INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI

A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor MI 48106-1346 USA
313/761-4700 800/521-0600

USING SIMULATED ANNEALING TO MINIMIZE THE COST OF MULTI POINT LINES IN CENTRALIZED COMPUTER NETWORKS. IMPLEMENTATION FOR WINDOWS 3.1™

Daniel B. Tomiuk

A Thesis
In
The Faculty
of
Commerce and Administration

Presented in Partial Fulfilment of the Requirements
for the Degree of Master of Science in Administration at
Concordia University
Montreal, Quebec, Canada

October 1996

ABSTRACT

Using Simulated Annealing to Minimize the Cost of Multipoint Lines in Centralized Computer Networks. Implementation for Windows 3.1™

Daniel B. Tomiuk

We focus on a problem encountered when designing centralized telecommunications networks, namely, the *terminal layout* problem. Given each terminal's geographical location, the problem consists in creating multipoint lines rooted at a central site (typically a concentrator) in order to save on connection costs. Well-known multipoint line topologies are the tree, the bus, and the loop. When terminals are assigned a weight representing the average traffic amount exchanged with the central site and lines are constrained by the amount of traffic they can carry, the tree-topology problem is referred to as the *Capacitated Minimum Spanning Tree (CMST)* problem. Algorithms that generate solutions for CMST problems create tree structured networks but can also be used to produce bus structured networks by imposing additional constraints. As for the loop topology, the problem is analogous to the *Vehicle Routing* problem found in Operation Research.

These problems are NP-Complete. Finding an optimal solution in an acceptable amount of time is, therefore, unlikely due to the exponential growth in complexity relative to problem size. Nevertheless, techniques yielding exact solutions have been developed but are limited to networks of no more than, say, 50 terminals. Alternatively, heuristics solve the problem to near-optimality with acceptable computational effort.

We designed applications with graphical output capabilities for Windows 3.1™ using simulated annealing (SA) in an attempt to improve upon well-known heuristic solutions. Our SA programs are presented along with computational results on data sets containing up to 250 terminals. Results are evaluated and compared with those obtained with other heuristic methods.

ACKNOWLEDGEMENTS.

The author wishes to thank first and foremost Professor Jean-Marie Bourjolly for his encouragement, guidance, continuous support, and the time he set aside for the valuable weekly discussions pertaining to this work and without whose help this thesis would have not been possible. I would also like to express my deepest gratitude to Professor Samuel Pierre for his suggestion of the research topic and his valuable help prior to and during this work. In addition, I would also like to thank Professor Mohan Gopalakrishnan for his help during my studies at Concordia and his input towards this thesis.

I kindly thank my friend, Gene Kapantow for his encouragement, valuable suggestions, and for his uncanny ability to 'surf the net' and find solutions to some programming problems I encountered when implementing the programs in Delphi 1.0™.

I thank the administrative staff of the M.Sc. program and particularly convey my deepest gratitude to Mrs. Heather Thomson and to Theresa Sarazin-Wadey for their patience, understanding, and all their help during my studies.

Finally, I am very grateful to my family for their support and understanding

TABLE OF CONTENTS

List of Figures:

Figure

List of Tables:

Table

CHAPTER 1 : INTRODUCTION, BASIC CONCEPTS, AND DEFINITIONS.

1.1 - Introduction.

Centralized computer networks contain a large number of components interconnected together. Basically these components represent terminals and a central site, while the connection links depict how these components are interconnected using various types of data communication medium, generally referred to as 'lines'.

The terminals exchange data among themselves and with the central site which can be either the central computer or a line-concentrating device connected to it. This exchange of information consists of sending and receiving messages over various types of medium intermittently with a transmission time usually lasting no more than a few seconds. By knowing the average amount of dialogue *(traffic or weight)* between each terminal and the central computer, it becomes possible to interconnect these components at a minimal cost given the type of network configuration desired and data transmission capabilities of the medium.

When the data communication medium linking network components is a line (a wire or a cable as opposed to radiated media such as satellite technology), the cost of the network is a function of the aggregate geographical distance the connection links must span in order for each terminal to be able to exchange data with the central site. If the amount of data exchanged between a terminal and the central site is quite large, the terminal and the host are typically connected in a *point-to-point* fashion, where the terminal completely monopolizes the transmission line connecting it to the central site.

However, if this is not the case, economies of scale dictate that in order to take advantage of commercially available lines whose maximum capacities are typically gauged with discrete values such as 2400, 9600, 14400, 28800, 57600 bits per second, one connects several computers onto a 'high-capacity' line, where the line is shared by several terminals at once. Lines structured in such a way are called *multidrop* or *multipoint* lines. Since a weight is associated with each terminal and a constraint is typically imposed on the capacity of transmitted data the lines can carry, the design of multipoint lines gives rise to a computationally complex problem called the *terminal layout problem.* For terminals connected to a central site using a multipoint line, three general types of line arrangements are possible. These are referred to as 'topologies'. A '*tree*' topology is a multipoint line where terminals are arranged hierarchically, a '*bus*' contains terminals arranged sequentially in a line, while '*loops*' are similar to buses but have both extremes connected to the central site instead of just one. Alternatively, when each terminal is directly linked to the central site using a point-to-point connection the network topology is called a '*star*'.

This thesis focuses on the adