss in transit to node 120  =  7.784862E-02
           Loss in transit to node 134  =  4.198416E-02
           Loss in transit to node 87  =  .1504624
           Loss in transit to node 174  =  .1194142
           Loss in transit to node 191  =  .1622747
           Loss in transit to node 194  =  .365163

           Lag in transit to node 215  =  1.009215E-02
           Lag in transit to node 46  =  5.820313E-02
           Lag in transit to node 93  =  3.668298E-02
           Lag in transit to node 113  =  .1631771
           Lag in transit to node 120  =  4.171875E-02
           Lag in transit to node 134  =  2.369792E-02
           Lag in transit to node 87  =  6.296742E-02
           Lag in transit to node 174  =  6.037527E-02
           Lag in transit to node 191  =  8.950721E-02
           Lag in transit to node 194  =  .1795098
```

Figure 5.3.2: Sample Output - Transportation Schedule

their respective sectors of the division rear so the next set of open nodes designated includes nodes 218, 231, and 244.

Upon solution using the new set of open nodes a significant improvement is noted in the total cost. Loss now accounts for 183.4849/184.5318 of the total cost compared to 254.8286/255.5546 for the initial set of open supply nodes. A third set of open nodes is now tried moving the supply points to the very rear of the division rear area. Nodes 226, 238, and 244 are now entered as the current set of open nodes and a solution obtained. Note that node 244 in the sector on the division left flank was used in the previous set of open nodes and appeared in the most efficient supply point report as very efficient, so it is decided to use this node in the next set of open nodes. Again, a significant decrease in total cost is obtained, but the gains are increasing at a slower rate as the supply points are moved to the extreme rear of the division rear area. The ratio of loss to total cost for the last plan (set of open nodes) is 135.4099/136.6055.

The network display after solution using the third set of nodes is illustrated in Figure 5.3.3. Note that the routes selected from the supply points to the supported units are traced on the network in purple. The reports selected by the user for analysis after solution of the third plan (nodes 226, 238, and 242) are shown in Figures 5.3.4 through 5.3.8. The Plan Comparison report (Figure 5.3.4) indicates that this set of open nodes yields significantly better results than the first two plans.

Several more attempts are made with different sets of open nodes (using 3 each time) and no improvement is obtained over the set of open nodes used in the third plan (226, 238, and 242). The user may conclude that using the current set of problem data, and particularly the value specifications for lag and loss, that locating far to the rear is the best option. This is illustrated by the "Plan Comparison" Report available to the user in the "ANALYSIS" function of the SPL.

If the user desires to change any of the problem data or the value specifications for lag and loss, this may be accomplished by selecting "MODIFY DATA" from the ANALYSIS menu and following the prompts. For example the user may decide to investigate what would happen it was decided that responsiveness is more important than losses. This would be done by creating a new set of problem data under a new name and entering the revised values for lag and loss. In order for lag to be weighted heavier than loss a very high value for w must be used relative to v. For example w = 10, v =

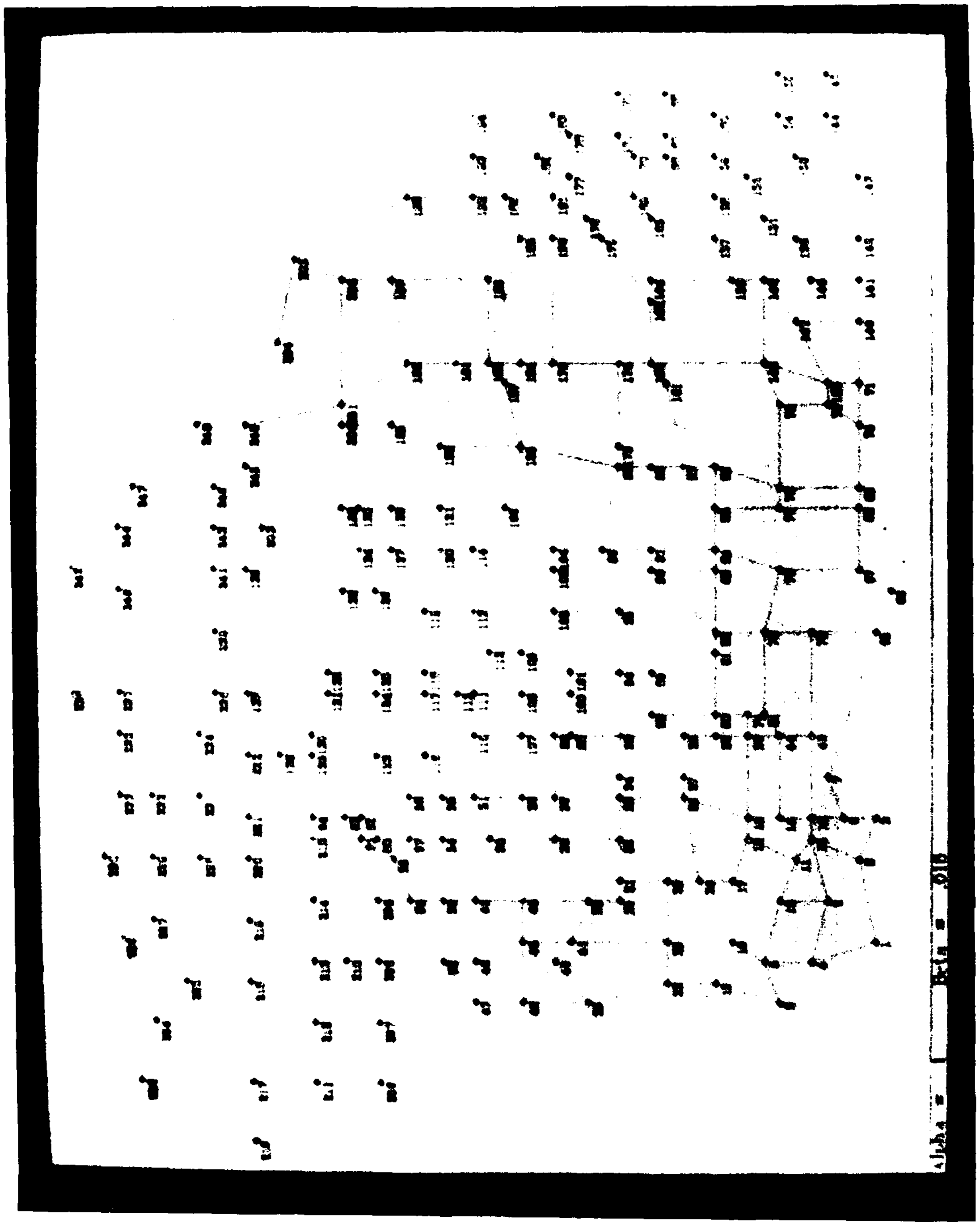

Figure 5.3.3: Network Display - Division Test Problem (Plan 3)

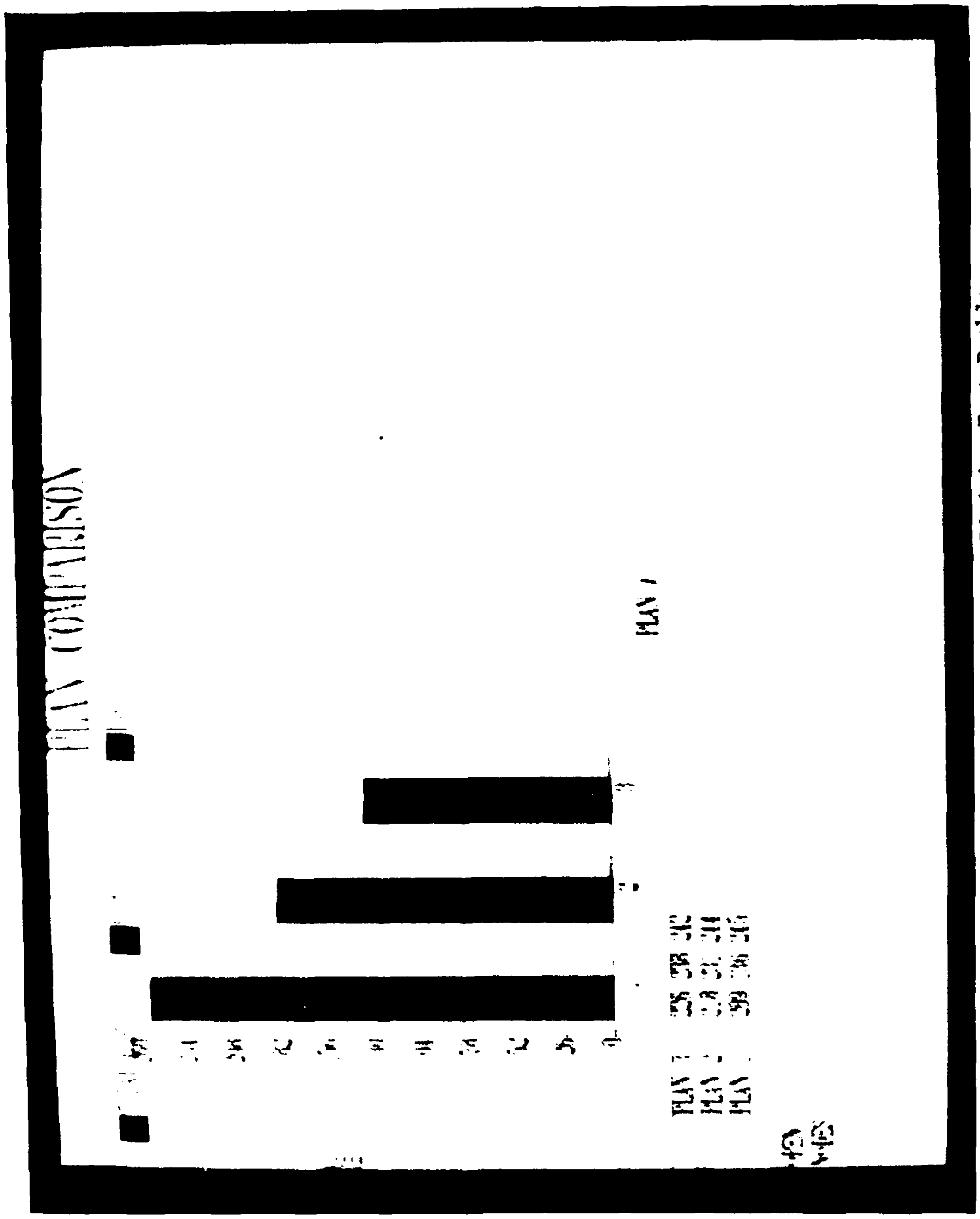

Figure 5.3.4: Plan Comparison - Division Test Problem

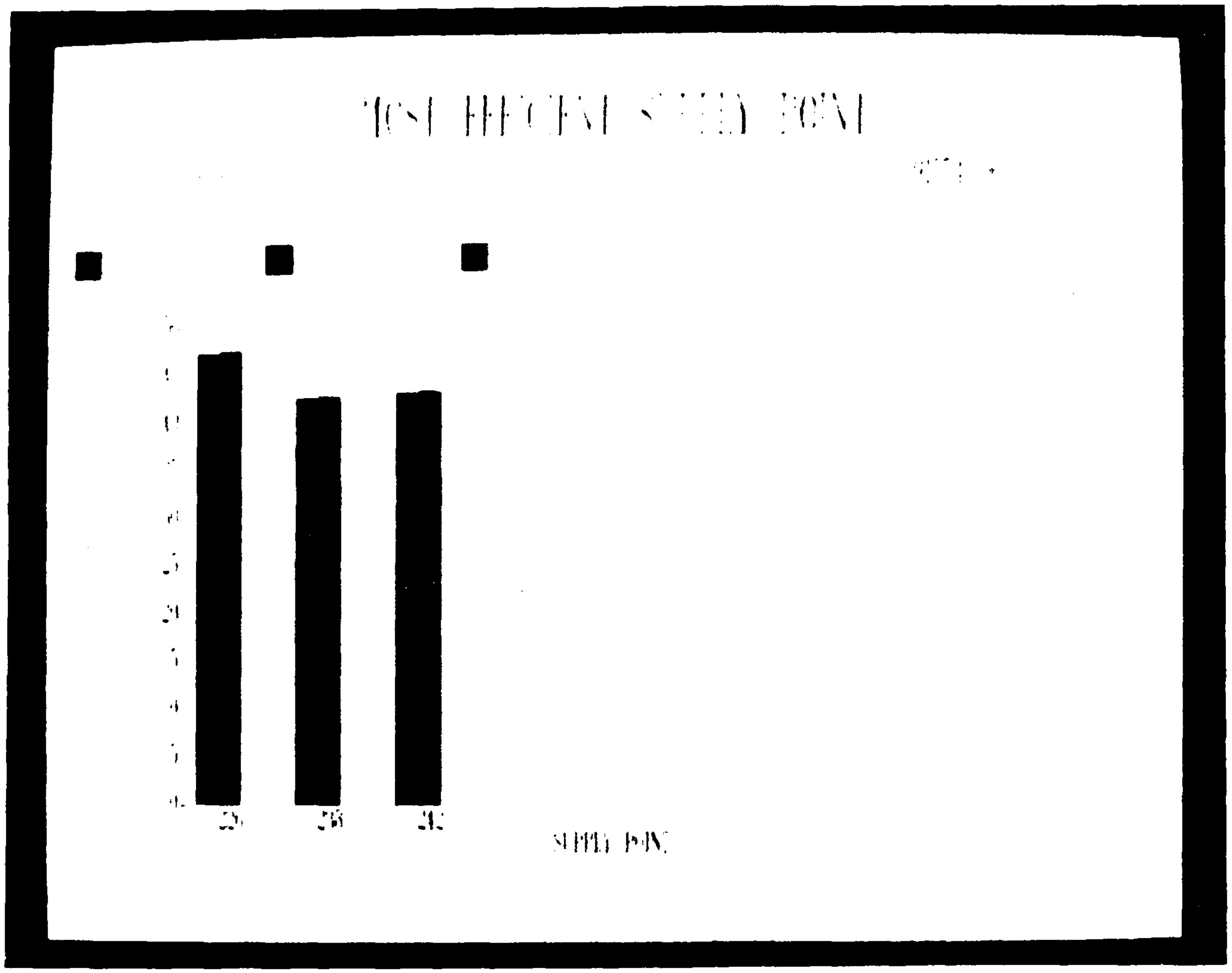

Figure 5.3.5: Most Efficient Supply Point - Division Test Problem

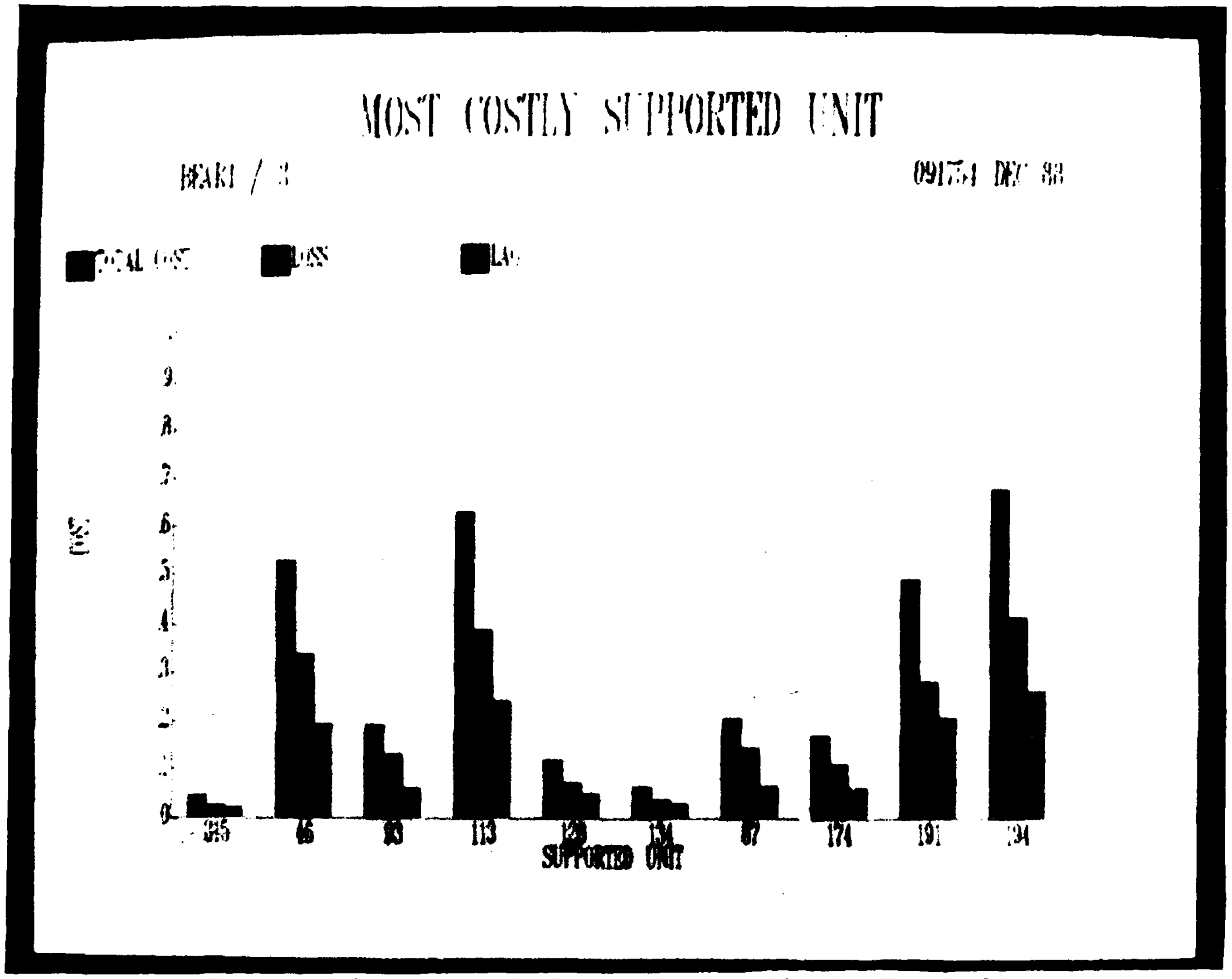

Figure 5.3.6: Most Costly Supported Unit - Division Test Problem

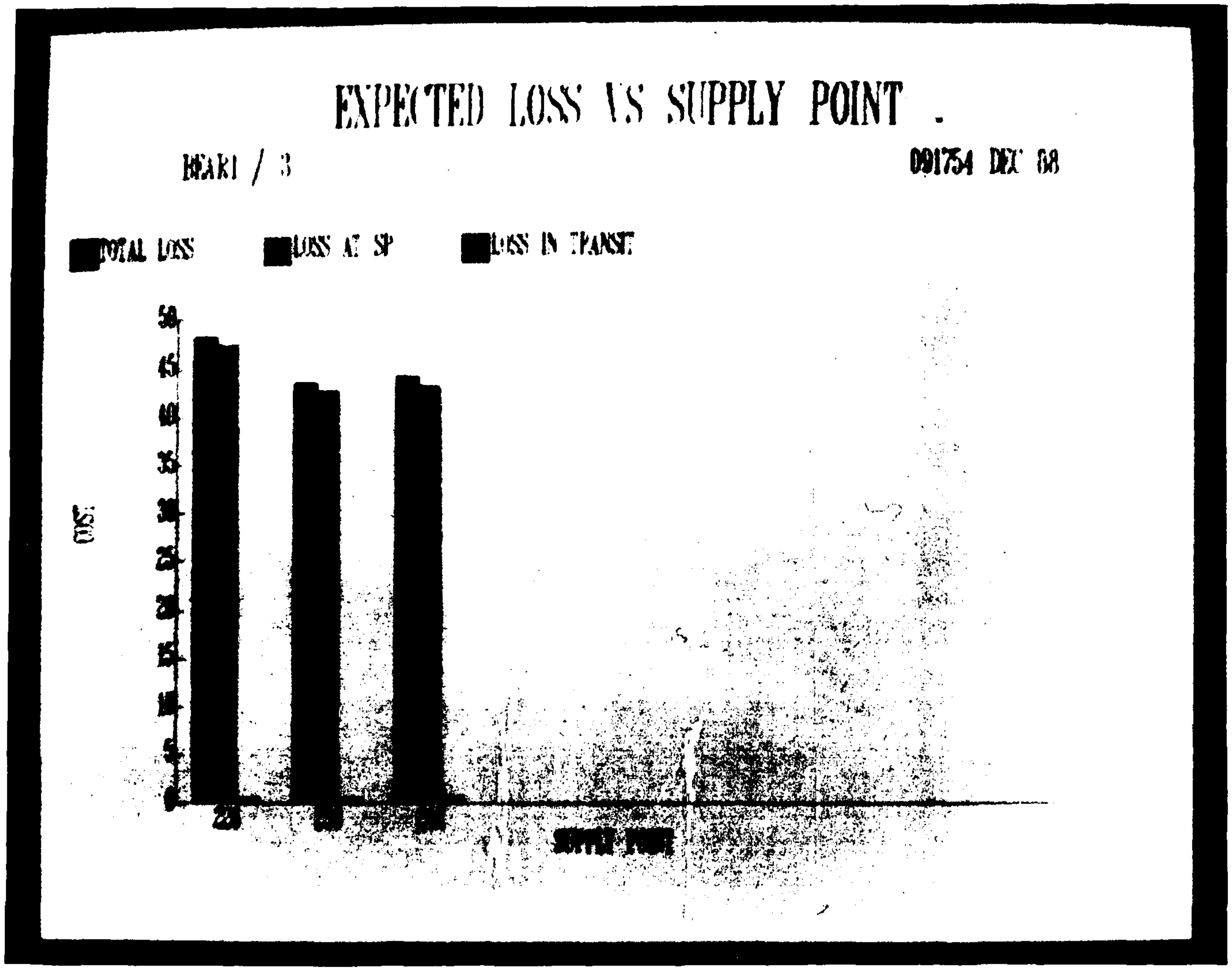

Figure 5.3.7: Expected Loss Report - Division Test Problem

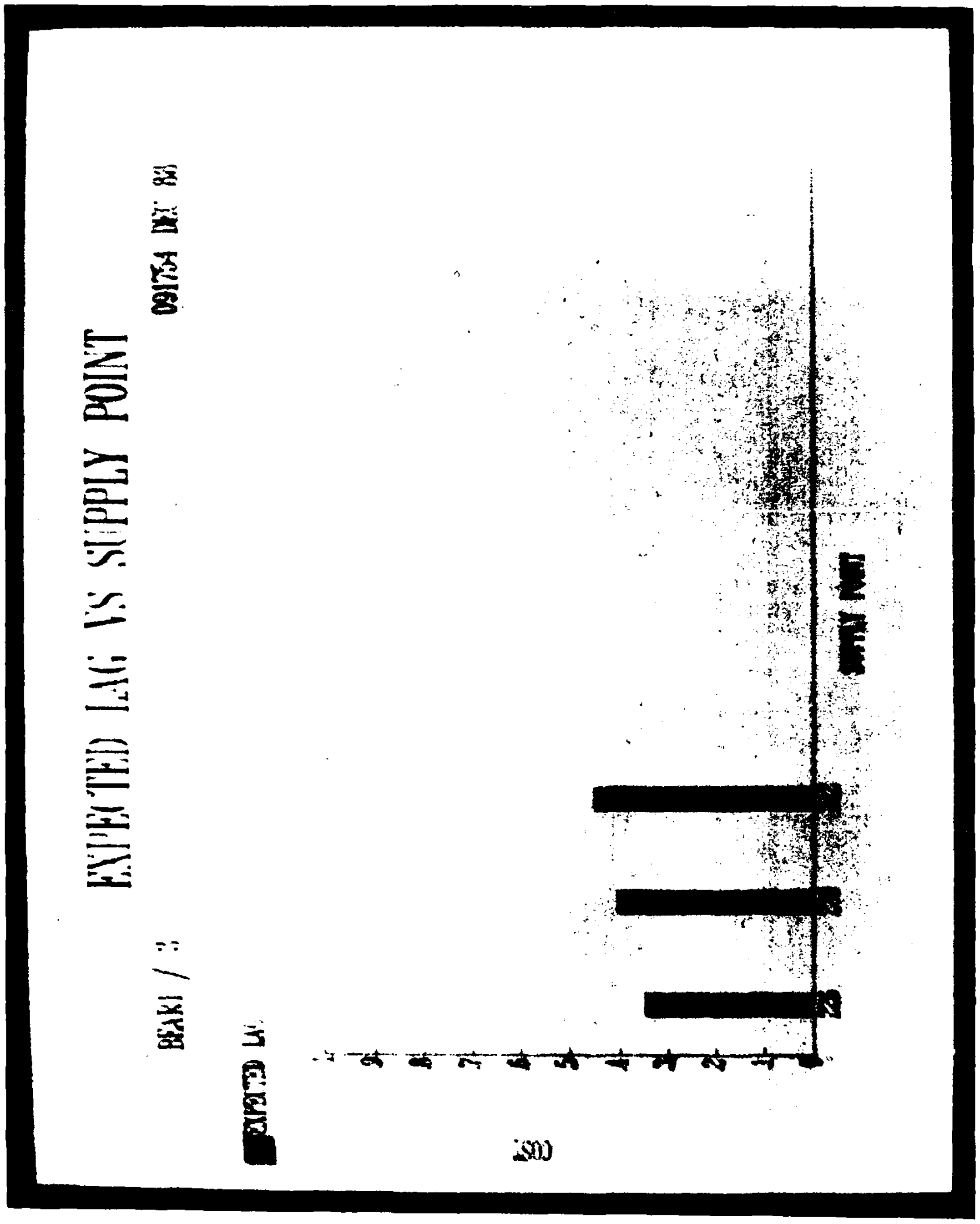

Figure 5.3.8: Expected Lag Report - Division Test Problem

.01, would place a higher value on responsiveness than on losses and should locate supply points farther forward. Because v is much less sensitive to change than w in the cost function, a much larger relative difference must be applied to make lag more important than loss. Essentially v must be set low enough to be practically zero compared with the value for w. Using these values for w and v, a new problem is attempted leaving all other problem data the same. The same sets of open nodes used previously are used in this problem in order to compare the relative solutions. The result for the first set of open nodes is a total cost of 74.01611. The lag component is 71.46718 while loss is only 2.548928. Entering the set of open nodes used in the first problem above which place the supply points as far back in the division rear area as possible (226, 238, and 242) as the next set of open nodes for comparison, a total cost of 118.8352 is obtained (Lag component = 117.4799, Loss component = 1.355291). Note that as the supply points have been "moved back" to a less vulnerable part of the battlefield, even though loss is decreased by a small amount, costs have increased instead of decreased as in the first problem set above because of the higher value of lag. This is due to the adjustment of the relative values for lag and loss as stated, and is the anticipated result.

Many other options are also open to the user at this point if unsatisfied with the results. The user may elect to change the number of supply points used in the problem or reallocate the amount of supply at each open node, for instance. Modifications such as these represent new problem data and must be treated as such, but would contribute significantly to a different solution.

5.3.b The "Brigade" Problem

A test problem for one of the brigades within the division was also completed during testing of the SPL. This problem used a reduced network, and the terrain and road network data base for the brigade problem included only the 3d Brigade sector of the map (64 nodes). The problem used only four demand nodes compared with ten in the "Division" problem above. The solution procedure is exactly the same as discussed above for the division problem. The major difference in the two problems was "solution time" as discussed in Section 5.3.c.

The network display for the Brigade problem has been enhanced using semi-scaling. Figure 5.3.9 illustrates the network before semi-scaling. Note the congested area at the top of the display around node 61. This area is highlighted in Figure 5.3.9. The results of semi-scaling are shown in Figure 5.3.10. The effects of semi-scaling are very subtle,

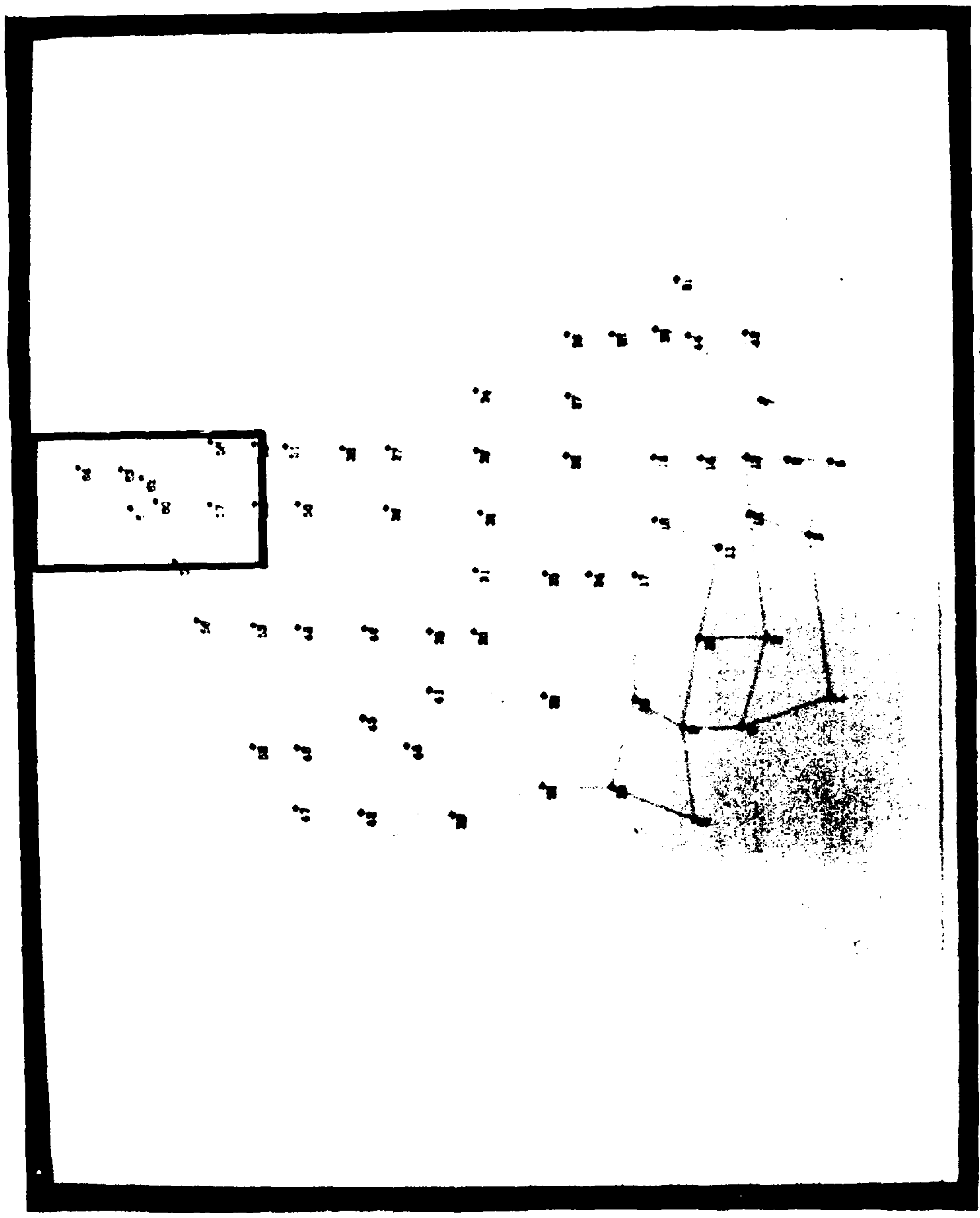

Figure 5.3.9: Network Display Without Semi-scaling

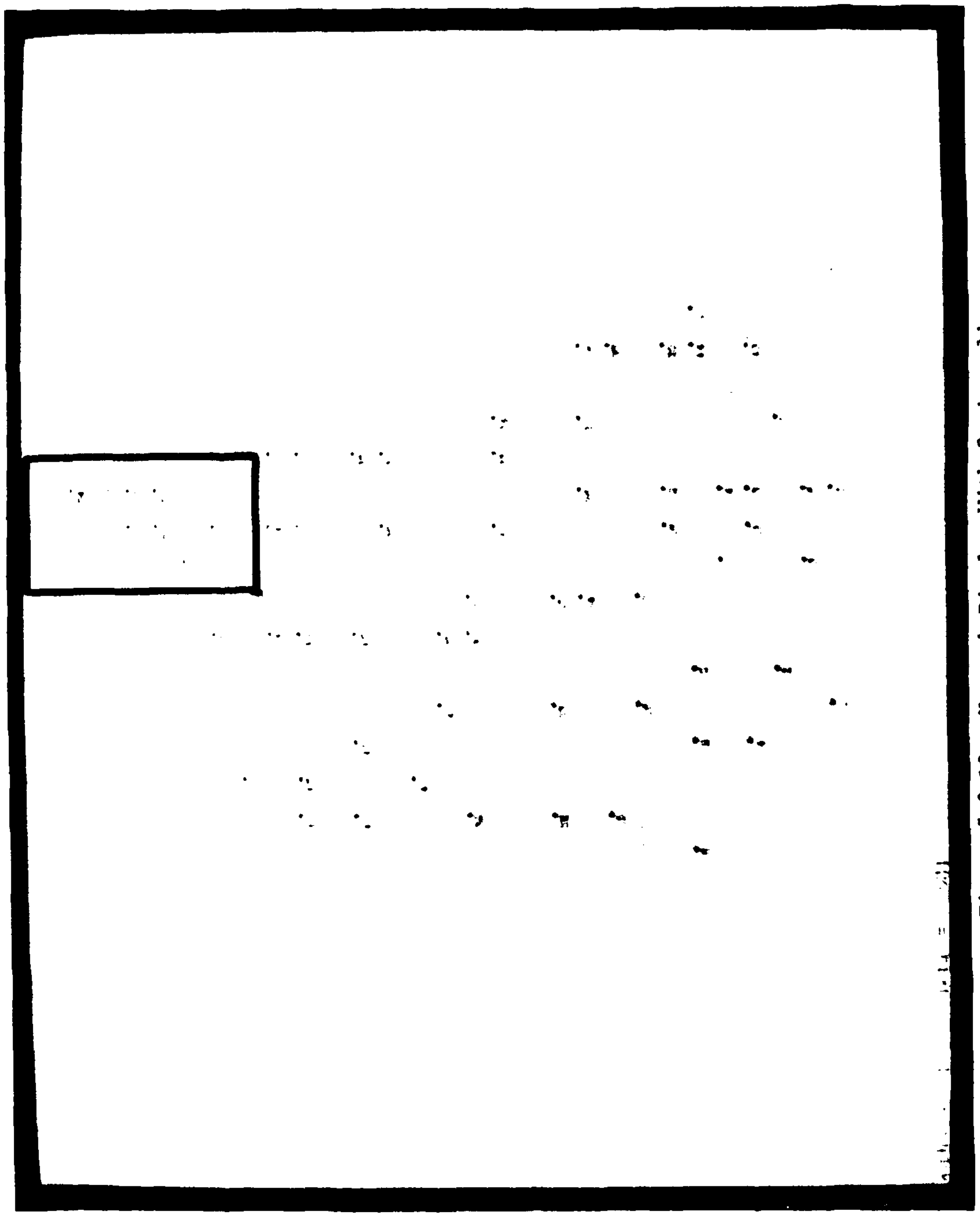

Figure 5.3.10: Network Display With Semi-scaling

but still enhance the display significantly.

During solution of the Brigade problem, the effect of the scaling parameter used in the default vulnerability calculations was examined. Any slight change in this parameter has a very significant effect on total "cost." To test the effect of the vulnerability parameter on the results, a solution was obtained for nodes 52 and 54, near the rear of the brigade rear area (w = .1, v = 1), using the default vulnerability value (.06). Total cost for this problem was 68.55272. A second set of nodes located well forward in the brigade sector was then selected (nodes 30 and 33) and a solution obtained using the same problem data (including the same vulnerability parameter). Total cost of this plan was 110.6674. The vulnerability parameter was then increased slightly (.01) until a solution with approximately the same total cost was obtained. Using a vulnerability parameter of .09, with all other problem data remaining unchanged, a total cost of 70.7784 was obtained for the forward set of open nodes (30 and 33). Figure 5.3.11 shows the forward and rear sets of nodes, circled in red and green respectively. Note that the forward set of nodes is approximately 8 kilometers forward of the rear nodes. The observation is that the vulnerability parameter is quite sensitive to change, as it nullified the difference in total costs over a distance of 8 kilometers (14 kms from the FLOT) with an increase of .03. Because the parameter is negative in the distribution function

$$f(d) = e^{-gd},$$

an increase will have a net effect of decreasing the values of the node and link vulnerabilities. The distribution curve "drops off" very quickly using this value.

At the extreme values for the vulnerability parameter (g), approximately 0 and approximately 1.0, the net effect will be that lag will dominate the objective function computed by the SPL (total cost) and a solution which locates the supply points very far forward (the farthest forward r "suitable" locations). Another way of stating this notion is that at either of the extremes $(0 < g < 1.0)$, the vulnerability is virtually the same for any point on the battlefield, and the cost function is dominated by the lag component. If g is very small (approximately 0) then the vulnerabilities for all locations on the battlefield are very high, and the lag component is the only difference between a location far forward and one far to the rear. At very large g (approximately 1.0) then vulnerabilities throughout the entire battlefield are approximately 0, and the cost function is dominated by the lag function which will result in a supply point location as far forward as possible.

There is some intermediate value of the vulnerability parameter that would locate the supply points farthest to the rear. This value would be very close to the lower extreme. The value for the vulnerability parameter, .06, used as the default is very close to this value and causes the loss component to be dominant in the total cost computations. Such a value would necessarily be between 0 and .06 ($0 < g < .06$). While there is no data to verify this, the default value represents approximate values of vulnerabilities based on current weapon systems, and can be modified as necessary as information becomes available and weapon systems change.

5.3.c Solution Times

The following response times were measured for the 249 node problem using 10 supported units and 3 supply points:

Network data input from data file: 118 seconds

Drawing network on graphics screen: 48 seconds

Input supported unit demand data from data file: 47 seconds

Semi-scaling network (user input and redraw network): 65 sec

User entered CSS data: 50 seconds*

Designate open nodes: 30 seconds*

Computation of solution (SOLVE): 7-9 minutes**

Analysis of results: 1 minute per report*

*User tasks are dependent upon user experience and familiarity with the SPL.

**Solution computation times vary with the number of supply nodes and supported units for any given size problem. The same problem using 4 supply points took 10-11 minutes to solve. Solution computation time for the brigade problem using 84 nodes, 2 supply points, and 4 supported units was approximately 20 seconds.

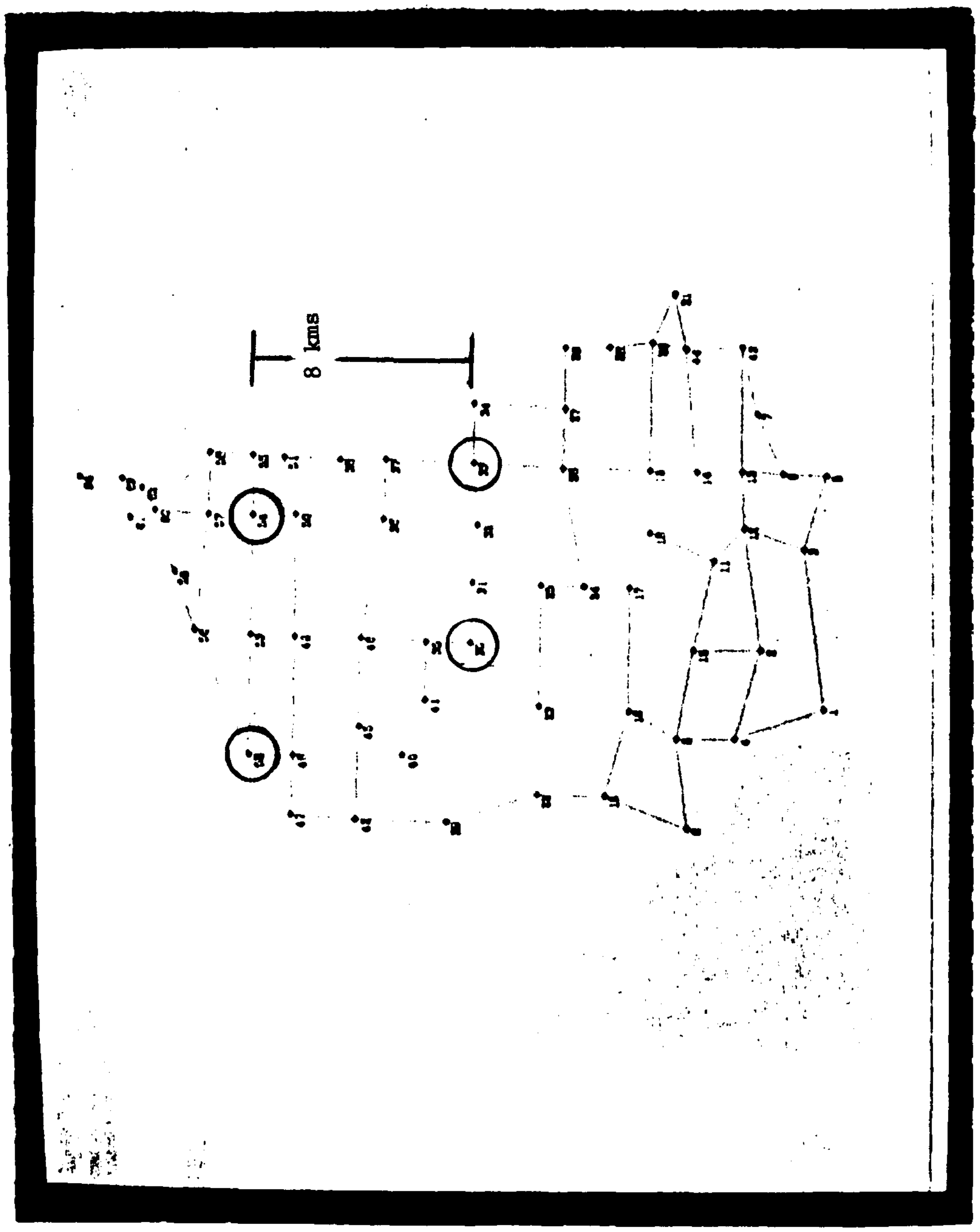

Figure 5.3.11: Effects of Vulnerability Parameter Modification

CHAPTER VI

CONCLUSIONS AND RECOMMENDATIONS

6.1 Results and Conclusions

Experimental results indicate that the location-allocation model is appropriate and the SPL Prototype works as predicted. The results from the experiment reproduce established doctrinal methods of locating supply points. When the relative weights of lag and loss were adjusted, the solution behaved as predicted. An emphasis on loss caused location further to the rear, while an emphasis on lag caused location closer to the front.

The completion and demonstration of the SPL Prototype yields several significant conclusions. These conclusions are essential to the further development of decision support systems used for CSS planning, and in some cases for decision aids used to support operational and tactical decision making. These conclusion, which will be detailed below, are:

(1) There is a need for map and road network data to be issued to the field. This system clarifies that need.

(2) There is a need for terrain and intelligence data to be available in the field, as planned in current battlefield information systems development research. This system provides an example of decision support systems that would require such data.

(3) Semi-scaling is a simple enhancement for roadway networks that enhance viewability without greatly decreasing recognition of patterns.

(4) The Prototype SPL appears suitable as a tool for training and for doctrinal development.

(5) The Prototype SPL provides an example, perhaps the first one in the Army, of the kind of Decision Support System that will become possible when automated situation maps and automated battlefield information become a reality.

The prototype DSS demonstrates many of the data demands for future map and road network data systems, and for a wide range of battlefield information. Although the data required by the SPL Prototype represents only the most fundamental elements of terrain and road network data, the importance and the usefulness of this information has been illustrated. Automated battlefield information will also be essential to tactical decision aids or decision support systems as demonstrated by the data demands of the SPL Prototype.

The first attempt at providing a semi-scaled display has been successfully demonstrated. Although the performance of the semi-scaling technique used by the Prototype SPL is limited by hardware capabilities, the acumen and potential of the semi-scaling concept and theory has been demonstrated.

The Prototype SPL provides quantified corroboration of rear-area location doctrine by reproducing established doctrine from operational data. A successful automated system should support current doctrine while eliminating many of the time consuming manual tasks.

The practical power of interactive network flow optimization as it applies to CSS planning on the battlefield has also been demonstrated by the Prototype SPL.

6.2 Recommendations

The following areas are presented as potential enhancements of the SPL Prototype:

6.2 a Quantitative Procedures

(1) The current method of providing value specifications for lag and loss is somewhat cumbersome and may cause difficulty for the inexperienced user of the SPL. Acceptable values and exact definitions are not easily found or understood. Although the user is provided with a range of recommended values for w and v, further enhancements

could make this much easier. Further testing and analysis of these quantities could lead to a more exact method of qualifying their value. This task could also be made simpler for the user by providing a verbal definition as part of the interface, and then assigning quantitative value based on the user's answer to several prompts such as "How important is it that minimum losses of supplies are sustained from enemy action?" By having the user provide a "degree" of importance (for instance if the user was given 5 choices of degrees of importance) for both loss and lag, then the combination of the values provided by the user could be quantified by the SPL and used accordingly.

(2) Further effectiveness testing of location-allocation solutions is a logical follow-on to this research. An interesting technique would be to obtain an exact solution to a small supply point location problem (approximately 30 - 40 nodes) using a main frame computer with APEX IV or similar linear programming package, while also implementing several heuristic procedures from the literature as part of the SPL Prototype. It would be informative to test the relative solution quality achieved by a human operator using the primary SPL solution technique versus the implemented heuristics, versus the exact solution, with the exact solution serving as the standard.

(3) Further development of the semi-scaling technique would be useful not only for the SPL but as a general graphics tool for controlled distortion of any pictorial object to reduce crowding of detail. The current implementation of semi-scaling disallows the ranking parameter BETA from being so large as to cause ties for cell membership. This keeps the number of cells large and limits the allowable amount of rescaling. A proposed application of the assignment algorithm to break ties for cell membership was discussed in Section 3.2 and would be a useful topic of further research.

6.2.b User Interface

(1) The user interface of the SPL could be improved in several ways. Allowing the user to add/delete nodes and links directly from the graphics screen using the mouse would be a significant improvement. This enhancement is within the hardware capability of the system used by the SPL.

(2) Colors and combinations of colors on both the graphics screen and the text screen could be improved to reduce user fatigue and improve readability.

(3) The capability to superimpose military operational graphics over the network on the graphics screen for the current area of operations as well as the addition of a

military map reference system would aid the user significantly. This would eliminate the frequent need to refer to the mapboard during the solution procedure.

(4) A built-in text editor would significantly enhance the user's ability to provide or modify input/CSS data to the SPL.

(5) A "HELP" routine could be implemented to assist the user throughout the interactive solution and analysis procedures. A more sophisticated system of error tracking would also assist the user and prevent many common user errors.

6.2.b Analysis Procedures

The location-allocation heuristic use by the SPL Prototype of provided inconsistent results. A better choice of heuristic routines to assist the user in selecting open nodes should be available in the SPL. Although the Prototype SPL includes only one such location-allocation heuristic optimization routine, the "ANALYSIS" function is designed to accommodate additional procedures.

APPENDIX I

SPL PROGRAM

The attached program entitled "SPL" accepts all input and makes all computations for the Prototype Supply Point Locator. The program was designed and written by Larry L. Wheeler, CPT, FA, U.S. Army, assisted by Walter White, at Georgia Institute of Technology, Atlanta, Georgia.

```
'*****************************************************************************
'*                                                                           *
'*                            PROGRAM:  SPL                                  *
'*                                                                           *
'*                       (Supply Point Locator)                              *
'*                                                                           *
'*                                 By                                        *
'*                                                                           *
'*                       CPT LARRY L. WHEELER                                *
'*                                                                           *
'*                            assisted by                                    *
'*                                                                           *
'*                        CARL WALTER WHITE JR                               *
'*                                                                           *
'*   Program SPL locates Army supply facilities on a road network using      *
'*   Computer Graphic solutions.  For Documentation see Master's Thesis      *
'*   by Larry L. Wheeler, CPT, USA,  Georgia  Institute  of  Technology,     *
'*   December, 1988.  Professor Donovan Young, Chairman.                     *
'*                                                                           *
'*****************************************************************************

CONST False% = 0, True% = NOT False%
CONST NormalColor% = 2, ErrorColor% = 20
CONST MaxCol% = 57, MaxRow% = 20
CONST ErrorMessRow% = MaxRow% + 1, ErrorMessCol% = 1
CONST PromptRow% = MaxRow% + 2
CONST MainMenu% = 1, SubMenu1% = 2, TransSched% = 3
CONST ShowCursor% = -True%
CONST NoLink = 99999!
CONST CornerFile% = 1, NodeFile% = 2, NodeRND% = 3, DemandFile% = 4
CONST LinkFile% = 4, LinkRND = 5, SupplyFile% = 6, PlanFile% = 7
CONST Background% = 4, NodeColor% = 2, TextColor% = 3, LinkColor% = 1
CONST PathLinkColor% = 5, SourceNodeColor% = 6, DemandNodeColor% = 7
CONST MaxNNodes% = 250
CONST PlanEnd$ = "************************************************************"

TYPE DemandType
  Num AS INTEGER
  Dmnd AS SINGLE
END TYPE

TYPE SupplyType
  Num AS INTEGER
  Sup AS SINGLE
END TYPE

TYPE NodeType
  E AS SINGLE
  N AS SINGLE
  SSE AS SINGLE
  SSN AS SINGLE
  Elev AS SINGLE
  Terr AS STRING * 40
  Desc AS STRING * 40
  Vuln AS SINGLE
  DOS AS INTEGER
  Suitable AS INTEGER
END TYPE

TYPE LinkType
  Head AS INTEGER
  Tail AS INTEGER
  Length AS SINGLE
  RType AS INTEGER
  Vuln AS SINGLE
END TYPE

DECLARE SUB DelLink (NLinks%)
DECLARE SUB DelNode (NNodes%, NodePointer%(), NLinks%, NSinks%, Demand() AS DemandType)
DECLARE SUB AddNode (NNodes%, NodePointer%())
DECLARE SUB LoadDemandFile (NNodes%, NodePointer%(), NSinks%, Demand() AS DemandType)
DECLARE SUB LoadCornerFile (XMin!, YMin!, XMax!, YMax!)
DECLARE SUB NameProblem (Problem AS STRING)
DECLARE SUB Solve (NNodes%, NodePointer%(), WI, VI, Supply() AS SupplyType, Demand() AS DemandType, NSinks%, NSources%, Problem AS S
TRING, Lag!(), Loss!(), DTG$)
DECLARE FUNCTION ShortestPath! (Start%, Stp%, Dist!(), P$)
DECLARE SUB RiskGuide (WI, VI)
DECLARE SUB SemiScale (NNodes%, NLinks%, NodePointer%())
DECLARE SUB InitGraphicsScreen (XMax!, XMin!, YMax!, YMin!)
DECLARE SUB LocAlloc (NSources%, Supply() AS SupplyType, NSinks%, Sink() AS DemandType, WI, VI, FLOTX1!, FLOTY1!, FLOTX2!, FLOTY2!)
DECLARE SUB Transport (CI(), Supply() AS SupplyType, Demand() AS DemandType, NSinks%, NSources%, UnusedSupply!, UnmetDemand!, TotalF
low!, XI())
DECLARE SUB LoadProb (Flag%, Problem AS STRING)
DECLARE SUB Save (Problem AS STRING)
DECLARE SUB CostVsPlan (NPlans%, Problem AS STRING, NSinks%)
DECLARE SUB MostEffSource (NSources%, NSinks%, Supply() AS SupplyType, Lag!(), Loss!(), Problem AS STRING, DTG$, NPlans%)
DECLARE SUB MostCostlyUnit (NSources%, NSinks%, Demand() AS DemandType, Lag!(), Loss!(), Problem AS STRING, DTG$, NPlans%)
DECLARE SUB LossVsSupply (NSources%, NSinks%, Supply() AS SupplyType, Loss!(), Problem AS STRING, DTG$, NPlans%)
DECLARE SUB LagVsSupply (NSources%, NSinks%, Supply() AS SupplyType, Lag!(), Problem AS STRING, DTG$, NPlans%)
DECLARE SUB LoadLinkFile (NLinks%, NNodes%, NodePointer%())
DECLARE SUB DrawLinks (NLinks%)
DECLARE SUB ComputeLinkVul (NLinks%)
DECLARE SUB GetLink (I%, Hd%, T1%, L!, Rt%, V!)
DECLARE SUB PutLink (I%, Hd%, T1%, L!, Rt%, V!)
DECLARE SUB LoadNodeFile (NNodes%, NodePointer%())
DECLARE SUB DrawNodes (NNodes%, NodePointer%(), NSinks%, Demand() AS DemandType, NSources%, Supply() AS SupplyType)
DECLARE SUB PrintNodeNum (NNodes%, NodePointer%(), NSinks%, Demand() AS DemandType)
DECLARE SUB ComputeNodeVul (NNodes%, NodePointer%(), FLOTX1!, FLOTY1!, FLOTX2!, FLOTY2!)
DECLARE SUB EnterDOS (NNodes%, NodePointer%())
DECLARE SUB ModifyNode (NNodes%, NodePointer%(), Demand() AS DemandType, NSinks%)
DECLARE SUB OpenSource (NNodes%, NodePointer%(), NSources%, Supply() AS SupplyType, NSinks%, Sink() AS DemandType)
DECLARE SUB GetNode (I%, East!, North!, SSEast!, SSNorth!, El!, T$, D$, V!, D%, S%)
DECLARE SUB PutNode (NN%, East!, North!, SSEast!, SSNorth!, El!, T$, D$, V!, D%, S%)
DECLARE SUB PrintError (EString$)
DECLARE SUB Waiting (W$, WRow%, WCol%, KeyStroke$)
DECLARE SUB DisplayMessage (Message$)
DECLARE SUB ScrollText (MinC%, MaxC%, MinR%, MaxR%, N%)

OPTION BASE 1
COMMON /MainMenu/ CurrentLine%
COMMON /Corners/ MapXMin, MapYMin, MapXMax, MapYMax
COMMON /FLOT/ FLOTX1, FLOTY1, FLOTX2, FLOTY2
COMMON SHARED /ErrorVar/ ProgramError%
COMMON SHARED /Param/ VulParam

DIM FKey$(16)                           'Keyboard scan codes for the function
                                        'keys F1-F8 and Shift F1- Shift F8.
DIM Speed(5)                            'Speed  in kph for road class 1 to 5
DIM Problem AS STRING * 8
DIM NodePointer%(MaxNNodes%)            'Pointer to location of a node in the
                                        'random access file.
'$DYNAMIC
REDIM Demand(1) AS DemandType           'Demand at demand node J.
NSinks% = 0
REDIM Supply(1) AS SupplyType           'Supply at supply node I,
NSources% = 0
REDIM Lag(1, 1), Loss(1, 1)
VulParam = .06

'Set up error handling subroutine.
```

```
ON ERROR GOTO ErrorHandle
ProgramError% = False%

'Define the function keys.
FOR I% = 1 TO 8
  FKey$(I%) = CHR$(0) + CHR$(58 + I%)
  FKey$(I% + 8) = CHR$(0) + CHR$(83 + I%)
NEXT I%

'Define the speed array. Speed is in kilometers per hour.
Speed(1) = 88! * 24!          'convert to Kms per day
Speed(2) = 65! * 24!
Speed(3) = 56! * 24!
Speed(4) = 48! * 24!
Speed(5) = 44! * 24!

'Initialize the text screen.
KEY OFF
WIDTH 80
COLOR NormalColor%, 0, 0

'Initialize the main menu screen.
SCREEN 0, , MainMenu%, 0
COLOR NormalColor%, 0, 0
CLS

'Draw the menu boxes for the main menu screen.
LOCATE 1, 1: PRINT CHR$(218);
PRINT STRING$(MaxCol% - 2, CHR$(196));
PRINT CHR$(210);
PRINT STRING$(79 - MaxCol%, CHR$(196));
PRINT CHR$(191);
FOR I% = 2 TO MaxRow% - 1
  LOCATE I%, 1: PRINT CHR$(179)
  LOCATE I%, MaxCol%: PRINT CHR$(186)
  LOCATE I%, 80: PRINT CHR$(179)
NEXT I%
LOCATE MaxRow%, 1: PRINT CHR$(192);
PRINT STRING$(MaxCol% - 2, CHR$(196));
PRINT CHR$(208);
PRINT STRING$(79 - MaxCol%, CHR$(196));
PRINT CHR$(217)

'Copy the menu boxes to the screen for submenu 1.
PCOPY MainMenu%, SubMenu1%

'Print menu selections for main menu.
LOCATE 2, 64: PRINT "MAIN MENU"
LOCATE 3, 64: PRINT "---------"
LOCATE 4, 59: PRINT " F1. LOAD NW FILES"
LOCATE 5, 59: PRINT " F2. LOAD PROB FILE"
LOCATE 6, 59: PRINT " F3. SEMI-SCALING"
LOCATE 7, 59: PRINT " F4. ENTER CSS DATA"
LOCATE 8, 59: PRINT " F5. RISK GUIDE"
LOCATE 9, 59: PRINT " F6. MODIFY NETWORK"
LOCATE 10, 59: PRINT " F7. DESIGNATE"
LOCATE 11, 59: PRINT "      OPEN NODES"
LOCATE 12, 59: PRINT " F8. SOLVE"
LOCATE 13, 59: PRINT "SF1. ANALYSIS"
LOCATE 14, 59: PRINT "SF2. PRINT"
LOCATE 15, 59: PRINT "SF3. EXIT"

'Print menu selections for submenu 1.
SCREEN 0, , SubMenu1%, 0
COLOR 3, 0, 0
```

```
LOCATE 2, 62: PRINT "ANALYSIS MENU"
LOCATE 3, 62: PRINT "-------------"
LOCATE 4, 59: PRINT " F1. HEURISTIC     "
LOCATE 5, 59: PRINT " F2. TRANS SCHEDULE"
LOCATE 6, 59: PRINT " F3. PLAN COMPARISON"
LOCATE 7, 59: PRINT " F4. MOST EFFICIENT"
LOCATE 8, 59: PRINT "        SUPPLY POINT"
LOCATE 9, 59: PRINT " F5. MOST COSTLY UNIT"
LOCATE 10, 59: PRINT " F6. LOSS VS SP"
LOCATE 11, 59: PRINT " F7. LAG VS SP"
LOCATE 12, 59: PRINT " F8. MODIFY DATA"
LOCATE 13, 59: PRINT "SF1. Main Menu"

'Initialize the mouse if present and its associated variables.
MousePresent% = False%
CALL MOUSE(MousePresent%, M2%, M3%, M4%)
IF MousePresent% THEN
  CALL MOUSE(7, M2%, MaxCol% * 8, 78 * 8)
  CALL MOUSE(8, M2%, 3 * 8, (MaxRow% - 2) * 8)
  CALL MOUSE(10, 0, &HO, &H7700)
END IF

'Display the mainmenu screen.
SCREEN 0, NormalColor%, 0, 0
COLOR NormalColor%, 0, 0
CLS
PCOPY MainMenu%, 0
CurrentLine% = 2
VIEW PRINT MaxRow% + 1 TO 25

'Enter the main program loop.
Menu% = MainMenu%
ScreenChanged% = False%
Flag% = 0
DO
  'Display the current menu screen.
  IF ScreenChanged% THEN
    PCOPY Menu%, 0
    CurrentLine% = 2
    ScreenChanged% = False%
  END IF

  'Get menu selection.
  DO
    IF MousePresent% THEN CALL MOUSE(ShowCursor%, M2%, M3%, M4%)
    DO
      Selection$ = INKEY$
      IF MousePresent% THEN
        CALL MOUSE(3, Button%, MouseCol%, MouseRow%)
        MouseRow% = MouseRow% \ 8 - 2
        IF Menu% = MainMenu% AND MouseRow% > 7 THEN MouseRow% = MouseRow% - 1
        IF Menu% = SubMenu1% AND MouseRow% > 4 THEN MouseRow% = MouseRow% - 1
        IF (Button% AND 1) OR (Button% AND 2) THEN
          Selection$ = FKey$(MouseRow%)
        END IF
      END IF
    LOOP UNTIL Selection$ <> ""
    Ok% = False%
    I% = 0
    WHILE NOT Ok% AND I% < 10
      I% = I% + 1
      Ok% = (Selection$ = FKey$(I%))
    WEND
    Button% = False%
    IF Selection$ = CHR$(0) + CHR$(25) THEN
      CALL DisplayMessage("Current Vulnerability parameter:  " + STR$(VulParam))
      LOCATE PromptRow%, 1
      LINE INPUT "Enter vulnerability parameter (Return for no change):  ", Lin$
      CLS
      Lin$ = LTRIM$(RTRIM$(Lin$))
      IF LEN(Lin$) > 0 THEN VulParam = VAL(Lin$)
      CALL DisplayMessage("New Vulnerability parameter:  " + STR$(VulParam))
      CALL ComputeNodeVul(NNodes%, NodePointer%(), FLOTX1, FLOTY1, FLOTX2, FLOTY2)
      CALL ComputeLinkVul(NLinks%)
    END IF
  LOOP UNTIL Ok%
  CLS

' Process the menu selection.
  IF Menu% = MainMenu% THEN
    IF Selection$ = FKey$(1) THEN
      NSources% = 0
      NSinks% = 0
      CALL NameProblem(Problem)
      Corners$ = "B:CORNERS.MAP"
      CALL LoadCornerFile(MapXMin, MapYMin, MapXMax, MapYMax)
      CALL LoadNodeFile(NNodes%, NodePointer%())
      CALL LoadLinkFile(NLinks%, NNodes%, NodePointer%())
      CALL InitGraphicsScreen(MapXMax, MapXMin, MapYMax, MapYMin)
      CALL DrawLinks(NLinks%)
      CALL DrawNodes(NNodes%, NodePointer%(), NSinks%, Demand(), NSources%, Supply())
      CALL PrintNodeNum(NNodes%, NodePointer%(), NSinks%, Demand())
      Alpha = 0!
      Beta = 0!
      NewProb% = True%
      Flag% = 1
    ELSEIF Selection$ = FKey$(2) THEN
      NSources% = 0
      CLOSE #PlanFile%
      CALL LoadProb(Flag%, Problem)
      IF Flag% = 4 THEN
        CALL InitGraphicsScreen(MapXMax, MapXMin, MapYMax, MapYMin)
        CALL DrawLinks(NLinks%)
        CALL DrawNodes(NNodes%, NodePointer%(), NSinks%, Demand(), NSources%, Supply())
        CALL PrintNodeNum(NNodes%, NodePointer%(), NSinks%, Demand())
      END IF
      Alpha = 0!
      Beta = 0!
    ELSEIF Selection$ = FKey$(3) THEN
      IF Flag% = 0 THEN
        CALL PrintError("Please select F1 or F2 first.")
      ELSE
        CALL SemiScale(NNodes%, NLinks%, NodePointer%())
        IF GraphicsChanged% THEN
          CALL InitGraphicsScreen(MapXMax, MapXMin, MapYMax, MapYMin)
          GraphicsChanged% = False%
        END IF
        CALLS SETVIEWPORT(0!, 0!, 1!, 1!, Background%, Background%)
        CALLS SETWORLD(MapXMin, MapYMin, MapXMax, MapYMax)
        IF Alpha > 0! THEN
          T$ = "Alpha = " + STR$(Alpha) + "     Beta = " + STR$(Beta)
          STH! = (15! / 768!) * (MapYMax - MapYMin)
          CALLS SETSTEXT(STH!, 1!, 0)
          CALLS MOVTCURABS(MapXMin, MapYMin - (31 / 768!) * (MapYMax - MapYMin))
          CALLS STEXT(T$)
        END IF
        CALLS SETVIEWPORT(0!, 0!, 1!, .98, TextColor%, Background%)
        CALLS SETWORLD(MapXMin, MapYMin, MapXMax, MapYMax)
        CALL DrawLinks(NLinks%)
        CALL DrawNodes(NNodes%, NodePointer%(), NSinks%, Demand(), NSources%, Supply())
        CALL PrintNodeNum(NNodes%, NodePointer%(), NSinks%, Demand())
      END IF
```

```
ELSEIF Selection$ = FKey$(4) THEN
  IF Flag% = 0 THEN
    CALL PrintError("Please select F1 or F2 first.")
  ELSEIF Flag% >= 7 THEN
    CALL NameProblem(Problem)
  END IF
  IF Flag% > 0 THEN
    NSources% = 0
    Dmnd$ = "B:DEMAND.DAT"
    CALL LoadDemandFile(NNodes%, NodePointer%(), NSinks%, Demand())
    IF GraphicsChanged% THEN
      CALL InitGraphicsScreen(MapXMax, MapXMin, MapYMax, MapYMin)
      GraphicsChanged% = False%
    END IF
    CALLS SETVIEWPORT(0!, 0!, 1!, 1!, Background%, Background%)
    CALLS SETWORLD(MapXMin, MapYMin, MapXMax, MapYMax)
    IF Alpha > 0! THEN
      T$ = "Alpha = " + STR$(Alpha) + "     Beta = " + STR$(Beta)
      STH! = (15! / 768!) * (MapYMax - MapYMin)
      CALLS SETSTEXT(STH!, 1!, 0)
      CALLS MOVTCURABS(MapXMin, MapYMin - (3! / 768!) * (MapYMax - MapYMin))
      CALLS STEXT(T$)
    END IF
    CALLS SETVIEWPORT(0!, 0!, 1!, .98, TextColor%, Background%)
    CALLS SETWORLD(MapXMin, MapYMin, MapXMax, MapYMax)
    CALL DrawLinks(NLinks%)
    CALL DrawNodes(NNodes%, NodePointer%(), NSinks%, Demand(), NSources%, Supply())
    CALL PrintNodeNum(NNodes%, NodePointer%(), NSinks%, Demand())
    CALL ComputeNodeVul(NNodes%, NodePointer%(), FLOTX1, FLOTY1, FLOTX2, FLOTY2)
    CALL ComputeLinkVul(NLinks%)
    CALL EnterDOS(NNodes%, NodePointer%())
    NewProb% = True%
    Flag% = 3
  END IF
ELSEIF Selection$ = FKey$(5) THEN
  IF Flag% < 3 THEN
    CALL PrintError("Please select F4 first.")
  ELSEIF Flag% >= 7 THEN
    CALL NameProblem(Problem)
  END IF
  IF Flag% >= 3 THEN
    NSources% = 0
    CALL RiskGuide(W, V)
    NewProb% = True%
    Flag% = 4
  END IF
ELSEIF Selection$ = FKey$(6) THEN
  IF Flag% < 4 THEN
    CALL PrintError("Please select F5 first.")
  ELSEIF Flag% >= 7 THEN
    CALL NameProblem(Problem)
  END IF
  IF Flag% >= 4 THEN
    NSources% = 0
    DO
      DO
        CALL Waiting("Modify Nodes or Links or Stop modify (N/L/S)?", PromptRow%, 1, RR$)
        RR$ = UCASE$(RR$)
      LOOP UNTIL RR$ = "N" OR RR$ = "L" OR RR$ = "S" OR RR$ = CHR$(13)
      CLS
      IF RR$ = CHR$(13) THEN RR$ = "N"
      IF RR$ = "N" THEN
        DO
          CALL Waiting("Modify, Add, or Delete a node (M/A/D)? ", PromptRow%, 1, R$)
          R$ = UCASE$(R$)
        LOOP UNTIL R$ = "M" OR R$ = "A" OR R$ = "D" OR R$ = CHR$(13)
        IF R$ = CHR$(13) THEN R$ = "M"
        IF R$ = "M" THEN
          CALL ModifyNode(NNodes%, NodePointer%(), Demand(), NSinks%)
        ELSEIF R$ = "A" THEN
          CALL AddNode(NNodes%, NodePointer%())
        ELSEIF R$ = "D" THEN
          CALL DelNode(NNodes%, NodePointer%(), NLinks%, NSinks%, Demand())
        END IF
      ELSEIF RR$ = "L" THEN
        DO
          CALL Waiting("Modify, Add, or Delete a link (M/A/D)? ", PromptRow%, 1, R$)
          R$ = UCASE$(R$)
        LOOP UNTIL R$ = "M" OR R$ = "A" OR R$ = "D" OR R$ = CHR$(13)
        IF R$ = CHR$(13) THEN R$ = "M"
        IF R$ = "M" THEN
           CALL ModifyLink(NLinks%, NNodes%, NodePointer%())
        ELSEIF R$ = "A" THEN
           CALL AddLink(NLinks%, NNodes%, NodePointer%())
        ELSEIF R$ = "D" THEN
          CALL DelLink(NLinks%)
        END IF
      END IF
      CLS
    LOOP UNTIL RR$ = "S"
    NewProb% = True%
    GraphicsChanged% = True%
    Alpha = 0!
    Beta = 0!
  END IF
ELSEIF Selection$ = FKey$(7) THEN
  IF Flag% < 4 THEN
    CALL PrintError("Please select F6 first.")
  ELSE
    CALL OpenSource(NNodes%, NodePointer%(), NSources%, Supply(), NSinks%, Demand())
    CALL InitGraphicsScreen(MapXMax, MapXMin, MapYMax, MapYMin)
    GraphicsChanged% = False%
    CALL DrawLinks(NLinks%)
    CALL DrawNodes(NNodes%, NodePointer%(), NSinks%, Demand(), NSources%, Supply())
    CALL PrintNodeNum(NNodes%, NodePointer%(), NSinks%, Demand())
    IF NewProb% THEN
      CLOSE #PlanFile%
      OPEN Problem + ".PLN" FOR OUTPUT AS #PlanFile%
      NewProb% = False%
      NPlans% = 0
    END IF
    Flag% = 6
  END IF
ELSEIF Selection$ = FKey$(8) THEN
  IF Flag% < 6 THEN
    CALL PrintError("Please select F7 first.")
  ELSE
    CALL Solve(NNodes%, NodePointer%(), W, V, Supply(), Demand(), NSinks%, NSources%, Problem, Lag(), Loss(), DTGS)
    ScreenChanged% = True%
    GraphicsChanged% = False%
    Flag% = 7
    CALL Save(Problem)
  END IF
ELSEIF Selection$ = FKey$(9) THEN
  IF Flag% < 7 THEN
    CALL PrintError("Please select F8 first.")
  ELSE
    Menu% = SubMenu1%
    ScreenChanged% = True%
  END IF
ELSEIF Selection$ = FKey$(10) THEN
  IF Flag% < 7 THEN
```

```
      CALL PrintError("Please select F8 first.")
    ELSE
      DO
        CALL Waiting("Output to printer or screen (P/S)?", PromptRow%, 1, Answer$)
        Answer$ = UCASE$(Answer$)
      LOOP UNTIL Answer$ = "P" OR Answer$ = "S"
      IF Answer$ = "P" THEN
        OPEN "LPT1:" FOR OUTPUT AS #98
      ELSE
        OPEN "SCRN:" FOR OUTPUT AS #98
        VIEW PRINT
        ScreenChanged% = True%
        CLS
      END IF
      OPEN "LASTPLAN" FOR INPUT AS #99
      WHILE NOT EOF(99)
        LINE INPUT #99, Lin$
        PRINT #98, Lin$
      WEND
      IF Answer$ = "S" THEN
        CALL Waiting("Press any key to continue.", 24, 0, R$)
        ScreenChanged% = True%
        CLS
        VIEW PRINT MaxRow% + 1 TO 25
      ELSE
        PRINT #98, CHR$(12)            'Page feed.
      END IF
      CLOSE #98
      CLOSE #99
    END IF
  END IF
ELSEIF Menu% = SubMenu1% THEN
  'process submenu selection.
  IF Selection$ = FKey$(1) THEN
    CALL LocAlloc(NSources%, Supply(), NSinks%, Demand(), W, V, FLOTX1, FLOTY1, FLOTX2, FLOTY2)
    ScreenChanged% = True%
  ELSEIF Selection$ = FKey$(2) THEN
    DO
      CALL Waiting("Output to printer or screen (P/S)?", PromptRow%, 1, Answer$)
      Answer$ = UCASE$(Answer$)
    LOOP UNTIL Answer$ = "P" OR Answer$ = "S"
    IF Answer$ = "P" THEN
      OPEN "LPT1:" FOR OUTPUT AS #98
    ELSE
      OPEN "SCRN:" FOR OUTPUT AS #98
      VIEW PRINT
      ScreenChanged% = True%
      CLS
    END IF
    OPEN "LASTPLAN" FOR INPUT AS #99
    WHILE NOT EOF(99)
      LINE INPUT #99, Lin$
      PRINT #98, Lin$
    WEND
    IF Answer$ = "S" THEN
      CALL Waiting("Press any key to continue.", 24, 0, R$)
      ScreenChanged% = True%
      CLS
      VIEW PRINT MaxRow% + 1 TO 25
    ELSE
      PRINT #98, CHR$(12)            'Page feed.
    END IF
    CLOSE #98
    CLOSE #99
  ELSEIF Selection$ = FKey$(3) THEN
    CALL CostVsPlan(NPlans%, Problem, NSinks%)
```

```
      GraphicsChanged% = True%
      CALL Waiting("Press any key to continue.", 24, 0, R$)
    ELSEIF Selection$ = FKey$(4) THEN
      CALL MostEffSource(NSources%, NSinks%, Supply(), Lag(), Loss(), Problem, DTG$, NPlans%)
      GraphicsChanged% = True%
      CALL Waiting("Press any key to continue.", 24, 0, R$)
    ELSEIF Selection$ = FKey$(5) THEN
      CALL MostCostlyUnit(NSources%, NSinks%, Demand(), Lag(), Loss(), Problem, DTG$, NPlans%)
      GraphicsChanged% = True%
      CALL Waiting("Press any key to continue.", 24, 0, R$)
    ELSEIF Selection$ = FKey$(6) THEN
      CALL LossVsSupply(NSources%, NSinks%, Supply(), Loss(), Problem, DTG$, NPlans%)
      GraphicsChanged% = True%
      CALL Waiting("Press any key to continue.", 24, 0, R$)
    ELSEIF Selection$ = FKey$(7) THEN
      CALL LagVsSupply(NSources%, NSinks%, Supply(), Lag(), Problem, DTG$, NPlans%)
      GraphicsChanged% = True%
      CALL Waiting("Press any key to continue.", 24, 0, R$)
    ELSEIF Selection$ = FKey$(8) THEN
      DO
        CALL Waiting("Modify Problem data or Designate new set of open nodes (D/P)? ", PromptRow%, 1, R$)
        R$ = UCASE$(R$)
      LOOP UNTIL R$ = "D" OR R$ = "P" OR R$ = CHR$(13)
      IF R$ = "P" THEN
        CALL DisplayMessage("Go to Main Menu and select MODIFY NETWORK.")
      ELSE
        CALL DisplayMessage("Go to Main Menu and select DESIGNATE OPEN NODES.")
      END IF
    ELSEIF Selection$ = FKey$(9) THEN
      Menu% = MainMenu%
      ScreenChanged% = True%
    END IF
  END IF
  Answer$ = "N"
  IF Menu% = MainMenu% AND Selection$ = FKey$(11) AND NOT ScreenChanged% THEN
    DO
      LOCATE PromptRow%, 1
      INPUT "Do you really want to quit (N/Y)? ", Answer$
      CLS
      IF LEN(Answer$) > 0 THEN
        Answer$ = UCASE$(LEFT$(Answer$, 1))
      ELSE
        Answer$ = "N"
      END IF
    LOOP UNTIL Answer$ = "N" OR Answer$ = "Y"
  END IF
LOOP UNTIL Answer$ = "Y"

END

ErrorHandle:
  ProgramError% = True%
  RESUME NEXT

REM $STATIC
SUB LoadCornerFile (XMin, YMin, XMax, YMax)

SHARED Corners$

  DO
    ProgramError% = False%
    OPEN Corners$ FOR INPUT AS #CornerFile%
    IF ProgramError% THEN
      CLOSE #CornerFile%
      CALL PrintError("ERROR:  Unable to locate file '" + Corners$ + "'.")
      Corners$ = ""
      LOCATE PromptRow%, 1
      INPUT "Enter file name containing corner data:  "; Corners$
      CLS
    END IF
  LOOP UNTIL LEN(Corners$) = 0 OR NOT ProgramError%

  IF ProgramError% THEN
    LOCATE PromptRow%, 1
    INPUT "Enter lower left corner (X,Y):  "; XMin, YMin
    CLS
    LOCATE PromptRow%, 1
    INPUT "Enter upper right corner (X,Y):  "; XMax, YMax
    CLS
    Corners$ = "CORNERS.MAP"
    KILL Corners$
    OPEN Corners$ FOR OUTPUT AS #CornerFile%
    PRINT #CornerFile%, XMax, XMin
    PRINT #CornerFile%, YMax, YMin
  ELSE
    INPUT #CornerFile%, XMax, XMin, YMax, YMin
  END IF
  CLOSE #CornerFile%
  ProgramError% = False%

  'Display corner data in message window.
  CALL DisplayMessage("Minimum easting :  " + STR$(XMin))
  CALL DisplayMessage("Minimum northing:  " + STR$(YMin))
  CALL DisplayMessage("Maximum easting :  " + STR$(XMax))
  CALL DisplayMessage("Maximum northing:  " + STR$(YMax))
  CALL Waiting("Press any key to continue", 24, 0, R$)

  'Convert corners if necessary.
  XMax = XMax - 10000! * (XMax <= XMin)
  YMax = YMax - 10000! * (YMax <= YMin)

END SUB

SUB LoadDemandFile (NNodes%, NodePointer%(), NSinks%, Demand() AS DemandType)

SHARED CurrentLine%
SHARED Dmnd$

  DO
    ProgramError% = False%
    OPEN Dmnd$ FOR INPUT AS #DemandFile%
    IF ProgramError% THEN
      CLOSE #DemandFile%
      CALL PrintError("ERROR:  Unable to locate file '" + Dmnd$ + "'.")
      Dmnd$ = ""
      INPUT "Enter file name containing demand data:  "; Dmnd$
      CLS
    END IF
  LOOP UNTIL LEN(Dmnd$) = 0 OR NOT ProgramError%

  IF ProgramError% THEN
    DO
      LOCATE PromptRow%, 1
      INPUT "Enter the number of demand nodes:  "; NSinks%
      IF NSinks% < 0 THEN CALL PrintError("Negative values are not allowed here.")
      IF NSinks% > NNodes% - 1 THEN CALL PrintError("Value too large.")
    LOOP UNTIL NSinks% > 0 AND NSinks% < NNodes%
    CLS

    KILL "DEMAND.DAT"
    OPEN "DEMAND.DAT" FOR OUTPUT AS #DemandFile%
    PRINT #DemandFile%, NSinks%
  REDIM Demand(NSinks%) AS DemandType

  I% = 1
  WHILE I% <= NSinks%
    DO
      LOCATE PromptRow%, 1
      PRINT "Enter the node no. for demand node"; I%; ":  ";
      INPUT Demand(I%).Num
      Exists% = False%
      LookAt% = 1
      WHILE LookAt% <= NNodes% AND NOT Exists%
        Exists% = NodePointer%(LookAt%) = Demand(I%).Num
        LookAt% = LookAt% + 1
      WEND
      Found% = False%
      LookAt% = 1
      WHILE LookAt% < I% AND NOT Found%
        Found% = (Demand(LookAt%).Num = Demand(I%).Num)
        LookAt% = LookAt% + 1
      WEND
      IF NOT Exists% THEN CALL PrintError("That node does not exist.")
      IF Found% THEN CALL PrintError("That node is already entered.")
    LOOP UNTIL Exists% AND NOT Found%
    CLS

    DO
      LOCATE PromptRow%, 1
      PRINT "Enter the demand at node no."; Demand(I%).Num; ":  ";
      INPUT Demand(I%).Dmnd
      CLS
      IF Demand(I%).Dmnd < 0! THEN CALL PrintError("Demand must be positive.")
    LOOP UNTIL Demand(I%).Dmnd >= 0!

    CALL DisplayMessage("Demand node no.:   " + STR$(Demand(I%).Num))
    CALL DisplayMessage("   Demand:   " + STR$(Demand(I%).Dmnd))

    Answer$ = ""
    DO
      LOCATE PromptRow%, 1
      INPUT "Do you want to make changes (Y/N)?"; Answer$
      CLS
      IF LEN(Answer$) > 0 THEN
        Answer$ = UCASE$(LEFT$(Answer$, 1))
      ELSE
        Answer$ = "Y"
      END IF
    LOOP UNTIL Answer$ = "Y" OR Answer$ = "N"

    IF Answer$ = "N" THEN
      PRINT #DemandFile%, Demand(I%).Num, Demand(I%).Dmnd
      I% = I% + 1
    END IF
  WEND
  ProgramError% = False%
ELSE
  INPUT #DemandFile%, NSinks%
  REDIM Demand(NSinks%) AS DemandType

  R$ = ""
  FOR I% = 1 TO NSinks%
    INPUT #DemandFile%, Demand(I%).Num, Demand(I%).Dmnd
    CALL DisplayMessage("Demand node no.:   " + STR$(Demand(I%).Num))
    CALL DisplayMessage("   Demand:   " + STR$(Demand(I%).Dmnd))
    IF CurrentLine% = MaxRow% AND R$ <> "Q" AND I% <> NSinks% THEN
      CALL Waiting("Press any key to continue or 'Q' to continue without waiting.", 24, 0, R$)
      CALL ScrollText(2, MaxCol% - 1, 2, MaxRow% - 1, MaxRow% - 2)
      CurrentLine% = 2
    END IF
  NEXT I%
  CALL Waiting("Press any key to continue.", 24, 0, R$)
 END IF
 CLOSE #DemandFile%

END SUB

SUB LoadLinkFile (NLinks%, NNodes%, NodePointer%())

SHARED CurrentLine%
DIM LTemp AS LinkType
DIM Dist(NNodes%, NNodes%)

 FOR I% = 1 TO NNodes%
   FOR J% = 1 TO NNodes%
     Dist(I%, J%) = NoLink
   NEXT J%
 NEXT I%

 Links$ = "B:LINK.DAT"
 DO
   ProgramError% = False%
   OPEN Links$ FOR INPUT AS #LinkFile%
   IF ProgramError% THEN
     CLOSE #LinkFile%
     CALL PrintError("ERROR:  Unable to locate file '" + Links$ + "'.")
     Links$ = ""
     LOCATE PromptRow%, 1
     INPUT "Enter file name containing link data:  ", Links$
     CLS
   END IF
 LOOP UNTIL LEN(Links$) = 0 OR NOT ProgramError%

 IF ProgramError% THEN
   LOCATE PromptRow%, 1
   INPUT "Enter the number of links:  ", NLinks%
   CLS

   Links$ = "LINK.DAT"
   KILL Links$
   OPEN Links$ FOR OUTPUT AS #LinkFile%
   PRINT #LinkFile%, NLinks%
   CLOSE #LinkRND%
   KILL "LINK.RND"
   OPEN "LINK.RND" FOR RANDOM AS #LinkRND% LEN = LEN(LTemp)

   I% = 1
   WHILE I% <= NLinks%
     DO
       DO
         LOCATE PromptRow%, 1
         INPUT "Enter node number for head:  ", LTemp.Head
         CLS
         Found% = False%
         A% = 1
         WHILE A% <= NNodes% AND NOT Found%
           Found% = LTemp.Head = NodePointer%(A%)
           IF NOT Found% THEN A% = A% + 1
         WEND
         IF NOT Found% THEN CALL PrintError("That node does not exist.")
       LOOP UNTIL Found%

       DO
         LOCATE PromptRow%, 1
```

```
        INPUT "Enter node number for tail:  ", LTemp.Tail
        CLS
        Found% = False%
        B% = 1
        WHILE B% <= NNodes% AND NOT Found%
          Found% = LTemp.Tail = NodePointer%(B%)
          IF NOT Found% THEN B% = B% + 1
        WEND
        IF NOT Found% THEN CALL PrintError("That node does not exist.")
      LOOP UNTIL Found%

      Found% = Dist(A%, B%) <> NoLink
    LOOP UNTIL NOT Found%

    CALL DisplayMessage("Node no.:  " + STR$(I%))
    CALL DisplayMessage("   Head :  " + STR$(LTemp.Head))
    CALL DisplayMessage("   Tail :  " + STR$(LTemp.Tail))

    DO
      LOCATE PromptRow%, 1
      INPUT "Enter the length of the link:  ", LTemp.Length
      CLS
      IF LTemp.Length <= 0! THEN CALL PrintError("Length must be greater than zero.")
    LOOP UNTIL LTemp.Length > 0!
    Dist(A%, B%) = LTemp.Length
    Dist(B%, A%) = LTemp.Length
    CALL DisplayMessage("   Length:  " + STR$(LTemp.Length))

    DO
      LOCATE PromptRow%, 1
      INPUT "Enter the road class of the link:  ", LTemp.RType
      CLS
      IF LTemp.RType < 1 OR LTemp.RType > 5 THEN
        CALL PrintError("Road class must be in the range of 1 to 5.")
      END IF
    LOOP UNTIL LTemp.RType >= 1 AND LTemp.RType <= 5
    CALL DisplayMessage("  Class:  " + STR$(LTemp.RType))

    DO
      LOCATE PromptRow%, 1
      INPUT "Do you want to make any changes (Y/N)"; Answer$
      CLS
      IF LEN(Answer$) > 0 THEN
        Answer$ = UCASE$(LEFT$(Answer$, 1))
      ELSE
        Answer$ = "Y"
      END IF
    LOOP UNTIL Answer$ = "Y" OR Answer$ = "N"

    IF Answer$ = "N" THEN
      PRINT #LinkFile%, I%, LTemp.Head, LTemp.Tail, LTemp.Length, LTemp.RType
      CALL PutLink(I%, LTemp.Head, LTemp.Tail, LTemp.Length, LTemp.RType, 0!)
      I% = I% + 1
    END IF
  WEND
  ProgramError% = False%
ELSE
  INPUT #LinkFile, NLinks%
  R$ = ""
  CLOSE #LinkRND%
  KILL "LINK.RND"
  OPEN "LINK.RND" FOR RANDOM AS #LinkRND% LEN = LEN(LTemp)

  N% = NLinks%
  FOR I% = 1 TO NLinks%
    INPUT #LinkFile, LNum%
```

```
      INPUT #LinkFile, LTemp.Head, LTemp.Tail
      INPUT #LinkFile, LTemp.Length
      INPUT #LinkFile, LTemp.RType
      CALL DisplayMessage("Link no.:  " + STR$(LNum%))
      CALL DisplayMessage("   Head      :  " + STR$(LTemp.Head))
      CALL DisplayMessage("   Tail      :  " + STR$(LTemp.Tail))
      CALL DisplayMessage("   Length    :  " + STR$(LTemp.Length))
      CALL DisplayMessage("   Road class:  " + STR$(LTemp.RType))
      A% = 0
      B% = 0
      FOR J% = 1 TO NNodes%
        IF NodePointer%(J%) = LTemp.Head THEN A% = J%
        IF NodePointer%(J%) = LTemp.Tail THEN B% = J%
      NEXT J%
      IF A% = 0 OR B% = 0 THEN
        CALL PrintError("WARNING:  Undefined node in data file " + Links$)
        CALL Waiting("Press any key to continue.", 24, 0, R$)
        N% = N% - 1
      ELSEIF Dist(A%, B%) <> NoLink THEN
        CALL PrintError("WARNING:  Repeated link in data file " + Links$)
        CALL Waiting("Press any key to continue.", 24, 0, R$)
        N% = N% - 1
      ELSE
        Dist(A%, B%) = LTemp.Length
        Dist(B%, A%) = LTemp.Length
        CALL PutLink(I%, LTemp.Head, LTemp.Tail, LTemp.Length, LTemp.RType, 0!)
      END IF
      IF CurrentLine% = MaxRow% AND I% <> NLinks% THEN
        IF R$ <> "Q" THEN CALL Waiting("Press any key to continue or 'Q' to continue without waiting.", 24, 0, R$)
        CALL ScrollText(2, MaxCol% - 1, 2, MaxRow% - 1, MaxRow% - 2)
        CurrentLine% = 2
      END IF
    NEXT I%
    NLinks% = N%
    CALL Waiting("Press any key to continue.", 24, 0, R$)
  END IF
  CLOSE #LinkFile%

END SUB

SUB LoadNodeFile (NNodes%, NodePointer%())

DIM NTemp AS NodeType
SHARED CurrentLine%
SHARED MapXMin, MapYMin, MapXMax, MapYMax

  Nodes$ = "B:NODE.DAT"
  DO
    ProgramError% = False%
    OPEN Nodes$ FOR INPUT AS #NodeFile%
    IF ProgramError% THEN
      CLOSE #NodeFile%
      CALL PrintError("ERROR:  Unable to locate file '" + Nodes$ + "'.")
      Nodes$ = ""
      LOCATE PromptRow%, 1
      INPUT "Enter file name containing node data:  "; Nodes$
      CLS
    END IF
  LOOP UNTIL LEN(Nodes$) = 0 OR NOT ProgramError%

  IF ProgramError% THEN
    'User input node data from keyboard
    'data is saved to file "NODE.DAT"
    DO
      LOCATE PromptRow%, 1
      INPUT "Enter the number of nodes in the network:  ", NNodes%
  CLS
  IF NNodes% <= 0 OR NNodes% > MaxNNodes% THEN
    CALL PrintError("Value must be between 1 and " + STR$(MaxNNodes%) + ".")
  END IF
LOOP UNTIL NNodes% > 0 AND NNodes% <= MaxNNodes%

'Initialize arrays and files associated with node data.
Nodes$ = "NODE.DAT"
KILL Nodes$
OPEN Nodes$ FOR OUTPUT AS #NodeFile%
PRINT #NodeFile%, NNodes%
CLOSE #NodeRND%
KILL "NODE.RND"
OPEN "NODE.RND" FOR RANDOM AS #NodeRND% LEN = LEN(NTemp)

I% = 1
WHILE I% <= NNodes%
  DO
    LOCATE PromptRow%, 1
    INPUT "Enter the node number:  ", NodePointer%(I%)
    CLS
    Found% = False%
    LookAt% = 1
    WHILE LookAt% < I% AND NOT Found%
      Found% = NodePointer%(LookAt%) = NodePointer%(I%)
      LookAt% = LookAt% + 1
    WEND
    IF Found% THEN CALL PrintError("That node number is already in use.")
  LOOP UNTIL NOT Found%

  DO
    LOCATE PromptRow%, 1
    PRINT "Enter the Easting and Northing for node"; NodePointer%(I%); ":  ";
    INPUT NTemp.E, NTemp.N
    CLS
    Found% = False%
    LookAt% = 1
    WHILE LookAt% < I% AND NOT Found%
      CALL GetNode(NodePointer%(LookAt%), E1, N1, SSE1, SSN1, E1, T$, D$, V, D%, S%)
      IF NTemp.N = N1 THEN Found% = (NTemp.E = E1)
      LookAt% = LookAt% + 1
    WEND
    IF Found% THEN CALL PrintError("There is already a node at that location.")
    Ok% = MapXMin <= NTemp.E AND NTemp.E <= MapXMax
    Ok% = Ok% AND MapYMin <= NTemp.N AND NTemp.N <= MapYMax
    IF NOT Ok% AND NOT Found% THEN CALL PrintError("That location is out of bounds.")
  LOOP UNTIL Ok% AND NOT Found%
  NTemp.SSE = NTemp.E - 10000! * (NTemp.E < MapXMin)
  NTemp.SSN = NTemp.N - 10000! * (NTemp.N < MapYMin)

  CALL DisplayMessage("Node no.:  " + STR$(NodePointer%(I%)))
  CALL DisplayMessage("   Easting    :  " + STR$(NTemp.E))
  CALL DisplayMessage("   Northing   :  " + STR$(NTemp.N))

  LOCATE PromptRow%, 1
  INPUT "Enter the elevation above sea level:  "; NTemp.Elev
  CLS
  CALL DisplayMessage("   Elevation  :  " + STR$(NTemp.Elev))

  LOCATE PromptRow%, 1
  INPUT "Enter a brief description of the terrain (up to 40 char.):  ", NTemp.Terr
  CLS
  CALL DisplayMessage("   Terrain    :  " + NTemp.Terr)

  LOCATE PromptRow%, 1
  INPUT "Enter a brief description of the area (up to 40 char.):  ", NTemp.Desc
```

```
      CLS
      CALL DisplayMessage("   Description:  " + NTemp.Desc)

      DO
        LOCATE PromptRow%, 1
        INPUT "Enter node suitability parameter (0 for unsuitable, -1 for suitable):  ", S%
        CLS
      LOOP UNTIL S% = -1 OR S% = 0
      CALL DisplayMessage("   Suitability:  " + STR$(S%))

      Answer$ = ""
      DO
        LOCATE PromptRow%, 1
        PRINT "Make any changes to node"; NodePointer%(I%); " (Y/N)";
        INPUT Answer$
        CLS
        IF LEN(Answer$) > 0 THEN
          Answer$ = UCASE$(LEFT$(Answer$, 1))
        ELSE
          Answer$ = "Y"
        END IF
      LOOP UNTIL Answer$ = "Y" OR Answer$ = "N"

      IF Answer$ = "N" THEN
        PRINT #NodeFile%, NodePointer%(I%), NTemp.E, NTemp.N, NTemp.Elev, NTemp.Terr, NTemp.Desc, S%
        CALL PutNode(NodePointer%(I%), NTemp.E, NTemp.N, NTemp.SSE, NTemp.SSN, NTemp.Elev, NTemp.Terr, NTemp.Desc, 01, 0, S%)
        I% = I% + 1
      END IF
    WEND
    ProgramError% = False%
  ELSE
    R$ = ""
    IF NOT EOF(NodeFile%) THEN
      INPUT #NodeFile%, NNodes%
      CLOSE #NodeRND%
      KILL "NODE.RND"
      OPEN "NODE.RND" FOR RANDOM AS #NodeRND% LEN = LEN(NTemp)
    END IF

    FOR I% = 1 TO NNodes%
      INPUT #NodeFile%, NodePointer%(I%), NTemp.E, NTemp.N, NTemp.Elev, T$, D$, S%
      NTemp.Terr = T$
      NTemp.Desc = D$
      CALL DisplayMessage("Node no.:  " + STR$(NodePointer%(I%)))
      CALL DisplayMessage("   Easting    :  " + STR$(NTemp.E))
      CALL DisplayMessage("   Northing   :  " + STR$(NTemp.N))
      CALL DisplayMessage("   Elevation  :  " + STR$(NTemp.Elev))
      CALL DisplayMessage("   Terrain    :  " + NTemp.Terr)
      CALL DisplayMessage("   Description:  " + NTemp.Desc)
      CALL DisplayMessage("   Suitability:  " + STR$(S%))
      NTemp.SSE = NTemp.E - 100001 * (NTemp.E < MapXMin)
      NTemp.SSN = NTemp.N - 100001 * (NTemp.N < MapYMin)
      IF CurrentLine% = MaxRow% THEN
        IF R$ <> "Q" THEN CALL Waiting("Press any key to continue or 'Q' to continue without waiting.", 24, 0, R$)
        CALL ScrollText(2, MaxCol% - 1, 2, MaxRow% - 1, MaxRow% - 2)
        CurrentLine% = 2
      END IF
      CALL PutNode(NodePointer%(I%), NTemp.E, NTemp.N, NTemp.SSE, NTemp.SSN, NTemp.Elev, NTemp.Terr, NTemp.Desc, 01, 0, S%)
    NEXT I%
    CALL Waiting("Press any key to continue.", 24, 0, R$)
  END IF
  CLOSE #NodeFile%

END SUB

SUB LoadProb (Flag%, Problem AS STRING)
```

```
SHARED CurrentLine%, W, V, NPlane%
SHARED MapXMax, MapXMin, MapYMax, MapYMin
SHARED FLOTX1, FLOTY1, FLOTX2, FLOTY2
SHARED NNodes%, NodePointer%(), NLinks%
SHARED NSinks%, Demand() AS DemandType
DIM NTemp AS NodeType
DIM LTemp AS LinkType

  'Get the problem to be loaded.
  Answer$ = "N"
  DO
    IF Answer$ = "N" THEN CALL NameProblem(Problem)
    ProgramError% = False%
    OPEN Problem + ".DAT" FOR INPUT AS #99
    OPEN Problem + ".PLN" FOR APPEND AS #PlanFile%
    IF ProgramError% THEN
      CLOSE #99
      CALL PrintError("ERROR:  Unable to locate " + Problem + " files.")
      CALL Waiting("(A)bort, (R)etry or (N)ew name ", PromptRow%, 1, Answer$)
      Answer$ = UCASE$(Answer$)
    END IF
  LOOP UNTIL Answer$ = "A" OR NOT ProgramError%
  IF Answer$ = "A" THEN
    CLOSE #99
    CLOSE #PlanFile%
    ProgramError% = False%
  END IF
  IF Answer$ = "A" THEN EXIT SUB

  'Get the corners of the map.
  INPUT #99, MapXMax
  INPUT #99, MapXMin
  INPUT #99, MapYMax
  INPUT #99, MapYMin
  CALL DisplayMessage("Minimum easting :  " + STR$(MapXMin))
  CALL DisplayMessage("Minimum northing:  " + STR$(MapYMin))
  CALL DisplayMessage("Maximum easting :  " + STR$(MapXMax))
  CALL DisplayMessage("Maximum northing:  " + STR$(MapYMax))
  CALL Waiting("Press any key to continue", 24, 0, R$)

  'Get the node data.
  R$ = ""
  INPUT #99, NNodes%
  CLOSE #NodeRND%
  KILL "NODE.RND"
  OPEN "NODE.RND" FOR RANDOM AS #NodeRND% LEN = LEN(NTemp)
  FOR I% = 1 TO NNodes%
    INPUT #99, NodePointer%(I%)
    INPUT #99, NTemp.E
    INPUT #99, NTemp.N
    INPUT #99, NTemp.Elev
    INPUT #99, NTemp.Terr
    INPUT #99, NTemp.Desc
    INPUT #99, NTemp.Vuln
    INPUT #99, NTemp.DOS
    INPUT #99, NTemp.Suitable
    CALL DisplayMessage("Node no.:  " + STR$(NodePointer%(I%)))
    CALL DisplayMessage("   Easting    :  " + STR$(NTemp.E))
    CALL DisplayMessage("   Northing   :  " + STR$(NTemp.N))
    CALL DisplayMessage("   Elevation  :  " + STR$(NTemp.Elev))
    CALL DisplayMessage("   Terrain    :  " + NTemp.Terr)
    CALL DisplayMessage("   Description:  " + NTemp.Desc)
    CALL DisplayMessage("   Suitability:  " + STR$(NTemp.Suitable))
    NTemp.SSE = NTemp.E - 10000! * (NTemp.E < MapXMin)
    NTemp.SSN = NTemp.N - 10000! * (NTemp.N < MapYMin)
  CALL PutNode(NodePointer%(I%), NTemp.E, NTemp.N, NTemp.SSE, NTemp.SSN, NTemp.Elev, NTemp.Terr, NTemp.Desc, NTemp.Vuln, NTemp.DOS
, NTemp.Suitable)
  IF CurrentLine% = MaxRow% AND I% <> NNodes% THEN
    IF R$ <> "Q" THEN CALL Waiting("Press any key to continue or 'Q' to continue without waiting.", 24, 0, R$)
    CALL ScrollText(2, MaxCol% - 1, 2, MaxRow% - 1, MaxRow% - 2)
    CurrentLine% = 2
  END IF
NEXT I%
CALL Waiting("Press any key to continue.", 24, 0, R$)

'Get the link data.
INPUT #99, NLinks%
R$ = ""
CLOSE #LinkRND%
KILL "LINK.RND"
OPEN "LINK.RND" FOR RANDOM AS #LinkRND% LEN = LEN(LTemp)
FOR I% = 1 TO NLinks%
  INPUT #99, LTemp.Head
  INPUT #99, LTemp.Tail
  INPUT #99, LTemp.Length
  INPUT #99, LTemp.RType
  INPUT #99, LTemp.Vuln
  CALL DisplayMessage("Link no.:  " + STR$(I%))
  CALL DisplayMessage("   Head      :  " + STR$(LTemp.Head))
  CALL DisplayMessage("   Tail      :  " + STR$(LTemp.Tail))
  CALL DisplayMessage("   Length    :  " + STR$(LTemp.Length))
  CALL DisplayMessage("   Road class:  " + STR$(LTemp.RType))
  CALL PutLink(I%, LTemp.Head, LTemp.Tail, LTemp.Length, LTemp.RType, LTemp.Vuln)
  IF CurrentLine% = MaxRow% AND I% <> NLinks% THEN
    IF R$ <> "Q" THEN CALL Waiting("Press any key to continue or 'Q' to continue without waiting.", 24, 0, R$)
    CALL ScrollText(2, MaxCol% - 1, 2, MaxRow% - 1, MaxRow% - 2)
    CurrentLine% = 2
  END IF
NEXT I%
CALL Waiting("Press any key to continue.", 24, 0, R$)

'Get the demand data.
INPUT #99, NSinks%
REDIM Demand(NSinks%) AS DemandType

R$ = ""
FOR I% = 1 TO NSinks%
  INPUT #99, Demand(I%).Num
  INPUT #99, Demand(I%).Dmnd
  CALL DisplayMessage("Demand node no.:  " + STR$(Demand(I%).Num))
  CALL DisplayMessage("   Demand:  " + STR$(Demand(I%).Dmnd))
  IF CurrentLine% = MaxRow% AND R$ <> "Q" AND I% <> NSinks% THEN
    CALL Waiting("Press any key to continue or 'Q' to continue without waiting.", 24, 0, R$)
    CALL ScrollText(2, MaxCol% - 1, 2, MaxRow% - 1, MaxRow% - 2)
    CurrentLine% = 2
  END IF
NEXT I%
CALL Waiting("Press any key to continue.", 24, 0, R$)

'Get the Variable data file name.
INPUT #99, FLOTX1
INPUT #99, FLOTY1
INPUT #99, FLOTX2
INPUT #99, FLOTY2
INPUT #99, W
INPUT #99, V
INPUT #99, NPlans%
INPUT #99, VulParam
CLOSE #99

Flag% = 4
```

```
END SUB

SUB Save (Problem AS STRING)

SHARED CurrentLine%, W, V, NPlane%
SHARED MapXMax, MapXMin, MapYMax, MapYMin
SHARED FLOTX1, FLOTY1, FLOTX2, FLOTY2
SHARED NNodes%, NodePointer%(), NLinks%
SHARED NSinks%, Demand() AS DemandType

  KILL Problem + ".DAT"
  OPEN Problem + ".DAT" FOR OUTPUT AS #99

  'Save the corners of the map.
  PRINT #99, STR$(MapXMax)
  PRINT #99, STR$(MapXMin)
  PRINT #99, STR$(MapYMax)
  PRINT #99, STR$(MapYMin)

  'Save the node data.
  PRINT #99, STR$(NNodes%)
  FOR I% = 1 TO NNodes%
    CALL GetNode(NodePointer%(I%), East, North, SSEast, SSNorth, E1, T$, D$, Vuln, DOS%, S%)
    PRINT #99, STR$(NodePointer%(I%))
    PRINT #99, STR$(East)
    PRINT #99, STR$(North)
    PRINT #99, STR$(E1)
    T$ = CHR$(34) + T$ + CHR$(34)
    D$ = CHR$(34) + D$ + CHR$(34)
    PRINT #99, T$
    PRINT #99, D$
    PRINT #99, STR$(Vuln)
    PRINT #99, STR$(DOS%)
    PRINT #99, STR$(S%)
  NEXT I%

  'Save the link data.
  PRINT #99, STR$(NLinks%)
  FOR I% = 1 TO NLinks%
    CALL GetLink(I%, Hd%, T1%, Length, RC%, Vuln)
    PRINT #99, STR$(Hd%)
    PRINT #99, STR$(T1%)
    PRINT #99, STR$(Length)
    PRINT #99, STR$(RC%)
    PRINT #99, STR$(Vuln)
  NEXT I%

  'Save the demand data.
  PRINT #99, STR$(NSinks%)
  FOR I% = 1 TO NSinks%
    PRINT #99, STR$(Demand(I%).Num)
    PRINT #99, STR$(Demand(I%).Dmnd)
  NEXT I%

  'Save the Variable data file name.
  PRINT #99, STR$(FLOTX1)
  PRINT #99, STR$(FLOTY1)
  PRINT #99, STR$(FLOTX2)
  PRINT #99, STR$(FLOTY2)
  PRINT #99, STR$(W)
  PRINT #99, STR$(V)
  PRINT #99, STR$(NPlane%)
  PRINT #99, STR$(VulParam)
  CLOSE #99
```

```
END SUB

SUB SemiScale (NNodes%, NLinks%, NodePointer%())

DIM ColNo%(NNodes%), RowNo%(NNodes%)
SHARED MapXMax, MapYMax, MapXMin, MapYMin
SHARED Alpha, Beta

  CALL DisplayMessage("Computing normalized coords.")
  XDif = MapXMax - MapXMin
  YDif = MapYMax - MapYMin
  FOR I% = 1 TO NNodes%
    CALL GetNode(NodePointer%(I%), E1, N1, SSE1, SSN1, E11, T1$, D1$, V1, D1%, S1%)
    E1 = E1 - 10000! * (E1 < MapXMin)
    N1 = N1 - 10000! * (N1 < MapYMin)
    SSE1 = (E1 - MapXMin) / XDif
    SSN1 = (N1 - MapYMin) / YDif
    CALL PutNode(NodePointer%(I%), E1, N1, SSE1, SSN1, E11, T1$, D1$, V1, D1%, S1%)
  NEXT I%

  CLS
  DO
    LOCATE PromptRow%, 1
    INPUT "Enter spreading parameter alpha (0 <= alpha <= 1):  "; Alpha
    IF Alpha < 0! OR Alpha > 1! THEN PrintError "Can't you read."
  LOOP UNTIL 0! <= Alpha AND Alpha <= 1!

  CLS
  DO
    DO
      LOCATE PromptRow%, 1
      PRINT "Enter ranking parameter beta (0 < beta <= 1):  ";
      INPUT Beta
      IF Beta <= 0! OR Beta > 1! THEN PrintError "Can't you read."
    LOOP UNTIL 0! < Beta AND Beta <= 1!
    CLS

    CALL DisplayMessage("Assigning cells.")
    FOR I% = 1 TO NNodes%
      CALL GetNode(NodePointer%(I%), E1, N1, SSE1, SSN1, E11, T1$, D1$, V1, D1%, S1%)
      ColNo%(I%) = SSE1 / Beta
      RowNo%(I%) = SSN1 / Beta
    NEXT I%

    CALL DisplayMessage("Checking for nodes assigned to the same cell.")
    Ok% = True%
    FOR I% = 1 TO NNodes% - 1
      IF RowNo%(I%) = RowNo%(I% + 1) THEN
        Ok% = Ok% AND ColNo%(I%) <> ColNo%(I% + 1)
      END IF
    NEXT I%
    IF NOT Ok% THEN CALL PrintError("Beta is too large.")
  LOOP UNTIL Ok%
  CLS

  CALL DisplayMessage("Computing the new Semi-scaled coords.")
  FOR I% = 1 TO NNodes%
    CALL GetNode(NodePointer%(I%), E1, N1, SSE1, SSN1, E11, T1$, D1$, V1, D1%, S1%)
    SSE1 = E1 + Alpha * (((ColNo%(I%) - .5) * XDif * Beta + MapXMin) - E1)
    SSN1 = N1 + Alpha * (((RowNo%(I%) - .5) * YDif * Beta + MapYMin) - N1)
    CALL PutNode(NodePointer%(I%), E1, N1, SSE1, SSN1, E11, T1$, D1$, V1, D1%, S1%)
  NEXT I%

END SUB

FUNCTION ShortestPath (Start%, Stp%, Dist(), P$)
```

```
SHARED NNodes%
I% = FRE(" ")           'Compresses the string memory.

  DIM PathDist(NNodes%), Visited%(NNodes%), Path$(NNodes%)

  FOR I% = 1 TO NNodes%
    IF I% = Start% THEN
      PathDist(I%) = 0!
      Visited%(I%) = True%
    ELSE
      PathDist(I%) = NoLink
      Visited%(I%) = False%
    END IF
    Path$(I%) = STR$(Start%)
  NEXT I%

  Y% = Start%
  SPN% = Y%
  DO
    SPA = NoLink
    FOR I% = 1 TO NNodes%
      IF NOT Visited%(I%) THEN
        IF PathDist(I%) > PathDist(Y%) + Dist(Y%, I%) THEN
          PathDist(I%) = PathDist(Y%) + Dist(Y%, I%)
          Path$(I%) = Path$(Y%)
        END IF
        IF PathDist(I%) <= SPA THEN
          P$ = Path$(I%)
          SPA = PathDist(I%)
          SPN% = I%
        END IF
      END IF
    NEXT I%
    Path$(SPN%) = P$ + "," + LTRIM$(STR$(SPN%))
    Y% = SPN%
    Visited%(Y%) = True%
  LOOP UNTIL Y% = Stp%
  ShortestPath = PathDist(Stp%)
  P$ = LTRIM$(Path$(Stp%))

END FUNCTION

SUB Solve (NNodes%, NodePointer%(), W, V, Supply() AS SupplyType, Demand() AS DemandType, NSinks%, NSources%, Problem AS STRING, Lag
(), Loss(), DTG$)

SHARED NLinks%, Speed(), NPlans%
I% = FRE(" ")           'Compresses the string memory.
REDIM Cost(NSources% + 1, NSinks% + 1), X(NSources% + 1, NSinks% + 1)
REDIM Lag(NSources%, 0 TO NSinks%)
REDIM P1$(NSources%, NSinks%), P2$(NSources%, NSinks%)
REDIM LinkCost(NNodes%, NNodes%)

  VIEW PRINT
  CLS
  LOCATE 10, 1
  PRINT TAB(11); "-------------------------------------------------------"
  PRINT TAB(11); "This routine solves the  Transportation  Problem  for  the"
  PRINT TAB(11); "Supply  Point  Locator  where the user wishes to determine"
  PRINT TAB(11); "the minimum cost schedule for  the  movement  of  supplies"
  PRINT TAB(11); "from specified supply points to specified supported units."
  PRINT TAB(11); "-------------------------------------------------------"

  NPlans% = NPlans% + 1
  FOR I% = 1 TO NNodes%
    FOR J% = 1 TO NNodes%
    LinkCost(I%, J%) = NoLink
  NEXT J%
NEXT I%

FOR I% = 1 TO NLinks%
  CALL GetLink(I%, Hd%, T1%, DIJ, RC%, QIJ)
  A% = 1
  WHILE Hd% <> NodePointer%(A%)
    A% = A% + 1
  WEND
  B% = 1
  WHILE T1% <> NodePointer%(B%)
    B% = B% + 1
  WEND
  Cst = (W + V * QIJ) * DIJ / Speed(RC%)
  LinkCost(A%, B%) = Cst
  LinkCost(B%, A%) = Cst
NEXT I%

CALL Waiting("Press any key to continue.", 24, 0, R$)
CLS

PRINT "Computing the cost matrix."
FOR I% = 1 TO NSources%
  A% = 1
  WHILE Supply(I%).Num <> NodePointer%(A%)
    A% = A% + 1
  WEND
  FOR J% = 1 TO NSinks%
    B% = 1
    WHILE Demand(J%).Num <> NodePointer%(B%)
      B% = B% + 1
    WEND
    Cost(I%, J%) = ShortestPath(A%, B%, LinkCost(), P1$(I%, J%))
  NEXT J%
NEXT I%

PRINT "Solving the transportation problem."
CALL Transport(Cost(), Supply(), Demand(), (NSinks%), (NSources%), UnusedSupply, UnmetDemand, TotalFlow, X())
ERASE Cost            'The Cost array is no longer needed at this point.

'Initialize the LinkCost array for computation of lag.
FOR I% = 1 TO NLinks%
  CALL GetLink(I%, Hd%, T1%, DIJ, RC%, QIJ)
  A% = 1
  WHILE Hd% <> NodePointer%(A%)
    A% = A% + 1
  WEND
  B% = 1
  WHILE T1% <> NodePointer%(B%)
    B% = B% + 1
  WEND
  Cst = DIJ / Speed(RC%)
  LinkCost(A%, B%) = Cst
  LinkCost(B%, A%) = Cst
NEXT I%

CALLS SETCOLOR(PathLinkColor%)

'Computing the Lag array.
LagTot = 0!
FOR I% = 1 TO NSources%
  FOR J% = 1 TO NSinks%
    Path$ = P1$(I%, J%)
    B% = VAL(Path$)
    P2$(I%, J%) = STR$(NodePointer%(B%))
    IF INSTR(Path$, ",") > 0 THEN
      Path$ = RIGHT$(Path$, LEN(Path$) - INSTR(Path$, ","))
    ELSE
      Path$ = ""
    END IF
    Lag(I%, J%) = 0!
    WHILE LEN(Path$) > 0
      A% = B%
      B% = VAL(Path$)
      P2$(I%, J%) = P2$(I%, J%) + STR$(NodePointer%(B%))
      IF INSTR(Path$, ",") > 0 THEN
        Path$ = RIGHT$(Path$, LEN(Path$) - INSTR(Path$, ","))
      ELSE
        Path$ = ""
      END IF
      CALL GetNode(NodePointer%(A%), E, N, SSE1, SSN1, E1, T$, D$, PI, AI%, S1%)
      CALL GetNode(NodePointer%(B%), E, N, SSE2, SSN2, E1, T$, D$, PI, AI%, S1%)
      CALLS NOVABS(SSE1, SSN1)
      CALLS LNABS(SSE2, SSN2)
      Lag(I%, J%) = Lag(I%, J%) + LinkCost(A%, B%)
    WEND
    Lag(I%, J%) = W * Lag(I%, J%) * X(I%, J%)
    LagTot = LagTot + Lag(I%, J%)
  NEXT J%
  Lag(I%, 0) = 0!
NEXT I%

'Initialize the LinkCost array for computation of loss.
FOR I% = 1 TO NLinks%
  CALL GetLink(I%, Hd%, T1%, DIJ, RC%, QIJ)
  A% = 1
  WHILE Hd% <> NodePointer%(A%)
    A% = A% + 1
  WEND
  B% = 1
  WHILE T1% <> NodePointer%(B%)
    B% = B% + 1
  WEND
  Cst = LinkCost(A%, B%) * QIJ
  LinkCost(A%, B%) = Cst
  LinkCost(B%, A%) = Cst
NEXT I%

'Computing the Loss array.
LossTot = 0!
REDIM Loss(NSources%, 0 TO NSinks%)
FOR I% = 1 TO NSources%
  FOR J% = 1 TO NSinks%
    Path$ = P1$(I%, J%)
    B% = VAL(Path$)
    IF INSTR(Path$, ",") > 0 THEN
      Path$ = RIGHT$(Path$, LEN(Path$) - INSTR(Path$, ","))
    ELSE
      Path$ = ""
    END IF
    Loss(I%, J%) = 0!
    WHILE LEN(Path$) > 0
      A% = B%
      B% = VAL(Path$)
      IF INSTR(Path$, ",") > 0 THEN
        Path$ = RIGHT$(Path$, LEN(Path$) - INSTR(Path$, ","))
      ELSE
        Path$ = ""
      END IF
      Loss(I%, J%) = Loss(I%, J%) + LinkCost(A%, B%)
    WEND
```

```
      Loss(I%, J%) = V * Loss(I%, J%) * X(I%, J%)
      LossTot = LossTot + Loss(I%, J%)
    NEXT J%
    CALL GetNode(Supply(I%).Num, E, N, SSE1, SSN1, E1, T$, D$, PI, AI%, S1%)
    Loss(I%, 0) = V * PI * AI% * Supply(I%).Sup
    LossTot = LossTot + Loss(I%, 0)
NEXT I%

'save the solution.
D$ = DATE$
T$ = TIME$
DTG$ = MID$(D$, 4, 2) + LEFT$(T$, 2) + MID$(T$, 4, 2) + " "
SELECT CASE VAL(D$)
  CASE 1
    DTG$ = DTG$ + "JAN "
  CASE 2
    DTG$ = DTG$ + "FEB "
  CASE 3
    DTG$ = DTG$ + "MAR "
  CASE 4
    DTG$ = DTG$ + "APR "
  CASE 5
    DTG$ = DTG$ + "MAY "
  CASE 6
    DTG$ = DTG$ + "JUN "
  CASE 7
    DTG$ = DTG$ + "JLY "
  CASE 8
    DTG$ = DTG$ + "AUG "
  CASE 9
    DTG$ = DTG$ + "SEP "
  CASE 10
    DTG$ = DTG$ + "OCT "
  CASE 11
    DTG$ = DTG$ + "NOV "
  CASE 12
    DTG$ = DTG$ + "DEC "
END SELECT
DTG$ = DTG$ + RIGHT$(D$, 2)
ProbPlan$ = Problem + " / " + LTRIM$(STR$(NPlans%))

KILL "LASTPLAN"
OPEN "LASTPLAN" FOR OUTPUT AS #99
PRINT #99, TAB(40 - LEN(ProbPlan$) \ 2); ProbPlan$
PRINT #99, TAB(40 - LEN(DTG$) \ 2); DTG$
WV$ = "W = " + STR$(W) + "         V = " + STR$(V)
PRINT #99, TAB(31); "Transportation Solution"
PRINT #99, TAB(31); "-----------------------"
PRINT #99, TAB(16); "Schedule"; TAB(55); "Resupply"
PRINT #99, TAB(16); "--------"; TAB(55); "--------"
PRINT #99, TAB(9); "Source"; TAB(21); "Destination"; TAB(43); "Quantity (S/T)"; TAB(65); "Costs"
PRINT #99, TAB(7); "-----------------------------";
PRINT #99, TAB(40); "--------------------------------"

FOR I% = 1 TO NSources%
  FOR J% = 1 TO NSinks%
    IF X(I%, J%) <> 0! THEN
      PRINT #99, TAB(9); "Node "; Supply(I%).Num;
      PRINT #99, TAB(23); "Node "; Demand(J%).Num;
      PRINT #99, TAB(48); X(I%, J%);
      PRINT #99, TAB(64); Lag(I%, J%) + Loss(I%, J%)
    END IF
  NEXT J%
NEXT I%
PRINT #99, TAB(7); STRING$(64, "-")
PRINT #99, TAB(14); "Unused Supply = "; UnusedSupply;
PRINT #99, TAB(40); "Total Costs   = "; LagTot + LossTot
PRINT #99, TAB(14); "Unmet Demand  = "; UnmetDemand;
PRINT #99, TAB(40); "Loss (S/T)    = "; LossTot
PRINT #99, TAB(14); "Total Flow    = "; TotalFlow;
PRINT #99, TAB(40); "Lag  (S/T)    = "; LagTot

'Save the plan.
PRINT #PlanFile%, NPlans%
PRINT #PlanFile%, DTG$
PRINT #PlanFile%, LagTot + LossTot
PRINT #PlanFile%, LagTot
PRINT #PlanFile%, LossTot
PRINT #PlanFile%, NSources%
FOR I% = 1 TO NSources%
  PRINT #PlanFile%, Supply(I%).Num
  PRINT #PlanFile%, Supply(I%).Sup
  FOR J% = 1 TO NSinks%
    PRINT #PlanFile%, P2$(I%, J%)
  NEXT J%
NEXT I%
RINT #PlanFile%, PlanEnd$

CLS
LOCATE 12, 1
PRINT TAB(40 - LEN(WV$) \ 2); WV$
PRINT TAB(14); "Unused Supply = "; UnusedSupply;
PRINT TAB(40); "Total Costs   = "; LagTot + LossTot
PRINT TAB(14); "Unmet Demand  = "; UnmetDemand;
PRINT TAB(40); "Loss (S/T)    = "; LossTot
PRINT TAB(14); "Total Flow    = "; TotalFlow;
PRINT TAB(40); "Lag  (S/T)    = "; LagTot

CALL Waiting("Press any key to continue.", 24, 0, R$)
CLS
PRINT #99, " "
FOR I% = 1 TO NSources%
  PRINT TAB(17); "Loss at supply point"; Supply(I%).Num; " = "; Loss(I%, 0)
  PRINT #99, TAB(17); "Loss at supply point"; Supply(I%).Num; " = "; Loss(I%, 0)
NEXT I%
CALL Waiting("Press any key to continue.", 24, 0, R$)
CLS
PRINT #99, " "
FOR J% = 1 TO NSinks%
  LossTot = 0!
  FOR I% = 1 TO NSources%
    LossTot = LossTot + Loss(I%, J%)
  NEXT I%
  PRINT TAB(17); "Loss in transit to node"; Demand(J%).Num; " = "; LossTot
  PRINT #99, TAB(17); "Loss in transit to node"; Demand(J%).Num; " = "; LossTot
NEXT J%
CALL Waiting("Press any key to continue.", 24, 0, R$)
CLS
PRINT #99, " "
FOR J% = 1 TO NSinks%
  LagTot = 0!
  FOR I% = 1 TO NSources%
    LagTot = LagTot + Lag(I%, J%)
  NEXT I%
  PRINT TAB(17); "Lag in transit to node"; Demand(J%).Num; " = "; LagTot
  PRINT #99, TAB(17); "Lag in transit to node"; Demand(J%).Num; " = "; LagTot
NEXT J%

PRINT #99, " "
FOR I% = 1 TO NSources%
  PRINT #99, TAB(17); "Paths from supply point "; Supply(I%).Num
```

```
    FOR J% = 1 TO NSinks%
      PRINT #99, TAB(17); P2$(I%, J%)
    NEXT J%
    PRINT #99, " "
  NEXT I%
  PRINT #99, " "
  PRINT #99, TAB(40 - LEN(WV$) \ 2); WV$

  CLOSE #99

  CALL Waiting("Press any key to continue.", 24, 0, R$)
  CLS
  VIEW PRINT MaxRow% + 1 TO 25

END SUB

SUB Transport (C(), Supply() AS SupplyType, Demand() AS DemandType, NSinks%, NSources%, UnusedSupply, UnmetDemand, TotalFlow, X())
DEFINT I-J, N-N

DIM CP%(NSources% + 1)
DIM K(NSources% + 1)
DIM K2(NSources% + 1)
DIM S%(NSources% + 1)
DIM S1(NSources% + 1)
DIM U(NSources% + 1)

DIM D%(NSinks% + 1)
DIM DM%(NSinks% + 1)
DIM K3(NSinks% + 1)
DIM L1(NSinks% + 1)
DIM V(NSinks% + 1)

DIM K1(NSources% + 1, NSinks% + 1)

      'Convert SPL data for use by the transportation algorithm.
      S%(NSources% + 1) = 0
      D%(NSinks% + 1) = 0

      TotalSupply = 0!
      FOR I = 1 TO NSources%
        S%(I) = Supply(I).Num
        CP%(I) = Supply(I).Sup
        TotalSupply = TotalSupply + CP%(I)
      NEXT I

      TotalDemand = 0!
      FOR J = 1 TO NSinks%
        D%(J) = Demand(J).Num
        DM%(J) = Demand(J).Dmnd
        TotalDemand = TotalDemand + DM%(J)
      NEXT J

      'Establish dummy sources or destinations.
      IF TotalSupply < TotalDemand THEN
        'Set penalty cost for shortages equal to zero.
        CP%(NSources% + 1) = TotalDemand - TotalSupply
        FOR J = 1 TO NSinks%
          C(NSources% + 1, J) = 0
        NEXT J
        NSources% = NSources% + 1
      ELSEIF TotalSupply > TotalDemand THEN
        'Set penalty cost for storage equal to zero.
        DM%(NSinks% + 1) = TotalSupply - TotalDemand
        FOR I = 1 TO NSources%
          C(I, NSinks% + 1) = 0
        NEXT I
      NSinks% = NSinks% + 1
    END IF

    'Solve the problem
    FOR I = 1 TO NSources%
      T = C(I, 1)
      FOR J = 2 TO NSinks%
        IF T >= C(I, J) THEN T = C(I, J)
      NEXT J
      U(I) = -T
      FOR J = 1 TO NSinks%
        IF C(I, J) <= T THEN X(I, J) = .5
      NEXT J
    NEXT I

    FOR J = 1 TO NSinks%
      T = 1E+30
      FOR I = 1 TO NSources%
        IF X(I, J) > 0 THEN 9500 ELSE K(I) = C(I, J) + U(I)
        IF T >= K(I) THEN T = K(I)
      NEXT I
      V(J) = -T
      FOR I = 1 TO NSources%
        IF K(I) <= T THEN X(I, J) = .5
      NEXT I
9500 NEXT J

    FOR I = 1 TO NSources%
      FOR J = 1 TO NSinks%
        IF X(I, J) <> 0! THEN
          T = CP%(I)
          IF T >= DM%(J) THEN T = DM%(J)
          X(I, J) = X(I, J) + T
          CP%(I) = CP%(I) - T
          DM%(J) = DM%(J) - T
        END IF
      NEXT J
    NEXT I

    FOR J = 1 TO NSinks%
      IF DM%(J) <> 0 THEN 10200
    NEXT J
    GOTO 13100
10200 FOR I = 1 TO NSources%
      K2(I) = 0
      S1(I) = 0!
    NEXT I
    FOR I = 1 TO NSinks%
      K3(I) = 0
      L1(I) = 0
    NEXT I
    FOR I = 1 TO NSources%
      IF CP%(I) <> 0 THEN
        K2(I) = CP%(I)
        S1(I) = 0
      END IF
    NEXT I

    DO
      I = 1
      DO
        IF K2(I) <> 0 THEN
          FOR J = 1 TO NSinks%
            IF X(I, J) <> 0 AND K3(J) = 0 THEN
              K3(J) = K2(I)
              L1(J) = I
            IF DM%(J) <> 0 THEN 12200
          END IF
        NEXT J
      END IF
      I = I + 1
    LOOP UNTIL I > NSources%

    N1 = 0
    FOR J = 1 TO NSinks%
      IF K3(J) <> 0 THEN
        FOR I = 1 TO NSources%
          IF K2(I) = 0 AND X(I, J) > .5 THEN
            K2(I) = X(I, J) - .5
            IF K2(I) >= K3(J) THEN K2(I) = K3(J)
            S1(I) = J
            N1 = 1
          END IF
        NEXT I
      END IF
    NEXT J
  LOOP UNTIL N1 <> 1
  GOTO 13400

12200 F = DM%(J)
  IF F >= K3(J) THEN F = K3(J)
  A2 = J
  DM%(J) = DM%(J) - F

12600 A1 = L1(A2)
    X(A1, A2) = X(A1, A2) + F
    IF S1(A1) = 0 THEN 12900
    A2 = S1(A1)
    X(A1, A2) = X(A1, A2) - F
    GOTO 12600

12900 CP%(A1) = CP%(A1) - F
  FOR J = 1 TO NSinks%
    IF DM%(J) <> 0 THEN 10200
  NEXT J
13100 Q = 0
  FOR I = 1 TO NSources%
    FOR J = 1 TO NSinks%
      IF X(I, J) >= .5 THEN X(I, J) = X(I, J) - .5
      Q = Q + X(I, J) * C(I, J)
    NEXT J
  NEXT I
  GOTO 14800

13400 FOR I = 1 TO NSinks%
    FOR J = 1 TO NSources%
      K1(J, I) = 999991
    NEXT J
  NEXT I
  Q = 999991
  FOR I = 1 TO NSources%
    IF K2(I) <> 0 THEN
      FOR J = 1 TO NSinks%
        IF K3(J) = 0 THEN
          K1(I, J) = C(I, J) + U(I) + V(J)
          IF Q >= K1(I, J) THEN Q = K1(I, J)
        END IF
      NEXT J
    END IF
  NEXT I
  FOR I = 1 TO NSources%
    FOR J = 1 TO NSinks%
```

```
        IF K1(I, J) <= Q THEN X(I, J) = .8
      NEXT J
    NEXT I
    FOR J = 1 TO NSinks%
      IF K3(J) <> 0 THEN
        V(J) = V(J) + Q
        FOR I = 1 TO NSources%
          IF K2(I) <= 0 THEN X(I, J) = 0
        NEXT I
      END IF
    NEXT J
    FOR I = 1 TO NSources%
      IF K2(I) <> 0 THEN U(I) = U(I) - Q
    NEXT I
    GOTO 10300
14800

    TotalFlow = 0!
    FOR I = 1 TO NSources%
      FOR J = 1 TO NSinks%
        IF C(I, J) <> 0! AND X(I, J) > 0! THEN TotalFlow = TotalFlow + X(I, J)
      NEXT J
    NEXT I

    'Unused supply
    IF TotalSupply > TotalDemand THEN
      UnusedSupply = TotalSupply - TotalDemand
    ELSE
      UnusedSupply = 0!
    END IF

    'Unmet demand
    IF TotalDemand > TotalSupply THEN
      UnmetDemand = TotalDemand - TotalSupply
    ELSE
      UnmetDemand = 0!
    END IF

END SUB

SUB AddNode (NNodes%, NodePointer%())

SHARED MapXMin, MapYMin, MapXMax, MapYMax
SHARED FLOTX1, FLOTY1, FLOTX2, FLOTY2

  OkToAdd% = True%
  IF NNodes% = MaxNNodes% THEN OkToAdd% = False%
  IF NOT OkToAdd% THEN CALL PrintError("No room for a new node.")
  IF NOT OkToAdd% THEN EXIT SUB

  'Get node number from user.
  DO
    LOCATE PromptRow%, 1
    INPUT "Enter the node number of the node to add:  ", AN%
    CLS
    'Check for duplicate node number.
    Check% = 1
    Found% = False%
    WHILE Check% <= NNodes% AND NOT Found%
      Found% = AN% = NodePointer%(Check%)
      Check% = Check% + 1
    WEND
    IF Found% THEN CALL PrintError("Duplicate Node number.")
  LOOP UNTIL NOT Found%

  DO
  'Get Easting from user.
  DO
    LOCATE PromptRow%, 1
    INPUT "Enter easting:  ", NewE
    CLS
    NewSSE = NewE - 10000! * (NewE < MapXMin)
    Valid% = MapXMin <= NewSSE AND NewSSE <= MapXMax
    IF NOT Valid% THEN CALL PrintError("Value out of range.")
  LOOP UNTIL Valid%

  'Get Northing from user.
  DO
    LOCATE PromptRow%, 1
    INPUT "Enter northing:  ", NewN
    CLS
    NewSSN = NewN - 10000! * (NewN < MapYMin)
    Valid% = MapYMin <= NewSSN AND NewSSN <= MapYMax
    IF NOT Valid% THEN CALL PrintError("Value out of range.")
  LOOP UNTIL Valid%

  'Check for node already at new location.
  Check% = 1
  Found% = False%
  WHILE Check% <= NNodes% AND NOT Found%
    CALL GetNode(NodePointer%(Check%), CE, CN, CSSE, CSSN, CEl, CT$, CD$, CV, CD%, CS%)
    Found% = NewE = CE AND NewN = CN
    Check% = Check% + 1
  WEND

  IF Found% THEN CALL PrintError("There is already a node at that location.")

LOOP UNTIL NOT Found%

CALL DisplayMessage("Node no.:  " + STR$(AN%))
CALL DisplayMessage("  Easting:  " + STR$(NewE))
CALL DisplayMessage("  Northing:  " + STR$(NewN))

'Get elevation from user.
LOCATE PromptRow%, 1
INPUT "Enter elevation:  ", NewEl
CLS
CALL DisplayMessage("  Elevation:  " + STR$(NewEl))

'Get terrain descriptor from user.
LOCATE PromptRow%, 1
INPUT "Enter terrain descriptor:  ", NewT$
CLS
IF LEN(NewT$) > 40 THEN NewT$ = LEFT$(NewT$, 40)
CALL DisplayMessage("  Terrain:  " + NewT$)

'Get area descriptor from user.
LOCATE PromptRow%, 1
INPUT "Enter area descriptor:  ", NewD$
CLS
IF LEN(NewD$) > 40 THEN NewD$ = LEFT$(NewD$, 40)
CALL DisplayMessage("  Description:  " + NewD$)

'Compute default vulnerability.
FLOT = SQR((FLOTX2 - FLOTX1) ^ 2 + (FLOTY2 - FLOTY1) ^ 2)
Dis = SQR((NewSSE - FLOTX1) ^ 2 + (NewSSN - FLOTY1) ^ 2)
ADotB = (FLOTX2 - FLOTX1) * (NewSSE - FLOTX1) + (FLOTY2 - FLOTY1) * (NewSSN - FLOTY1)
Angle = ATN(SQR(1! / (ADotB / (FLOT * Dis)) ^ 2 - 1!))
DDis = .01 * Dis * SIN(Angle)
OldV = EXP(-VulParam * DDis)

'Get Vulnerability from user.
```

```
DO
  CALL DisplayMessage("Default vulnerability:  " + STR$(OldV))
  LOCATE PromptRow%, 1
  LINE INPUT "Enter vulnerability (Return for default):  ", Lin$
  Lin$ = LTRIM$(RTRIM$(Lin$))
  IF LEN(Lin$) > 0 THEN
    NewV = VAL(Lin$)
  ELSE
    NewV = OldV
  END IF
  IF NewV < 0! OR NewV > 1! THEN CALL PrintError("Value out of range.")
LOOP UNTIL 0! <= NewV AND NewV <= 1!
CALL DisplayMessage("  Vulnerability:  " + STR$(NewV))

'Get Days of supply from user.
DO
  LOCATE PromptRow%, 1
  INPUT "Enter the number of days of supply if any at the node;  ", NewDOS%
  CLS
  IF NewDOS% < 0 THEN CALL PrintError("Non-negative numbers only.")
LOOP UNTIL NewDOS% >= 0
CALL DisplayMessage("  Days of supply:  " + STR$(NewDOS%))

'Get the suitability parameter from the user.
DO
  LOCATE PromptRow%, 1
  INPUT "Enter node suitability parameter (0 for unsuitable, 1 for suitable):  ", NewS%
  CLS
LOOP UNTIL NewS% = 1 OR NewS% = 0
CALL DisplayMessage("   Suitability:  " + STR$(NewS%))

NNodes% = NNodes% + 1
NodePointer%(NNodes%) = AN%

CALL PutNode(AN%, NewE, NewN, NewSSE, NewSSN, NewE1, NewT8, NewD6, NewV, NewDOS%, NewS%)

END SUB

SUB ComputeNodeVul (NNodes%, NodePointer%(), FLOTX1, FLOTY1, FLOTX2, FLOTY2)

SHARED MapXMin, MapYMin, MapXMax, MapYMax
ON ERROR GOTO ErrorHandle

 DO
  Ok% = True%
  LOCATE PromptRow%, 1
  INPUT "Enter (X1,Y1) and (X2,Y2) to define the FLOT:  ", FLOTX1, FLOTY1, FLOTX2, FLOTY2
  FLOTX1 = FLOTX1 - 10000! * (FLOTX1 < MapXMin)
  FLOTX2 = FLOTX2 - 10000! * (FLOTX2 < MapXMin)
  FLOTY1 = FLOTY1 - 10000! * (FLOTY1 < MapYMin)
  FLOTY2 = FLOTY2 - 10000! * (FLOTY2 < MapYMin)
  IF FLOTX1 < MapXMin OR FLOTX1 > MapXMax OR FLOTY1 < MapYMin OR FLOTY1 > MapYMax THEN
    CALL PrintError("(X1,Y1) is out of bounds.")
    Ok% = False%
  END IF
  IF FLOTX2 < MapXMin OR FLOTX2 > MapXMax OR FLOTY2 < MapYMin OR FLOTY2 > MapYMax THEN
    CALL PrintError("(X2,Y2) is out of bounds.")
    Ok% = False%
  END IF
  IF FLOTX1 = FLOTX2 AND FLOTY1 = FLOTY2 THEN
    CALL PrintError("(X1,Y1) must not equal (X2,Y2).")
    Ok% = False%
  END IF
 LOOP UNTIL Ok%
 CLS
```

```
  FLOT = SQR((FLOTX2 - FLOTX1) ^ 2 + (FLOTY2 - FLOTY1) ^ 2)
  FOR I% = 1 TO NNodes%
    CALL GetNode(NodePointer%(I%), E, N, SSE, SSN, E1, T$, D$, V, D%, S%)
    TempE = E
    IF TempE < MapXMin THEN TempE = TempE + 10000!
    TempN = N
    IF TempN < MapYMin THEN TempN = TempN + 10000!
    Dis = SQR((TempE - FLOTX1) ^ 2 + (TempN - FLOTY1) ^ 2)
    ADotB = (FLOTX2 - FLOTX1) * (TempE - FLOTX1) + (FLOTY2 - FLOTY1) * (TempN - FLOTY1)
    Angle = ATN(SQR(1! / (ADotB / (FLOT * Dis)) ^ 2 - 1!))
    DDis = .01 * Dis * SIN(Angle)
    V = EXP(-VulParam * DDis)
    CALL PutNode(NodePointer%(I%), E, N, SSE, SSN, E1, T$, D$, V, D%, S%)
  NEXT I%

END SUB

SUB DelNode (NNodes%, NodePointer%(), NLinks%, NSinks%, Demand() AS DemandType)

  'Get node number from user.
  DO
    LOCATE PromptRow%, 1
    INPUT "Enter the node number of the node to delete (0 to stop):  ", DN%
    CLS
    'Find the node in the pointer array.
    Check% = 1
    Found% = False%
    WHILE Check% <= NNodes% AND NOT Found% AND DN% <> 0
      Found% = DN% = NodePointer%(Check%)
      Check% = Check% + 1
    WEND
    IF NOT Found% AND DN% <> 0 THEN CALL PrintError("That node does not exist.")
  LOOP UNTIL Found% OR DN% = 0
  IF DN% = 0 THEN EXIT SUB

  'Delete node pointer.
  Check% = Check% - 1
  FOR I% = Check% TO NNodes% - 1
    NodePointer%(I%) = NodePointer%(I% + 1)
  NEXT I%
  NNodes% = NNodes% - 1

  'Delete any links with a head or tail at the node.
  NewNLinks% = 0
  FOR I% = 1 TO NLinks%
    CALL GetLink(I%, Hd%, Tl%, Length, RC%, LinkVuln)
    IF Hd% <> DN% AND Tl% <> DN% THEN
      NewNLinks% = NewNLinks% + 1
      CALL PutLink(NewNLinks%, Hd%, Tl%, Length, RC%, LinkVuln)
    END IF
  NEXT I%
  NLinks% = NewNLinks%

  'Find the node in the demand pointer array.
  Check% = 1
  Found% = False%
  WHILE Check% <= NSinks% AND NOT Found%
    Found% = DN% = Demand(Check%).Num
    Check% = Check% + 1
  WEND

  'Delete the node from the demand pointer array if found.
  IF Found% THEN
    Check% = Check% - 1
    FOR I% = Check% TO NSinks% - 1
      Demand(I%).Num = Demand(I% + 1).Num
      Demand(I%).Dmnd = Demand(I% + 1).Dmnd
    NEXT I%
    NSinks% = NSinks% - 1
  END IF

END SUB

SUB DrawNodes (NNodes%, NodePointer%(), NSinks%, Demand() AS DemandType, NSources%, Supply() AS SupplyType)

SHARED MapXMax, MapXMin
ON ERROR GOTO ErrorHandle

  Radius! = (3! / 1024!) * (MapXMax - MapXMin)

  FOR I% = 1 TO NNodes%
    CALL GetNode(NodePointer%(I%), E1, N1, SSE1, SSN1, E11, T1$, D1$, V1, D1%, S1%)
    Check% = 1
    SupplyNode% = False%
    WHILE Check% <= NSources% AND NOT SupplyNode%
      SupplyNode% = NodePointer%(I%) = Supply(Check%).Num
      Check% = Check% + 1
    WEND
    Check% = 1
    DemandNode% = False%
    WHILE Check% <= NSinks% AND NOT DemandNode% AND NOT SupplyNode%
      DemandNode% = NodePointer%(I%) = Demand(Check%).Num
      IF DemandNode% THEN Dem = Demand(Check%).Dmnd
      Check% = Check% + 1
    WEND
    IF DemandNode% THEN
      CALLS SETCOLOR(DemandNodeColor%)
    ELSEIF NOT S1% THEN
      CALLS SETCOLOR(TextColor%)
    ELSE
      CALLS SETCOLOR(NodeColor%)
    END IF

    CALLS MOVABS(SSE1, SSN1)
    IF SupplyNode% THEN
      CALLS CIR(Radius!)
    ELSEIF DemandNode% THEN
      Dem = ((Dem / 30!) / 1024!) * (MapXMax - MapXMin)
      CALLS FCIR(Radius! + Dem)
    ELSE
      CALLS FCIR(Radius!)
    END IF
  NEXT I%

END SUB

SUB EnterDOS (NNodes%, NodePointer%())

ON ERROR GOTO ErrorHandle

  DO
    LOCATE PromptRow%, 1
    INPUT "Enter the DOS at any node in the network:  ", DaysOfSupply%
    CLS
    IF DaysOfSupply% < 0 THEN CALL PrintError("Value must be non-negative.")
  LOOP UNTIL DaysOfSupply% >= 0

  FOR I% = 1 TO NNodes%
    CALL GetNode(NodePointer%(I%), E, N, SSE, SSN, E1, T$, D$, V, D%, S%)
    CALL PutNode(NodePointer%(I%), E, N, SSE, SSN, E1, T$, D$, V, DaysOfSupply%, S%)
  NEXT I%
```

```
 DO
   LOCATE PromptRow%, 1
   INPUT "Should any nodes have a different value for DOS (Y/N)"; Answer$
   CLS
   IF LEN(Answer$) > 0 THEN
     Answer$ = UCASE$(LEFT$(Answer$, 1))
   ELSE
     Answer$ = "Y"
   END IF
 LOOP UNTIL Answer$ = "Y" OR Answer$ = "N"

 IF Answer$ = "Y" THEN
   DO
     DO
       LOCATE PromptRow%, 1
       INPUT "Enter node number to change (-1 to stop):  ", N%
       CLS
       Found% = False%
       IF N% <> -1 THEN
         LookAt% = 1
         WHILE LookAt% <= NNodes% AND NOT Found%
           Found% = N% = NodePointer%(LookAt%)
           LookAt% = LookAt% + 1
         WEND
       END IF
       Found% = Found% OR N% = -1
       IF NOT Found% THEN CALL PrintError("That node does not exist.")
     LOOP UNTIL Found%

     IF N% <> -1 THEN
       DO
         LOCATE PromptRow%, 1
         PRINT "Enter the DOS at node "; N%; ":  ";
         INPUT DaysOfSupply%
         CLS
         IF DaysOfSupply% < 0 THEN CALL PrintError("Value must be non-negative.")
       LOOP UNTIL DaysOfSupply% >= 0
       CALL GetNode(N%, E, N, SSE, SSN, E1, T$, D$, V, D%, S%)
       CALL PutNode(N%, E, N, SSE, SSN, E1, T$, D$, V, DaysOfSupply%, S%)
     END IF
   LOOP UNTIL N% = -1
 END IF

END SUB

SUB GetNode (I%, East, North, SSEast, SSNorth, E1, T$, D$, V, D%, S%)

DIM Temp AS NodeType
ON ERROR GOTO ErrorHandle

  GET #NodeRND%, I%, Temp

 East = Temp.E
 North = Temp.N
 SSEast = Temp.SSE
 SSNorth = Temp.SSN
 E1 = Temp.Elev
 T$ = Temp.Terr
 D$ = Temp.Desc
 V = Temp.Vuln
 D% = Temp.DOS
 S% = Temp.Suitable

END SUB

SUB ModifyNode (NNodes%, NodePointer%(), Demand() AS DemandType, NSinks%)
```

```
ON ERROR GOTO ErrorHandle

  DO
    LOCATE PromptRow%, 1
    INPUT "Enter the node number to modify:  ", ModNode%
    CLS
    Check% = 1
    Valid% = False%
    WHILE Check% <= NNodes% AND NOT Valid%
      Valid% = ModNode% = NodePointer%(Check%)
      Check% = Check% + 1
    WEND
    IF NOT Valid% THEN CALL PrintError("That node does not exist.")
  LOOP UNTIL Valid%

  CALL GetNode(ModNode%, OldE, OldN, OldSSE, OldSSN, OldEl, OldT$, OldD$, OldV, OldDOS%, OldS%)

  CALL DisplayMessage("Easting:  " + STR$(OldE))
  CALL DisplayMessage("Northing:  " + STR$(OldN))

  CALL DisplayMessage("Current Elevation:  " + STR$(OldEl))
  LOCATE PromptRow%, 1
  LINE INPUT "Enter Elevation (Return for no change):  ", Lin$
  CLS
  Lin$ = LTRIM$(RTRIM$(Lin$))
  IF LEN(Lin$) > 0 THEN
    NewEl = VAL(Lin$)
  ELSE
    NewEl = OldEl
  END IF

  CALL DisplayMessage("Current Terrain descriptor:  " + OldT$)
  LOCATE PromptRow%, 1
  LINE INPUT "Enter Terrain description (Return for no change):  ", Lin$
  CLS
  Lin$ = LTRIM$(RTRIM$(Lin$))
  IF LEN(Lin$) > 40 THEN Lin$ = LEFT$(Lin$, 40)
  IF LEN(Lin$) > 0 THEN
    NewT$ = Lin$
  ELSE
    NewT$ = OldT$
  END IF

  CALL DisplayMessage("Current Area descriptor:  " + OldD$)
  LOCATE PromptRow%, 1
  LINE INPUT "Enter Area description (Return for no change):  ", Lin$
  Lin$ = LTRIM$(RTRIM$(Lin$))
  IF LEN(Lin$) > 40 THEN Lin$ = LEFT$(Lin$, 40)
  IF LEN(Lin$) > 0 THEN
    NewD$ = Lin$
  ELSE
    NewD$ = OldD$
  END IF

  DO
    CALL DisplayMessage("Current Vulnerability:  " + STR$(OldV))
    LOCATE PromptRow%, 1
    LINE INPUT "Enter Node Vulnerability (Return for no change):  ", Lin$
    CLS
    Lin$ = LTRIM$(RTRIM$(Lin$))
    IF LEN(Lin$) > 0 THEN
      NewV = VAL(Lin$)
    ELSE
      NewV = OldV
    END IF
  IF NewV < 0! OR NewV > 1! THEN CALL PrintError("Vulnerability out of range.")
LOOP UNTIL 0! <= NewV AND NewV <= 1!

IF NewV <> OldV THEN
  FOR I% = 1 TO NLinks%
    CALL GetLink(I%, Hd%, T1%, Length, RC%, LinkVuln)
    IF Hd% = ModNode% OR T1% = ModNode% THEN
      Tail% = T1%
      IF Tail% = ModNode% THEN Tail% = Hd%
      CALL GetNode(Tail%, TE, TN, TSSE, TSSN, TE1, TT$, TD$, TV, TDOS%, TS%)
      LinkVuln = (NewV + TV) / 2!
      CALL PutLink(I%, Hd%, T1%, Length, RC%, LinkVuln)
    END IF
  NEXT I%
END IF

DO
  CALL DisplayMessage("Current Days of supply:  " + STR$(OldDOS%))
  LOCATE PromptRow%, 1
  LINE INPUT "Enter Days of supply (Return for no change):  ", Lin$
  CLS
  Lin$ = LTRIM$(RTRIM$(Lin$))
  IF LEN(Lin$) > 0 THEN
    NewDOS% = VAL(Lin$)
  ELSE
    NewDOS% = OldDOS%
  END IF
  IF NewDOS% < 0! THEN CALL PrintError("Negative number not allowed.")
LOOP UNTIL NewDOS% >= 0!

DO
  CALL DisplayMessage("Current Suitability parameter:  " + STR$(OldS%))
  LOCATE PromptRow%, 1
  LINE INPUT "Enter node suitability parameter (0 for unsuitable, 1 for suitable):  ", Lin$
  CLS
  Lin$ = LTRIM$(RTRIM$(Lin$))
  IF LEN(Lin$) > 0 THEN
    NewS% = VAL(Lin$)
  ELSE
    NewS% = OldS%
  END IF
LOOP UNTIL NewS% = 1 OR NewS% = 0

CALL PutNode(ModNode%, OldE, OldN, OldSSE, OldSSN, NewE1, NewT$, NewD$, NewV, NewDOS%, NewS%)

Check% = 1
Found% = False%
WHILE Check% <= NSinks% AND NOT Found%
  Found% = ModNode% = Demand(Check%).Num
  Check% = Check% + 1
WEND

IF Found% THEN
  Check% = Check% - 1
  DO
    CALL DisplayMessage("Current Demand:  " + STR$(Demand(Check%).Dmnd))
    LOCATE PromptRow%, 1
    LINE INPUT "Enter Demand at node (Return for no change):  ", Lin$
    CLS
    Lin$ = LTRIM$(RTRIM$(Lin$))
    IF LEN(Lin$) > 0 THEN
      NewDmnd = VAL(Lin$)
    ELSE
      NewDmnd = Demand(Check%).Dmnd
    END IF
    IF NewDmnd < 0! THEN CALL PrintError("Non-negative value only.")
    LOOP UNTIL NewDmnd >= 0!
    Demand(Check%).Dmnd = NewDmnd
  END IF

END SUB

SUB OpenSource (NNodes%, NodePointer%(), NSources%, Supply() AS SupplyType, NSinks%, Sink() AS DemandType)

ON ERROR GOTO ErrorHandle

  DO
    LOCATE PromptRow%, 1
    INPUT "Enter the number of source nodes:  "; NSources%
    IF NSources% < 1 THEN CALL PrintError("There must be at least 1 supply node.")
    IF NSources% > NNodes% - NSinks% THEN CALL PrintError("Value too large.")
  LOOP UNTIL NSources% > 0 AND NSources% <= NNodes% - NSinks%
  CLS

  REDIM Supply(NSources%) AS SupplyType

  I% = 1
  WHILE I% <= NSources%
    DO
      LOCATE PromptRow%, 1
      PRINT "Enter the node no. for supply node"; I%; ":  ";
      INPUT Supply(I%).Num
      CLS
      Exists% = False%
      LookAt% = 1
      WHILE LookAt% <= NNodes% AND NOT Exists%
        Exists% = Supply(I%).Num = NodePointer%(LookAt%)
        LookAt% = LookAt% + 1
      WEND
      Found% = False%
      LookAt% = 1
      WHILE LookAt% < I% AND NOT Found%
        Found% = Supply(I%).Num = Supply(LookAt%).Num
        LookAt% = LookAt% + 1
      WEND
      LookAt% = 1
      WHILE LookAt% <= NSinks% AND NOT Found%
        Found% = Supply(I%).Num = Sink(LookAt%).Num
        LookAt% = LookAt% + 1
      WEND
      CALL GetNode(Supply(I%).Num, East, North, SSEast, SSNorth, El, TS, DS, NodeVuln, DOS%, Suitable%)
      IF NOT Suitable% THEN CALL PrintError("That node is not a suitable supply point location.")
      IF Found% THEN CALL PrintError("That node is already entered.")
      IF NOT Exists% THEN CALL PrintError("That node does not exist.")
    LOOP UNTIL NOT Found% AND Exists% AND Suitable%

    DO
      LOCATE PromptRow%, 1
      PRINT "Enter the supply at node"; Supply(I%).Num
      INPUT Supply(I%).Sup
      CLS
      IF Supply(I%).Sup < 0! THEN CALL PrintError("Negative supplies are not allowed.")
    LOOP UNTIL Supply(I%).Sup >= 0!

    CALL DisplayMessage("Supply node no.:  " + STR$(Supply(I%).Num))
    CALL DisplayMessage("   Supply:  " + STR$(Supply(I%).Sup))

    Answer$ = ""
    DO
      LOCATE PromptRow%, 1
      INPUT "Do you want to make changes (Y/N)?"; Answer$
      CLS
      IF LEN(Answer$) > 0 THEN
        Answer$ = UCASE$(LEFT$(Answer$, 1))
      ELSE
        Answer$ = "Y"
      END IF
    LOOP UNTIL Answer$ = "Y" OR Answer$ = "N"

    IF Answer$ = "N" THEN I% = I% + 1
  WEND

END SUB

SUB PrintNodeNum (NNodes%, NodePointer%(), NSinks%, Demand() AS DemandType)

SHARED MapYMax, MapYMin
ON ERROR GOTO ErrorHandle

  CALLS SETCOLOR(TextColor%)
  STH! = (10! / 782.64) * (MapYMax - MapYMin)
  RadiusY! = (3! / 782.64) * (MapYMax - MapYMin)
  RadiusY! = RadiusY! + (1.5 / 782.64) * (MapYMax - MapYMin)
  CALLS SETSTEXT(STH!, 1!, 0)

  FOR I% = 1 TO NNodes%
    CALL GetNode(NodePointer%(I%), E1, N1, SSE1, SSN1, E11, T1$, D1$, V1, D1%, S1%)
    Num$ = LTRIM$(RTRIM$(STR$(NodePointer%(I%))))
    CALLS INQSTSIZE(Num$, Height, Wdth, Offset)
    Check% = 1
    DemandNode% = False%
    WHILE Check% <= NSinks% AND NOT DemandNode%
      DemandNode% = NodePointer%(I%) = Demand(Check%).Num
      IF DemandNode% THEN Dem = Demand(Check%).Dand
      Check% = Check% + 1
    WEND
    IF DemandNode% THEN
      Dem = ((Dem / 30! + 3!) / 782.64) * (MapYMax - MapYMin)
      Dem = Dem + (1.5 / 782.64) * (MapYMax - MapYMin)
    ELSE
      Dem = RadiusY!
    END IF
    CALLS MOVTCURABS(SSE1 - Wdth, SSN1 - Height - Dem)
    CALLS STEXT(Num$)
  NEXT I%
  CALLS MOVTCURABS(0!, MapYMin - 50!)

END SUB

SUB PutNode (NN%, East, North, SSEast, SSNorth, E1, T$, D$, V, D%, S%)

DIM Temp AS NodeType
ON ERROR GOTO ErrorHandle

  Temp.E = East
  Temp.N = North
  Temp.SSE = SSEast
  Temp.SSN = SSNorth
  Temp.Elev = E1
  Temp.Terr = T$
  Temp.Desc = D$
  Temp.Vuln = V
  Temp.DOS = D%
  Temp.Suitable = S%

  PUT #NodeRND%, NN%, Temp

END SUB
```

```
SUB CostVsPlan (NPlanes%, Problem AS STRING, NSinks%)

ON ERROR GOTO ErrorHandle

  IF NPlanes% > 10 THEN
    Start% = NPlanes% - 9
  ELSE
    Start% = 1
  END IF

  DIM ZTotal(Start% TO NPlanes%), Lag(Start% TO NPlanes%), Loss(Start% TO NPlanes%)

  CALLS WORLDOFF
  CALLS SETVIEWPORT(0!, 0!, 1!, 1!, Background%, Background%)
  CALLS SETSTEXT(50, .5, 0)
  CALLS SETSTCLR(TextColor%, TextColor%)

  Title$ = "PLAN COMPARISON"
  CALLS INQSTSIZE(Title$, H%, W%, 0%)
  CALLS MOVTCURABS(511 - W% \ 2, H%)
  CALLS STEXT(Title$)

  CALLS SETCOLOR(6)
  CALLS BAR(25, 50, 50, 75)
  CALLS MOVTCURABS(50, 75)
  CALLS SETSTEXT(25, .5, 0)
  CALLS STEXT("TOTAL COST")
  CALLS SETCOLOR(NodeColor%)
  CALLS BAR(200, 50, 225, 75)
  CALLS MOVTCURABS(225, 75)
  CALLS STEXT("LOSS")
  CALLS SETCOLOR(LinkColor%)
  CALLS BAR(375, 50, 400, 75)
  CALLS MOVTCURABS(400, 75)
  CALLS STEXT("LAG")
  CALLS SETCOLOR(TextColor%)

  CALLS SETSTEXT(25, .5, 1)
  YLable$ = "COST"
  CALLS INQSTSIZE(YLable$, H%, W%, 0%)
  CALLS MOVTCURABS(H%, 277 - W% \ 2)
  CALLS STEXT(YLable$)

  XLable$ = "PLAN #"
  CALLS SETSTEXT(25, .5, 0)
  CALLS INQSTSIZE(XLable$, H%, W%, 0%)
  CALLS MOVTCURABS(511 - W% \ 2, 527)
  CALLS STEXT(XLable$)

  CALLS MOVABS(0, 527)
  CALLS LNABS(1023, 527)

  CLOSE #PlanFile%
  OPEN Problem + ".PLN" FOR INPUT AS #PlanFile%

  LINE INPUT #PlanFile%, Lin$
  PlanNum% = VAL(Lin$)
  WHILE PlanNum% <> Start%
    DO
      LINE INPUT #PlanFile%, Lin$
      Lin$ = LTRIM$(RTRIM$(Lin$))
    LOOP UNTIL Lin$ = PlanEnd$
    LINE INPUT #PlanFile%, Lin$
    PlanNum% = VAL(Lin$)
  WEND
```

```
CALLS SETSTEXT(25, .5, 0)
CALLS INQSTSIZE(XLables$, H%, W%, 0%)
FOR I% = Start% TO NPlans%
  LINE INPUT #PlanFile%, Lin$
  LINE INPUT #PlanFile%, Lin$
  ZTotal(I%) = VAL(Lin$)
  LINE INPUT #PlanFile%, Lin$
  Lag(I%) = VAL(Lin$)
  LINE INPUT #PlanFile%, Lin$
  Loss(I%) = VAL(Lin$)
  LINE INPUT #PlanFile%, Lin$
  NSources% = VAL(Lin$)
  OpenNode$ = "PLAN " + LTRIM$(RTRIM$(STR$(I%))) + ":  "
  FOR J% = 1 TO NSources%
    LINE INPUT #PlanFile%, Lin$
    SNum% = VAL(Lin$)
    OpenNodes$ = OpenNodes$ + " " + LTRIM$(RTRIM$(STR$(SNum%)))
    LINE INPUT #PlanFile%, Lin$
    FOR K% = 1 TO NSinks%
      LINE INPUT #PlanFile%, Lin$
    NEXT K%
  NEXT J%
  CALLS MOVTCURABS(50, 527 + (NPlans% - I% + 1) * H%)
  CALLS STEXT(OpenNodes$)
  LINE INPUT #PlanFile%, Lin$
  IF I% <> NPlans% THEN LINE INPUT #PlanFile%, Lin$
NEXT I%

CALLS MOVABS(50, 527)
CALLS LNABS(50, 768)
CALLS MOVTCURABS(0, 647)
CALLS STEXT("OPEN")
CALLS MOVTCURABS(0, 648 + H%)
CALLS STEXT("NODES")

MaxY = ZTotal(Start%)
FOR I% = Start% + 1 TO NPlans%
  IF MaxY < ZTotal(I%) THEN MaxY = ZTotal(I%)
NEXT I%

YDiv = MaxY \ 10!
IF YDiv * 10! <> MaxY THEN
  IF YDiv > 1! THEN
    YDiv = YDiv + 1!
    MaxY = YDiv * 10!
  ELSEIF MaxY / 10! > 1! THEN
    YDiv = 2!
    MaxY = 20!
  ELSEIF MaxY / 10! > .5 THEN
    MaxY = 10!
    YDiv = 1!
  ELSEIF MaxY / 10! > .1 THEN
    MaxY = 5!
    YDiv = .5
  ELSEIF MaxY / 10! > .05 THEN
    MaxY = 1!
    YDiv = .1
  ELSE
    MaxY = .5
    YDiv = .05
  END IF
END IF

CALLS SETCOLOR(TextColor%)
FOR I% = 0 TO 10
```

```
    Y! = I% * YDiv
    Y$ = STR$(Y!)
    CALLS INQSTSIZE(Y$, H%, W%, O%)
    CX% = 124 - W% - 3
    CY% = 479 - I% * 404 * (YDiv / MaxY) + H% / 2!
    CALLS MOVTCURABS(CX%, CY%)
    CY% = CY% - H% / 2!
    CALLS MOVABS(124, CY%)
    CALLS LNABS(130, CY%)
    CALLS STEXT(Y$)
  NEXT I%
  CALLS MOVABS(124, 75)
  CALLS LNABS(124, 479)
  CALLS LNABS(1023, 479)
  CALLS MOVTCURABS(0, 0)

  FOR I% = Start% TO NPlane%
    Z% = 479 - ZTotal(I%) / YDiv * 404 * (YDiv / MaxY)
    La% = 479 - Lag(I%) / YDiv * 404 * (YDiv / MaxY)
    Lo% = 479 - Loss(I%) / YDiv * 404 * (YDiv / MaxY)
    C% = 124 + (I% - Start% + 1) * 90 - 45
    CALLS SETCOLOR(6)
    X1% = C% - 30
    X2% = C% - 10
    CALLS BAR(X1%, Z%, X2%, 479)
    CALLS SETCOLOR(NodeColor%)
    X1% = C% - 10
    X2% = C% + 10
    CALLS BAR(X1%, Lo%, X2%, 479)
    CALLS SETCOLOR(LinkColor%)
    X1% = C% + 10
    X2% = C% + 30
    CALLS BAR(X1%, La%, X2%, 479)
    T$ = LTRIM$(RTRIM$(STR$(I%)))
    CALLS INQSTSIZE(T$, H%, W%, O%)
    C% = C% - W% / 2!
    CALLS MOVTCURABS(C%, 507)
    CALLS STEXT(T$)
  NEXT I%

  CLOSE #PlanFile
  OPEN Problem + ".PLN" FOR APPEND AS #PlanFile%

END SUB

SUB InitGraphicsScreen (XMax, XMin, YMax, YMin)

CONST Device$ = "HALO9T.DEV"            'These files must be in the same
CONST Font$ = "HALO104.FNT"             'directory as SPL when exicuted.

ON ERROR GOTO ErrorHandle

  CALLS SETDEV(Device$)
  CALLS SETCRANGE(15)
  CALLS INITGRAPHICS(1)
  CALLS SETFONT(Font$)

  Red% = 16
  Blue% = 16
  Green% = 16
  CALLS SETCPAL(Background%, Red%, Green%, Blue%)
  Red% = 0
  Blue% = 15
  Green% = 0
  CALLS SETCPAL(NodeColor%, Red%, Green%, Blue%)
  Red% = 0
```

```
   BlueX = 0
   GreenX = 0
   CALLS SETCPAL(TextColor%, Red%, Green%, Blue%)
   Red% = 0
   Green% = 15
   Blue% = 0
   CALLS SETCPAL(LinkColor%, Red%, Green%, Blue%)
   Red% = 15
   Green% = 9
   Blue% = 4
   CALLS SETCPAL(DemandNodeColor%, Red%, Green%, Blue%)
   Red% = 15
   Green% = 15
   Blue% = 0
   CALLS SETCPAL(SourceNodeColor%, Red%, Green%, Blue%)
   Red% = 13
   Green% = 0
   Blue% = 15
   CALLS SETCPAL(PathLinkColor%, Red%, Green%, Blue%)
   Red% = 15
   Green% = 0
   Blue% = 0
   CALLS SETCPAL(PathLinkColor% + 1, Red%, Green%, Blue%)

   CALLS SETSTCLR(TextColor%, TextColor%)
   CALLS SETVIEWPORT(0!, 0!, 1!, 1!, Background%, Background%)
   CALLS SETVIEWPORT(0!, 0!, 1!, .98, TextColor%, Background%)
   CALLS SETWORLD(XMin, YMin, XMax, YMax)
   CALLS SETLNSTYLE(1)

END SUB

SUB LagVsSupply (NSources%, NSinks%, Supply() AS SupplyType, Lag(), Problem AS STRING, DTG$, NPlans%)

ON ERROR GOTO ErrorHandle

   CALLS WORLDOFF
   CALLS SETVIEWPORT(0!, 0!, 1!, 1!, Background%, Background%)
   CALLS SETSTEXT(50, .5, 0)
   CALLS SETSTCLR(TextColor%, TextColor%)

   Title$ = "EXPECTED LAG VS SUPPLY POINT"
   CALLS INQSTSIZE(Title$, H%, W%, 0%)
   CALLS MOVTCURABS(511 - W% \ 2, 100)
   CALLS STEXT(Title$)

   Title$ = LTRIM$(RTRIM$(Problem)) + " / " + LTRIM$(RTRIM$(STR$(NPlans%)))
   CALLS SETSTEXT(30, .5, 0)
   CALLS INQSTSIZE(Title$, H%, W%, 0%)
   CALLS MOVTCURABS(124, 135)
   CALLS STEXT(Title$)

   Title$ = DTG$
   CALLS INQSTSIZE(Title$, H%, W%, 0%)
   CALLS MOVTCURABS(900 - W%, 135)
   CALLS STEXT(Title$)

   CALLS SETCOLOR(LinkColor%)
   CALLS BAR(25, 175, 50, 200)
   CALLS MOVTCURABS(50, 200)
   CALLS SETSTEXT(25, .5, 0)
   CALLS STEXT("EXPECTED LAG")
   CALLS SETCOLOR(TextColor%)

   CALLS SETSTEXT(25, .5, 1)
   YLable$ = "COST"
```

```
CALLS INQSTSIZE(YLable$, H%, W%, O%)
CALLS MOVTCURABS(H% + 30, 447 - W% \ 2)
CALLS STEXT(YLable$)

XLable$ = "SUPPLY POINT"
CALLS SETSTEXT(25, .5, 0)
CALLS INQSTSIZE(XLable$, H%, W%, O%)
CALLS MOVTCURABS(511 - W% \ 2, 687)
CALLS STEXT(XLable$)

MaxY = 1!
FOR I% = 1 TO NSources%
  LagTot = 0!
  FOR J% = 1 TO NSinks%
    LagTot = LagTot + Lag(I%, J%)
  NEXT J%
  IF MaxY < LagTot THEN MaxY = LagTot
NEXT I%

YDiv = MaxY \ 10!
IF YDiv * 10! <> MaxY THEN
  IF YDiv > 1! THEN
    YDiv = YDiv + 1!
    MaxY = YDiv * 10!
  ELSEIF MaxY / 10! > 1! THEN
    YDiv = 2!
    MaxY = 20!
  ELSEIF MaxY / 10! > .5 THEN
    MaxY = 10!
    YDiv = 1!
  ELSEIF MaxY / 10! > .1 THEN
    MaxY = 5!
    YDiv = .5
  ELSEIF MaxY / 10! > .05 THEN
    MaxY = 1!
    YDiv = .1
  ELSE
    MaxY = .5
    YDiv = .05
  END IF
END IF

CALLS SETCOLOR(TextColor%)
FOR I% = 0 TO 10
  Y! = I% * YDiv
  Y$ = STR$(Y!)
  CALLS INQSTSIZE(Y$, H%, W%, O%)
  CX% = 124 - W% - 3
  CY% = 649 - I% * 40.4 + H% / 2!
  CALLS MOVTCURABS(CX%, CY%)
  CY% = CY% - H% / 2!
  CALLS MOVABS(124, CY%)
  CALLS LNABS(130, CY%)
  CALLS STEXT(Y$)
NEXT I%
CALLS MOVABS(124, 245)
CALLS LNABS(124, 649)
CALLS LNABS(900, 649)
CALLS MOVTCURABS(0, 0)

B% = (900 - 124) / NSources% - 30
IF B% > 60 THEN B% = 60
FOR I% = 1 TO NSources%
  LagTot = 0!
  FOR J% = 1 TO NSinks%
    LagTot = LagTot + Lag(I%, J%)
    NEXT J%
    Z% = 649 - LagTot / YDiv * 404 * (YDiv / MaxY)
    C% = 124 + (I% - 1) * (B% + 30) + B% / 2! + 15
    CALLS SETCOLOR(LinkColor%)
    X1% = C% - .5 * B% / 3!
    X2% = C% + .5 * B% / 3!
    CALLS BAR(X1%, Z%, X2%, 649)
    T$ = LTRIM$(RTRIM$(STR$(Supply(I%).Num)))
    CALLS INQSTSIZE(T$, H%, W%, O%)
    C% = C% - W% / 2!
    CALLS MOVTCURABS(C%, 677)
    CALLS STEXT(T$)
  NEXT I%

END SUB

SUB LossVsSupply (NSources%, NSinks%, Supply() AS SupplyType, Loss(), Problem AS STRING, DTG$, NPlane%)

ON ERROR GOTO ErrorHandle

  CALLS WORLDOFF
  CALLS SETVIEWPORT(0!, 0!, 1!, 1!, Background%, Background%)
  CALLS SETSTEXT(50, .5, 0)
  CALLS SETSTCLR(TextColor%, TextColor%)

  Title$ = "EXPECTED LOSS VS SUPPLY POINT"
  CALLS INQSTSIZE(Title$, H%, W%, O%)
  CALLS MOVTCURABS(511 - W% \ 2, 100)
  CALLS STEXT(Title$)

  Title$ = LTRIM$(RTRIM$(Problem)) + " / " + LTRIM$(RTRIM$(STR$(NPlane%)))
  CALLS SETSTEXT(30, .5, 0)
  CALLS INQSTSIZE(Title$, H%, W%, O%)
  CALLS MOVTCURABS(124, 135)
  CALLS STEXT(Title$)

  Title$ = DTG$
  CALLS INQSTSIZE(Title$, H%, W%, O%)
  CALLS MOVTCURABS(900 - W%, 135)
  CALLS STEXT(Title$)

  CALLS SETCOLOR(NodeColor%)
  CALLS BAR(25, 175, 50, 200)
  CALLS MOVTCURABS(50, 200)
  CALLS SETSTEXT(25, .5, 0)
  CALLS STEXT("TOTAL LOSS")
  CALLS SETCOLOR(DemandNodeColor%)
  CALLS BAR(200, 175, 225, 200)
  CALLS MOVTCURABS(225, 200)
  CALLS STEXT("LOSS AT SP")
  CALLS SETCOLOR(PathLinkColor%)
  CALLS BAR(375, 175, 400, 200)
  CALLS MOVTCURABS(400, 200)
  CALLS STEXT("LOSS IN TRANSIT")
  CALLS SETCOLOR(TextColor%)

  CALLS SETSTEXT(25, .5, 1)
  YLable$ = "COST"
  CALLS INQSTSIZE(YLable$, H%, W%, O%)
  CALLS MOVTCURABS(H% + 30, 447 - W% \ 2)
  CALLS STEXT(YLable$)

  XLable$ = "SUPPLY POINT"
  CALLS SETSTEXT(25, .5, 0)
  CALLS INQSTSIZE(XLable$, H%, W%, O%)
  CALLS MOVTCURABS(511 - W% \ 2, 697)
CALLS STEXT(XLabel$)

MaxY = 1!
FOR I% = 1 TO NSources%
  LossTot = Loss(I%, 0)
  FOR J% = 1 TO NSinks%
    LossTot = LossTot + Loss(I%, J%)
  NEXT J%
  IF MaxY < LossTot THEN MaxY = LossTot
NEXT I%

YDiv = MaxY \ 10!
IF YDiv * 10! <> MaxY THEN
  IF YDiv > 1! THEN
    YDiv = YDiv + 1!
    MaxY = YDiv * 10!
  ELSEIF MaxY / 10! > 1! THEN
    YDiv = 2!
    MaxY = 20!
  ELSEIF MaxY / 10! > .5 THEN
    MaxY = 10!
    YDiv = 1!
  ELSEIF MaxY / 10! > .1 THEN
    MaxY = 5!
    YDiv = .5
  ELSEIF MaxY / 10! > .05 THEN
    MaxY = 1!
    YDiv = .1
  ELSE
    MaxY = .5
    YDiv = .05
  END IF
END IF

CALLS SETCOLOR(TextColor%)
FOR I% = 0 TO 10
  Y! = I% * YDiv
  Y$ = STR$(Y!)
  CALLS INQSTSIZE(Y$, H%, W%, D%)
  CX% = 124 - W% - 3
  CY% = 649 - I% * 40.4 + H% / 2!
  CALLS MOVTCURABS(CX%, CY%)
  CY% = CY% - H% / 2!
  CALLS MOVABS(124, CY%)
  CALLS LNABS(130, CY%)
  CALLS STEXT(Y$)
NEXT I%
CALLS MOVABS(124, 245)
CALLS LNABS(124, 649)
CALLS LNABS(900, 649)
CALLS MOVTCURABS(0, 0)

B% = (900 - 124) / NSources% - 30
IF B% > 60 THEN B% = 60
FOR I% = 1 TO NSources%
  LossTot = 0!
  TransLoss = 0!
  FOR J% = 1 TO NSinks%
    TransLoss = TransLoss + Loss(I%, J%)
  NEXT J%
  LossTot = TransLoss + Loss(I%, 0)
  Z% = 649 - LossTot / YDiv * 404 * (YDiv / MaxY)
  La% = 649 - TransLoss / YDiv * 404 * (YDiv / MaxY)
  Lo% = 649 - Loss(I%, 0) / YDiv * 404 * (YDiv / MaxY)
  C% = 124 + (I% - 1) * (B% + 30) + B% / 2! + 15
  CALLS SETCOLOR(NodeColor%)
```

```
    X1% = C% - 1.5 * B% / 3!
    X2% = C% - .5 * B% / 3!
    CALLS BAR(X1%, Z%, X2%, 649)
    CALLS SETCOLOR(DemandNodeColor%)
    X1% = C% - .5 * B% / 3!
    X2% = C% + .5 * B% / 3!
    CALLS BAR(X1%, Lo%, X2%, 649)
    CALLS SETCOLOR(PathLinkColor%)
    X1% = C% + .5 * B% / 3!
    X2% = C% + 1.5 * B% / 3!
    CALLS BAR(X1%, La%, X2%, 649)
    T$ = LTRIM$(RTRIM$(STR$(Supply(I%).Num)))
    CALLS INQSTSIZE(T$, H%, W%, O%)
    C% = C% - W% / 2!
    CALLS MOVTCURABS(C%, 677)
    CALLS STEXT(T$)
  NEXT I%

END SUB

SUB MostCostlyUnit (NSources%, NSinks%, Demand() AS DemandType, Lag(), Loss(), Problem AS STRING, DTG$, NPlane%)

ON ERROR GOTO ErrorHandle

  CALLS WORLDOFF
  CALLS SETVIEWPORT(0!, 0!, 1!, 1!, Background%, Background%)
  CALLS SETSTEXT(50, .8, 0)
  CALLS SETSTCLR(TextColor%, TextColor%)

  Title$ = "MOST COSTLY SUPPORTED UNIT"
  CALLS INQSTSIZE(Title$, H%, W%, O%)
  CALLS MOVTCURABS(511 - W% \ 2, 100)
  CALLS STEXT(Title$)

  Title$ = LTRIM$(RTRIM$(Problem)) + " / " + LTRIM$(RTRIM$(STR$(NPlane%)))
  CALLS SETSTEXT(30, .5, 0)
  CALLS INQSTSIZE(Title$, H%, W%, O%)
  CALLS MOVTCURABS(124, 135)
  CALLS STEXT(Title$)

  Title$ = DTG$
  CALLS INQSTSIZE(Title$, H%, W%, O%)
  CALLS MOVTCURABS(900 - W%, 135)
  CALLS STEXT(Title$)

  CALLS SETCOLOR(8)
  CALLS BAR(25, 175, 50, 200)
  CALLS MOVTCURABS(50, 200)
  CALLS SETSTEXT(25, .5, 0)
  CALLS STEXT("TOTAL COST")
  CALLS SETCOLOR(NodeColor%)
  CALLS BAR(200, 175, 225, 200)
  CALLS MOVTCURABS(225, 200)
  CALLS STEXT("LOSS")
  CALLS SETCOLOR(LinkColor%)
  CALLS BAR(375, 175, 400, 200)
  CALLS MOVTCURABS(400, 200)
  CALLS STEXT("LAG")
  CALLS SETCOLOR(TextColor%)

  CALLS SETSTEXT(25, .5, 1)
  YLable$ = "COST"
  CALLS INQSTSIZE(YLable$, H%, W%, O%)
  CALLS MOVTCURABS(H% + 30, 447 - W% \ 2)
  CALLS STEXT(YLable$)
```

```
XLable$ = "SUPPORTED UNIT"
CALLS SETSTEXT(25, .5, 0)
CALLS INQSTSIZE(XLable$, H%, W%, O%)
CALLS MOVTCURABS(511 - W% \ 2, 697)
CALLS STEXT(XLable$)

MaxY = 1!
FOR J% = 1 TO NSinks%
  ZTot = 0!
  FOR I% = 1 TO NSources%
    ZTot = ZTot + Lag(I%, J%) + Loss(I%, J%)
  NEXT I%
  IF MaxY < ZTot THEN MaxY = ZTot
NEXT J%

YDiv = MaxY \ 10!
IF YDiv * 10! <> MaxY THEN
  IF YDiv > 1! THEN
    YDiv = YDiv + 1!
    MaxY = YDiv * 10!
  ELSEIF MaxY / 10! > 1! THEN
    YDiv = 2!
    MaxY = 20!
  ELSEIF MaxY / 10! > .5 THEN
    MaxY = 10!
    YDiv = 1!
  ELSEIF MaxY / 10! > .1 THEN
    MaxY = 5!
    YDiv = .5
  ELSEIF MaxY / 10! > .05 THEN
    MaxY = 1!
    YDiv = .1
  ELSE
    MaxY = .5
    YDiv = .05
  END IF
END IF

CALLS SETCOLOR(TextColor%)
FOR I% = 0 TO 10
  Y! = I% * YDiv
  Y$ = STR$(Y!)
  CALLS INQSTSIZE(Y$, H%, W%, O%)
  CX% = 124 - W% - 3
  CY% = 649 - I% * 40.4 + H% / 2!
  CALLS MOVTCURABS(CX%, CY%)
  CY% = CY% - H% / 2!
  CALLS MOVABS(124, CY%)
  CALLS LNABS(130, CY%)
  CALLS STEXT(Y$)
NEXT I%
CALLS MOVABS(124, 245)
CALLS LNABS(124, 649)
CALLS LNABS(900, 649)
CALLS MOVTCURABS(0, 0)

B% = (900 - 124) / NSinks% - 30
IF B% > 60 THEN B% = 60
FOR J% = 1 TO NSinks%
  ZTot = 0!
  LagTot = 0!
  LossTot = 0!
  FOR I% = 1 TO NSources%
    LagTot = LagTot + Lag(I%, J%)
    LossTot = LossTot + Loss(I%, J%)
  NEXT I%
```

```
      ZTot = LossTot + LagTot
      Z% = 649 - ZTot / YDiv * 404 * (YDiv / MaxY)
      La% = 649 - LagTot / YDiv * 404 * (YDiv / MaxY)
      Lo% = 649 - LossTot / YDiv * 404 * (YDiv / MaxY)
      C% = 124 + (J% - 1) * (B% + 20) + B% / 2! + 15
      CALLS SETCOLOR(6)
      X1% = C% - 1.5 * B% / 3!
      X2% = C% - .5 * B% / 3!
      CALLS BAR(X1%, Z%, X2%, 649)
      CALLS SETCOLOR(NodeColor%)
      X1% = C% - .5 * B% / 3!
      X2% = C% + .5 * B% / 3!
      CALLS BAR(X1%, Lo%, X2%, 649)
      CALLS SETCOLOR(LinkColor%)
      X1% = C% + .5 * B% / 3!
      X2% = C% + 1.5 * B% / 3!
      CALLS BAR(X1%, La%, X2%, 649)
      T$ = LTRIM$(RTRIM$(STR$(Demand(J%).Num)))
      CALLS INQSTSIZE(T$, H%, W%, O%)
      C% = C% - W% / 2!
      CALLS MOVTCURABS(C%, 677)
      CALLS STEXT(T$)
    NEXT J%

END SUB

SUB MostEffSource (NSources%, NSinks%, Supply() AS SupplyType, Lag(), Loss(), Problem AS STRING, DTG$, NPlans%)

ON ERROR GOTO ErrorHandle

    CALLS WORLDOFF
    CALLS SETVIEWPORT(0!, 0!, 1!, 1!, Background%, Background%)
    CALLS SETSTEXT(50, .5, 0)
    CALLS SETSTCLR(TextColor%, TextColor%)

    Title$ = "MOST EFFICIENT SUPPLY POINT"
    CALLS INQSTSIZE(Title$, H%, W%, O%)
    CALLS MOVTCURABS(511 - W% \ 2, 100)
    CALLS STEXT(Title$)

    Title$ = LTRIM$(RTRIM$(Problem)) + " / " + LTRIM$(RTRIM$(STR$(NPlans%)))
    CALLS SETSTEXT(30, .5, 0)
    CALLS INQSTSIZE(Title$, H%, W%, O%)
    CALLS MOVTCURABS(124, 135)
    CALLS STEXT(Title$)

    Title$ = DTG$
    CALLS INQSTSIZE(Title$, H%, W%, O%)
    CALLS MOVTCURABS(900 - W%, 135)
    CALLS STEXT(Title$)

    CALLS SETCOLOR(6)
    CALLS BAR(25, 175, 50, 200)
    CALLS MOVTCURABS(50, 200)
    CALLS SETSTEXT(25, .5, 0)
    CALLS STEXT("TOTAL COST")
    CALLS SETCOLOR(NodeColor%)
    CALLS BAR(200, 175, 225, 200)
    CALLS MOVTCURABS(225, 200)
    CALLS STEXT("LOSS")
    CALLS SETCOLOR(LinkColor%)
    CALLS BAR(375, 175, 400, 200)
    CALLS MOVTCURABS(400, 200)
    CALLS STEXT("LAG")
    CALLS SETCOLOR(TextColor%)
CALLS SETSTEXT(25, .5, 1)
YLable$ = "COST"
CALLS INQSTSIZE(YLable$, H%, W%, O%)
CALLS MOVTCURABS(H% + 30, 447 - W% \ 2)
CALLS STEXT(YLable$)

XLable$ = "SUPPLY POINT"
CALLS SETSTEXT(25, .5, 0)
CALLS INQSTSIZE(XLable$, H%, W%, O%)
CALLS MOVTCURABS(511 - W% \ 2, 697)
CALLS STEXT(XLable$)

MaxY = Loss(1, 0)
FOR I% = 1 TO NSources%
  TCost = Loss(I%, 0)
  FOR J% = 1 TO NSinks%
    TCost = TCost + Lag(I%, J%) + Loss(I%, J%)
  NEXT J%
  IF MaxY < TCost THEN MaxY = TCost
NEXT I%

YDiv = MaxY \ 10!
IF YDiv * 10! <> MaxY THEN
  IF YDiv > 1! THEN
    YDiv = YDiv + 1!
    MaxY = YDiv * 10!
  ELSEIF MaxY / 10 > 1! THEN
    YDiv = 2!
    MaxY = 20!
  ELSEIF MaxY / 10! > .5 THEN
    MaxY = 10!
    YDiv = 1!
  ELSEIF MaxY / 10! > .1 THEN
    MaxY = 5!
    YDiv = .5
  ELSEIF MaxY / 10! > .05 THEN
    MaxY = 1!
    YDiv = .1
  ELSE
    MaxY = .5
    YDiv = .05
  END IF
END IF

CALLS SETCOLOR(TextColor%)
FOR I% = 0 TO 10
  Y! = I% * YDiv
  Y$ = STR$(Y!)
  CALLS INQSTSIZE(Y$, H%, W%, O%)
  CX% = 124 - W% - 3
  CY% = 649 - I% * 40.4 + H% / 2!
  CALLS MOVTCURABS(CX%, CY%)
  CY% = CY% - H% / 2!
  CALLS MOVABS(124, CY%)
  CALLS LNABS(130, CY%)
  CALLS STEXT(Y$)
NEXT I%
CALLS MOVABS(124, 245)
CALLS LNABS(124, 649)
CALLS LNABS(900, 649)
CALLS MOVTCURABS(0, 0)

B% = (900 - 124) / NSources% - 30
IF B% > 60 THEN B% = 60
FOR I% = 1 TO NSources%
  Z% = 649 - (Lag(I%, 0) + Loss(I%, 0)) / YDiv * 404 * (YDiv / MaxY)
```

```
    La% = 649 - Lag(I%, 0) / YDiv * 404 * (YDiv / MaxY)
    Lo% = 649 - Loss(I%, 0) / YDiv * 404 * (YDiv / MaxY)
    C% = 124 + (I% - 1) * (B% + 30) + B% / 2! + 18
    CALLS SETCOLOR(6)
    X1% = C% - 1.5 * B% / 3!
    X2% = C% - .5 * B% / 3!
    CALLS BAR(X1%, Z%, X2%, 649)
    CALLS SETCOLOR(NodeColor%)
    X1% = C% - .5 * B% / 3!
    X2% = C% + .5 * B% / 3!
    CALLS BAR(X1%, Lo%, X2%, 649)
    CALLS SETCOLOR(LinkColor%)
    X1% = C% + .5 * B% / 3!
    X2% = C% + 1.5 * B% / 3!
    CALLS BAR(X1%, La%, X2%, 649)
    T$ = LTRIM$(RTRIM$(STR$(Supply(I%).Num)))
    CALLS INQSTSIZE(T$, H%, W%, O%)
    C% = C% - W% / 2!
    CALLS MOVTCURABS(C%, 677)
    CALLS STEXT(T$)
  NEXT I%

END SUB

SUB ComputeLinkVul (NLinks%)

ON ERROR GOTO ErrorHandle

  FOR I% = 1 TO NLinks%
    CALL GetLink(I%, Hd%, Tl%, L, Rt%, V)
    CALL GetNode(Hd%, E, N, SSE, SSN, E1, T$, D$, V1, D%, S%)
    CALL GetNode(Tl%, E, N, SSE, SSN, E1, T$, D$, V2, D%, S%)
    CALL PutLink(I%, Hd%, Tl%, L, Rt%, (V1 + V2) / 2!)
  NEXT I%

END SUB

SUB DelLink (NLinks%)

  LOCATE PromptRow%, 1
  INPUT "Enter head and tail of the link to delete:  ", Head%, Tail%
  Check% = 1
  Found% = False%
  WHILE Check% <= NLinks% AND NOT Found%
    CALL GetLink(Check%, Hd%, Tl%, L, RC%, V)
    Found% = (Head% = Hd% OR Head% = Tl%) AND (Tail% = Hd% OR Tail% = Tl%)
    Check% = Check% + 1
  WEND
  IF NOT Found% THEN CALL PrintError("That link does not exist.")
  IF NOT Found% THEN EXIT SUB

  FOR I% = Check% TO NLinks%
    CALL GetLink(I%, Hd%, Tl%, Length, RC%, LinkVuln)
    CALL PutLink(I% - 1, Hd%, Tl%, Length, RC%, LinkVuln)
  NEXT I%
  NLinks% = NLinks% - 1

END SUB

SUB DrawLinks (NLinks%)

ON ERROR GOTO ErrorHandle

  CALLS SETCOLOR(LinkColor%)

  FOR I% = 1 TO NLinks%
```

```
    CALL GetLink(I%, Hd%, Tl%, L, C%, V)
    CALL GetNode(Hd%, E, N, SSE1, SSN1, E1, T$, D$, V, D%, S%)
    CALL GetNode(Tl%, E, N, SSE2, SSN2, E1, T$, D$, V, D%, S%)
    CALLS MOVABS(SSE1, SSN1)
    CALLS LNABS(SSE2, SSN2)
  NEXT I%

END SUB

SUB GetLink (I%, Hd%, Tl%, L, Rt%, V)

DIM Temp AS LinkType
ON ERROR GOTO ErrorHandle

  GET #LinkRND%, I%, Temp

  Hd% = Temp.Head
  Tl% = Temp.Tail
  L = Temp.Length
  Rt% = Temp.RType
  V = Temp.Vuln

END SUB

SUB PutLink (I%, Hd%, Tl%, L, Rt%, V)

DIM Temp AS LinkType
ON ERROR GOTO ErrorHandle

  Temp.Head = Hd%
  Temp.Tail = Tl%
  Temp.Length = L
  Temp.RType = Rt%
  Temp.Vuln = V

  PUT #LinkRND%, I%, Temp

END SUB

SUB DisplayMessage (Message$)

SHARED CurrentLine%
ON ERROR GOTO ErrorHandle

  VIEW PRINT
  WHILE LEN(Message$) > 0
    IF LEN(Message$) > MaxCol% - 2 THEN
      CALL SplitLine(Message$, OutputLine$)
    ELSE
      OutputLine$ = Message$
      Message$ = ""
    END IF
    IF CurrentLine% = MaxRow% THEN
      CALL ScrollText(2, MaxCol% - 1, 2, MaxRow% - 1, 1)
      CurrentLine% = MaxRow% - 1
    END IF
    LOCATE CurrentLine%, 2: PRINT OutputLine$
    IF CurrentLine% <= MaxRow% - 1 THEN CurrentLine% = CurrentLine% + 1
  WEND
  VIEW PRINT MaxRow% + 1 TO 25

END SUB

SUB LocAlloc (NSources%, Supply() AS SupplyType, NSinks%, Sink() AS DemandType, W, V, FLOTX1, FLOTY1, FLOTX2, FLOTY2)

CONST Spd = 52! * 24!
CONST DOS% = 3
ON ERROR GOTO ErrorHandle

  VIEW PRINT
  CLS
  LOCATE 4, 18: PRINT "----------------------------------------"
  LOCATE 5, 18: PRINT "This  routine  solves  the  uncapacitated"
  LOCATE 6, 18: PRINT "Location-Allocation  problem heuristically"
  LOCATE 7, 18: PRINT "using Euclidean distances."
  LOCATE 8, 18: PRINT "----------------------------------------"
  CALL Waiting("Press any key to continue.", 24, 0, R$)
  CLS

  DIM Y%(NSources%, NSinks%)       'Flag set to True if Source I serves Sink J.
  DIM DW(NSinks%)                  'The demand weight at Sink J.
  DIM C(NSources%, NSinks%)        'The cost / unit distance from source I to
                                   'sink J.
  DIM D(NSources%, NSinks%)        'The distance for Source I to Sink J.
  DIM X(NSources%), Y(NSources%)   'The location of Source I.
  DIM X1(NSinks%), Y1(NSinks%)     'The location of Sink J.

  Lamda% = 1            'Number of iterations.
  ZLamda = 999991       'The total cost for iteration Lamda%.
  Eps = .01             'If Z(Lamda - 1) - Z(Lamda) <= Eps then supply points
                        'have been located.

  PRINT TAB(18); "Computing Heuristic Solution ...... Please Wait."

  FOR I% = 1 TO NSinks%
    CALL GetNode(Sink(I%).Num, X1(I%), Y1(I%), SSE, SSN, E1, T$, D$, V1, D%, S%)
    DW(I%) = Sink(I%).Dmnd
  NEXT I%

  'Make initial assignments.
  SinksPerSource% = NSinks% \ NSources%
  FOR I% = 1 TO NSources%
    FOR J% = SinksPerSource% * (I% - 1) + 1 TO I% * SinksPerSource%
      Y%(I%, J%) = 1
    NEXT J%
  NEXT I%
  Y%(NSources%, NSinks%) = 1

  'Get the initial supply point locations.
  FOR I% = 1 TO NSources%
    CALL GetNode(Supply(I%).Num, X(I%), Y(I%), SSE, SSN, E1, T$, D$, V1, D%, S%)
  NEXT I%

Step2:
  'Compute Euclidean distances and costs.
  AABS = SQR((FLOTX2 - FLOTX1) ^ 2 + (FLOTY2 - FLOTY1) ^ 2)
  FOR I% = 1 TO NSources%
    BABS = SQR((X(I%) - FLOTX1) ^ 2 + (Y(I%) - FLOTY1) ^ 2)
    IF BABS = 0! THEN BABS = .0001
    ADotB = (FLOTX2 - FLOTX1) * (X(I%) - FLOTX1) + (FLOTY2 - FLOTY1) * (Y(I%) - FLOTY1)
    IF ADotB = 0! THEN ADotB = .0001
    Angle = ATN(SQR(1! / (ADotB / (AABS * BABS)) ^ 2 - 1!))
    VulnI = EXP(-VulParam * (.001 * BABS * SIN(Angle)))
    FOR J% = 1 TO NSinks%
      BABS = SQR((X1(J%) - FLOTX1) ^ 2 + (Y1(J%) - FLOTY1) ^ 2)
      IF BABS = 0! THEN BABS = .0001
      ADotB = (FLOTX2 - FLOTX1) * (X1(J%) - FLOTX1) + (FLOTY2 - FLOTY1) * (Y1(J%) - FLOTY1)
      IF ADotB = 0! THEN ADotB = .0001
      Angle = ATN(SQR(1! / (ADotB / (AABS * BABS)) ^ 2 - 1!))
      VulnJ = EXP(-VulParam * (.001 * BABS * SIN(Angle)))
      D(I%, J%) = SQR((X(I%) - X1(J%)) ^ 2 + (Y(I%) - Y1(J%)) ^ 2)
      QIJ = (VulnI + VulnJ) / 2!
      C(I%, J%) = (W + V * QIJ) / Spd + V * VulnI * DOS%
    NEXT J%
  NEXT I%

Step3:
  'Make new assignments based on locations found in the previous iteration.
  FOR J% = 1 TO NSinks%
    MinCost = C(1, J%) * DW(J%) * D(1, J%)
    A% = 1
    FOR I% = 2 TO NSources%
      Cost = C(I%, J%) * DW(J%) * D(I%, J%)
      IF Cost <= MinCost THEN
        MinCost = Cost
        A% = I%
      END IF
      Y%(I%, J%) = 0
    NEXT I%
    Y%(A%, J%) = Lamda%
  NEXT J%

Step4:
  DO
    DO
      'Compute new locations for the sources.
      FOR I% = 1 TO NSources%
        B = 0!
        S = 0!
        C = 0!
        FOR J% = 1 TO NSinks%
          IF Y%(I%, J%) THEN
            IF D(I%, J%) = 0! THEN D(I%, J%) = .001
            B = (C(I%, J%) * DW(J%) * X1(J%)) / D(I%, J%) + B
            C = (C(I%, J%) * DW(J%) * Y1(J%)) / D(I%, J%) + C
            S = (C(I%, J%) * DW(J%)) / D(I%, J%) + S
          END IF
        NEXT J%
        IF S <> 0! THEN
          X(I%) = B / S
          Y(I%) = C / S
        END IF
      NEXT I%

Step5:
      'Compute Euclidean distances and costs.
      AABS = SQR((FLOTX2 - FLOTX1) ^ 2 + (FLOTY2 - FLOTY1) ^ 2)
      FOR I% = 1 TO NSources%
        BABS = SQR((X(I%) - FLOTX1) ^ 2 + (Y(I%) - FLOTY1) ^ 2)
        IF BABS = 0! THEN BABS = .0001
        ADotB = (FLOTX2 - FLOTX1) * (X(I%) - FLOTX1) + (FLOTY2 - FLOTY1) * (Y(I%) - FLOTY1)
        IF ADotB = 0! THEN ADotB = .0001
        Angle = ATN(SQR(1! / (ADotB / (AABS * BABS)) ^ 2 - 1!))
        VulnI = EXP(-VulParam * (.001 * BABS * SIN(Angle)))
        FOR J% = 1 TO NSinks%
          BABS = SQR((X1(J%) - FLOTX1) ^ 2 + (Y1(J%) - FLOTY1) ^ 2)
          IF BABS = 0! THEN BABS = .0001
          ADotB = (FLOTX2 - FLOTX1) * (X1(J%) - FLOTX1) + (FLOTY2 - FLOTY1) * (Y1(J%) - FLOTY1)
          IF ADotB = 0! THEN ADotB = .0001
          Angle = ATN(SQR(1! / (ADotB / (AABS * BABS)) ^ 2 - 1!))
          VulnJ = EXP(-VulParam * (.001 * BABS * SIN(Angle)))
          D(I%, J%) = SQR((X(I%) - X1(J%)) ^ 2 + (Y(I%) - Y1(J%)) ^ 2)
          QIJ = (VulnI + VulnJ) / 2!
          C(I%, J%) = (W + V * QIJ) / Spd + V * VulnI * DOS%
        NEXT J%
      NEXT I%

Step6:
      'Compute the objective function Z for the current iteration.
      ZLamdaM1 = ZLamda
      ZLamda = 0
      FOR I% = 1 TO NSources%
        FOR J% = 1 TO NSinks%
          IF Y%(I%, J%) THEN ZLamda = C(I%, J%) * DW(J%) * D(I%, J%) + ZLamda
        NEXT J%
      NEXT I%
      DeltaZ = ZLamdaM1 - ZLamda

    LOOP UNTIL DeltaZ >= 0! AND DeltaZ <= Eps

    Lamda% = Lamda% + 1

Step7:
    'Make new assignments based on locations found in the previous iteration.
    FOR J% = 1 TO NSinks%
      MinCost = C(1, J%) * DW(J%) * D(1, J%)
      A% = 1
      FOR I% = 2 TO NSources%
        Cost = C(I%, J%) * DW(J%) * D(I%, J%)
        IF Cost <= MinCost THEN
          MinCost = Cost
          A% = I%
        END IF
      NEXT I%
      Y%(A%, J%) = Lamda%
    NEXT J%

Step8:
    'Check to see if assignments have changed.
    Changed% = False%
    FOR I% = 1 TO NSources%
      FOR J% = 1 TO NSinks%
        IF Y%(I%, J%) <> Lamda% AND Y%(I%, J%) <> 0 THEN
          Changed% = True%
          Y%(I%, J%) = 0
        END IF
      NEXT J%
    NEXT I%
    PRINT Changed%

  LOOP UNTIL NOT Changed%

  'Display the results.
  CLS
  LOCATE 3, 32: PRINT "-------------------------"
  LOCATE 4, 32: PRINT " Recommended Open Nodes"
  LOCATE 5, 32: PRINT "-------------------------"
  LOCATE 9, 30: PRINT "New"
  LOCATE 10, 28: PRINT "Location"; TAB(45); "Vicinity Grid"
  LOCATE 11, 28: PRINT "---------"; TAB(45); "-------------"
  FOR I% = 1 TO NSources%
    IF X(I%) >= 100000! THEN X(I%) = X(I%) - 100000!
    IF Y(I%) >= 100000! THEN Y(I%) = Y(I%) - 100000!
    PRINT TAB(29); I%; TAB(45); INT(X(I%)); "/"; INT(Y(I%))
  NEXT I%
  CALL Waiting("Press <Enter> to continue...", 23, 5, R$)

  VIEW PRINT MaxRow% + 1 TO 25

END SUB

SUB NameProblem (Problem AS STRING)

ON ERROR GOTO ErrorHandle
```

```
  DO
    LOCATE PromptRow%, 1
    INPUT "Enter problem name:  ", Problem
    Problem = UCASE$(Problem)
    CLS
    IF INSTR(Problem, ".") > 0 THEN CALL PrintError("Do not use a '.' in the problem name.")
  LOOP UNTIL LEN(Problem) > 0 AND INSTR(Problem, ".") = 0
  CALL DisplayMessage("Problem name:  " + Problem)

END SUB

SUB PrintError (EString$)

ON ERROR GOTO ErrorHandle

  COLOR ErrorColor%
  LOCATE ErrorMessRow%, ErrorMessCol%
  PRINT EString$
  COLOR NormalColor%

END SUB

SUB RiskGuide (W, V)

ON ERROR GOTO ErrorHandle

  DO
    LOCATE PromptRow%, 1
    INPUT "Absolute or Relative (Hit Return for defaults) (A/R)"; Answer$
    CLS
    IF LEN(Answer$) > 0 THEN Answer$ = UCASE$(LEFT$(Answer$, 1))
  LOOP UNTIL Answer$ = "A" OR Answer$ = "R" OR Answer$ = ""

  IF Answer$ = "A" THEN
    DO
      LOCATE PromptRow%, 1
      INPUT "Enter the values for W and V:  ", W, V
      CLS
      IF W < 0! OR V < 0! THEN CALL PrintError("Both values must be non-negative.")
    LOOP UNTIL W >= 0! AND V >= 0! AND (W <> 0! OR V <> 0!)
  ELSEIF Answer$ = "R" THEN
    DO
      LOCATE PromptRow%, 1
      INPUT "Enter the value of W (.01 <= W <= .10):  ", W
      V = 1! - W
    LOOP UNTIL .9 <= V AND V <= .99
  ELSE
    W = 1!
    V = 1!
  END IF
  CLS

END SUB

SUB ScrollText (MinC%, MaxC%, MinR%, MaxR%, N%)

'       SUB ScrollText uses a machine call to the BIOS interrupt that
'       scrolls text in a specified window to scroll the contents of
'       that window.  This service is interrupt 10h, service 6.

CONST HighByte% = 256, LowByte% = 1
CONST ScrollService% = 6

DIM InReg AS RegTypeX, OutReg AS RegTypeX
ON ERROR GOTO ErrorHandle
```

```
  InReg.ax = HighByte% * ScrollService%         'Select BIOS service
  InReg.ax = InReg.ax + LowByte% * N%           'Number of lines to scroll
  InReg.bx = HighByte% * 2                      'Set color of new lines
  InReg.cx = HighByte% * (MinR% - 1)            'Upper left of window
  InReg.cx = InReg.cx + LowByte% * (MinC% - 1)
  InReg.dx = HighByte% * (MaxR% - 1)            'Lower right of window
  InReg.dx = InReg.dx + LowByte% * (MaxC% - 1)
  INTRRPT% = 16                                 'BIOS interrupt number

  CALL INTERRUPTX(INTRRPT%, InReg, OutReg)      'Call BIOS interrupt

END SUB

SUB SplitLine (Message$, OutputLine$)

ON ERROR GOTO ErrorHandle

  MaxLen% = MaxCol% - 2
  I% = MaxLen%
  Ok% = MID$(Message$, I%, 1) = " "
  WHILE NOT Ok% AND I% <> 0
    I% = I% - 1
    IF I% <> 0 THEN Ok% = MID$(Message$, I%, 1) = " "
  WEND
  IF I% <> 0 THEN Ok% = MID$(Message$, I%, 1) <> " "
  WHILE NOT Ok% AND I% <> 0
    I% = I% - 1
    IF I% <> 0 THEN Ok% = MID$(Message$, I%, 1) <> " "
  WEND
  IF I% = 0 THEN
    OutputLine$ = LEFT$(Message$, MaxLen%)
    Message$ = RIGHT$(Message$, LEN(Message$) - MaxLen%)
  ELSE
    OutputLine$ = LEFT$(Message$, I%)
    Message$ = LTRIM$(RIGHT$(Message$, LEN(Message$) - I%))
  END IF

END SUB

SUB Waiting (W$, WRow%, WCol%, KeyStroke$)

ON ERROR GOTO ErrorHandle

  IF WCol% > 0 THEN
    LOCATE WRow%, WCol%
  ELSE
    LOCATE WRow%, (80 - LEN(W$)) \ 2
  END IF
  PRINT W$

  KeyStroke$ = ""
  DO
    KeyStroke$ = INKEY$
  LOOP UNTIL KeyStroke$ <> ""

  IF WCol% > 0 THEN
    LOCATE WRow%, WCol%
  ELSE
    LOCATE WRow%, (80 - LEN(W$)) \ 2
  END IF
  PRINT SPACE$(LEN(W$))

END SUB
```

APPENDIX II

TEST PROBLEM - OPERATIONS ORDER AND SPECIAL SITUATION

The attached operations order and special situation were used to test the implementation of the SPL and is a modification of a problem used by the Combined Arms Services Staff School, Fort Leavenworth, Kansas.

(CLASSIFICATION)

Copy 5 of 24 copies
52d Inf Div (Mech)
Effingham, Kansas
21 November 19____
DKL 28

OPORD 37-88

REF: Leavenworth-Valley Falls map sheet, 50-310, 1:50,000

TIME ZONE USED THROUGHOUT THE ORDER: SIERRA

TASK ORGANIZATION:

<u>1st Bde</u>	<u>2d Bde</u>	<u>3d Bde</u>	<u>52d CAB</u>
1-77 Mech	1-2 Armor	2-141 Mech	151 Atk Hel
1-78 Mech	1-3 Armor	2-142 Mech	153 Atk Hel
1-4 Armor	1-79 Mech	1-5 Armor	1-23 Cav
1-40 FA (DS)	1-41 FA (DS)	2-71 FA (DS)	52 TAMC
1st FSB	2d FSB	3d FSB	

<u>DIVARTY</u>	<u>DIV Troops</u>	<u>DISCOM</u>
2-41 FA (GS)	1-441 ADA	HHC/MMC
52d MLRS Btry	52d Engr	1st MSB
	52d MI	52d Div Band
	52d Sig	
	52d MP Co	
	52d Chem Co	
	508th Engr	

1. <u>SITUATION</u>

a. Enemy Forces.

(1) Airborne elements of suspected battalion size have been reported southwest of Kansas City, Kansas. Smaller elements may be located further west.

(2) Intelligence indicates that enemy forces will attack from the south between Kansas City and Choke Mountain to link up with airborne assault forces.

(3) ANNEX A, Intelligence.

b. Friendly Forces.

(1) 14th TAF furnishes close air support (CAS) to 52d Mech Div, priority to 1st Bde.

(2) 54th Inf Div (Mech) defends in sector to the west.

(3) 14th U.S. Corps conducts offensive operations to the east in Iowa and northwestern Missouri.

c. Attachments/Detachments. 508th Engr Bn (Cbt) with separate engineer companies remain attached.

2. MISSION. Commencing 240001 Nov 88 52D Inf Div (Mech) defends in sector along PL BLUE. On order delays in sector and defends along PL WHITE. Be prepared to conduct offensive operations.

3. EXECUTION.

a. Concept of Operation, ANNEX C, Operations Overlay.

(1) Maneuver: 1st, 2d, and 3d Bde defend along PL White. 52d CAB conducts covering force operations and on order becomes Div reserve. 2d Bde be prepared to reinforce 52d CAB in covering force operations. On order division delays to PL WHITE and defends. On order division delays to PL RED and defends.

(2) Fires. Priority of air support and FA fires to 1st Bde. ANNEX F, Fire Support. Nuclear fires planned but used only on order from Division Commander.

b. 1st Bde. Defend along PL BLUE in left sector.

c. 2d Bde. Defend along PL BLUE in center sector.

d. 3d Bde.

(1) Defend along PL BLUE in right sector.

(2) Be prepared to reinforce DISCOM/DSA with one Mech Infantry Company.

e. 52d CAB.

(1) Conduct covering force battle over entire division front.

(2) Upon withdrawal from CFA become Div reserve.

(3) Screen Div right flank.

f. DIVARTY. See ANNEX F, Fire Support.

g. 52d Engr Bn. Priority to 1st Bde.

h. DISCOM. Locate DSA in Div rear behind PL ORANGE. Priority of resupply is CL V. Provide defense of DSA.

j. Coordinating Instructions.

(1) All units report support locations ASAP.

(2) EEEI. What is enemy location, disposition, strength, and capability?

(3) Avoid damage to civilian property, particularly the city of Leavenworth, Kansas.

(4) Both federal prisons vicinity Leavenworth, Kansas remain occupied by prison guard forces and inmates.

(5) Operation Exposure Guidance: RS-0

4. SERVICE SUPPORT. ANNEX D, (Service Support)

5. COMMAND AND SIGNAL. ANNEX G, (Communications-Electronics). CEOI 6-8.

Acknowledge.

STARS
MG

OFFICIAL

/s/Alexander
ALEXANDER
G3

Annexes.
- A - Intelligence (Omitted)
- B - Not Used
- C - Operations Overlay
- D - Service Support (Omitted)
- E - Engineer (Omitted)
- F - Fire Support (Omitted)
- G - Communications-Electronics (Omitted)

Distribution: A
+ 508th Engr

ADDITIONAL CONSIDERATIONS

1. During November the soil is cold with a mean temperature above 38 degrees F. The soils are generally Aridisols-Argids and dry. Sun is out 10 hours per day and rain average is 1 inch for the month. Mean temperatures range from a daily low of 28 degrees F. to a high of 65 degrees F. with a mean of 42 degrees F. The area consists mainly of rolling plains and grasslands used primarily for agriculture; grain and grazing. November has been an exceptionally dry month with no rain and all temperatures 5-10 degrees above normal. Visibility is unlimited, surface winds are negligible and the moon is the same as today's date. The Missouri River is low due to the previous 4 dry months and flows at 1.6 mps with a temperature of 42 degrees F.

2. Washes are trafficable to tracks but not to most wheeled vehicles. Unimproved light duty roads are initially trafficable to tracks and wheels. Trails are passable to all tracks and wheeled vehicles up to 5 tons.

3. Light Data:

BMNT	0638	MOONRISE	1927
SUNRISE	0724	MOONSET	0110
SUNSET	1655		
EENT	1750		

4. The population is sparse, averaging less than 5 per square mile. The two primary population centers are Kansas City (pop. 2.1 million) and Leavenworth (pop. 32,000).

5. All Division units are organized at ALO 2 and are at full strength at that level.

6. The Division's resupply is being replenished at a rate that provides a three day basic load of POL, ammunition, rations and other classes of supply. Only normally stocked PLL and ASL items are available during the operation. Class IV items are available.

7. Water is plentiful from the Missouri River. The Division water supply is being furnished from 3 point along the Missouri River. Water can be produced at 1500 GPH from each of the water points in operation.

8. The G4 desires to move the Division Main headquarters and support command into the Division Rear Area as soon as practicable. Once the Division forward elements have occupied positions along PL Blue movement of Admin/Log elements will begin.

9. Rear Area Protection (RAP). Because of the unfavorable force ratios no dedicated security force will be assigned to the Division Rear. The DSA/MMC must provide its own security to react to level I and II threats. A security force (1 Rifle Co) will react within 30 minutes in the event of a level III threat.

Map 2-1: Special Situation Operations Map

REFERENCES

1. Anderson, Monty J., A Prototype Decision Support System for the Location of Military Water Points, Master's Thesis, Georgia Institute of Technology, Atlanta, GA, 1980.

2. Balinski, M.L., and H. Mills, "A Warehouse Problem," prepared for: Veteran's Administration; MATHEMATICA, Princeton, New Jersey, April 1960.

3. Ballou, Ronald H., "Locating Warehouses in a Logistics System," The Logistics Review, Vol 4, No. 19, Pg. 23-40, 1971

4. Banks, Jerry, J.P. Spoerer and R.L. Collins, IBM PC Applications for the Industrial Engineer and Manager, Englewood Cliffs, New Jersey: Prentice-Hall, 1986.

5. BASIC Personal Computer Hardware Reference Library, International Business Machines Corporation, P.O. Box 1328-C, Boca Raton, Florida, 33432, 1981.

6. Baumol, W.J., and P. Wolfe, "A Warehouse Location Problem," Operations Research, Vol 6, March-April 1958.

7. Combat Service Support, Department of the Army Field Manual, FM 100-10, March, 1983.

8. Combat Service Support Operations - Division, Department of the Army Field Manual, FM 63-2, November, 1983.

9. Cooper, Leon, "Heuristic Methods for Location Allocation Problems," Siam Review, Vol. 6, Pg. 94-108, 1964.

10. Cooper, Leon, "The Transportation-Location Problem," Operations Research, Vol. 20, No. 1, Pg. 94-108, 1972.

11. Cox, Michael J. and Kathleen Broell Sullivan, Structuring Programs in Microsoft BASIC, Boston: Boyd and Fraser Publishing Co., 1986.

12. "Draft Concept for Support of Land Combat with Digital Terrain Data," by LTC Richard Johnson, Executive Director, Defense Mapping Agency, Washington, D.C., July 1987.

13. Eilon, S., C.D.T. Watson-Gandy, and N. Christofides, Distribution Management: Mathematical Modeling and Practical Analysis. New York: Hafner, 1971.

14. Ellwein, L.B. and P.L. Gray, "Solving Fixed Charge Location-Allocation Problems with Capacity and Configuration Constraints," AIIE Transactions, Vol. 4, Pg. 290-299, 1971.

15. Feldman, E., F.A. Lehrer, and T.L. Ray, "Warehouse Location Under Continuous Economies of Scale," Management Science, Vol. 12, Pg. 670-684, 1966.

16. Francis, R.L. and J.A. White, Facility Layout and Location, Englewood Cliffs, NJ: Prentice-Hall, 1974.

17. Geoffrion, A.M. "A Guide to Computer-Assisted Methods for Distribution Systems Planning," Sloan Management Review, Vol. 16, Pg. 17-42, 1975.

18. Geoffrion, A.M., "Lagrangian Relaxation and its Uses in Integer Programming," Mathematical Programming Study, Vol. 2, Pg. 82-114, 1974.

19. Geoffrion, A.M. and G.W. Graves, "Multicommodity Distribution System Design by Benders Partition," Management Science, Vol. 20, Pg. 822-844, 1971.

20. Geoffrion, A.M and R. McBride, "Lagrangian Relaxation Applied to Capacitated Facility Location Problems," AIIE Transactions, Vol. 10, Pg. 40-47, 1977.

21. Glover, Fred, and Darwin Klingman, "Network Application in Industry and Government," AIIE Transactions, Vol. 9, pp. 363-376, 1977.

22. Guide for DSS Development, U.S. Army Information System Engineering Command, prepared by School of Industrial and Systems Engineering, Georgia Institute of Technology, Vol. 1, 1988.

23. Handler, Gabriel Y. and Pitu B. Mirchandani, Location on Networks Theory and Algorithms, Cambridge, MA: The MIT Press, 1979.

24. HALO 88 Graphics Kernel and Device Drivers Version 1.00.02 Functional Description Manual, Media Cybernetics, Inc., 8484 Georgia Avenue, Suite 200, Silver Spring, Maryland 20910, 1988.

25. Kuehn, A.A., and M.J. Hamburger, "A Heuristic Program for Locating Warehouses," Management Science, Vol. 9, pp. 645-666, 1963.

26. MacKinnon, Ross D., and G.M. Barber, "A New Approach to Network Generation and Map Representation: The Linear Case of the Location Allocation Problem," University of Toronto - York University Joint Program in Transportation, Research Report No. 3, 1971.

27. Mittra, Sitansu S., Decision Support Systems Tools and Techniques, New York: John Wiley and Sons, 1986.

28. Manne, A.S., "Plant Location Under Economies of Scale-Decentralization and Computation," Management Science, Vol. 11, Pg. 213-235, 1964.

29. Nagelhout, Robert V., and Gerald L. Thompson, "A Cost Operator Approach to Multistage Location-Allocation," Management Sciences Research Group, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburg: July, 1979.

30. Preston, C.P. Jr., "Interdiction of a Transportation Network," Master's Thesis, Naval Post Graduate School, Monterey, California, Pg. 8-10, 1972.

31. "Supply Point Locator," An In-Process Review submitted by Dr. Donovan Young, School of Industrial and Systems Engineering, Georgia Institute of Technology to the U.S. Army Institute for Research in Management Information and Computer Science, Atlanta, GA, June, 1987.

32. Toone, Joseph W., and Susan Titmas, "Introduction to JTIDS," Signal, Vol. 41, no. 12, pg. 55, 1987.

33. Kudla, Nancy R., "Artificial Intelligence and C^3I Reporting," Signal, Vol. 41, no. 8, pg 53, 1987.

34. Link, Patrick F., and Leslie G. Callahan, A Prototype Decision Support System for the Selection of Ammunition Transfer Points Based of Field Artillery Role Assignments - Final Summary Report, Technical Report RD-CR-83-2, Georgia Institute of Technology, Atlanta, GA, 1982.

35. Rardin, R.L. and U. Choe, "Tighter Relaxations of Fixed Charge Network Flow Problems," Georgia Institute of Technology, Industrial and Systems Engineering Report Series #J-79-18, May 1979.

36. Rardin, R.L. and V.E. Unger, "Solving Fixed Charge Network Problems with Group Theory-Based Penalties," Naval Research Logistics Quarterly, Vol. 23, No. 1, March 1976.

37. Taha, Hamdy A., Operations Research, an Introduction, 3d Edition, New York: Macmillan Publishing Company, Inc., 1982, pg 158-164.

38. Wagner, Harvey M., Principles of Operations Research, 2d Edition, Englewood Cliffs, New Jersery, 1975, pg 224.

39. Winston, Wayne L., Operations Research: Applications and Algorithms, Boston: Duxbury Press, 1987.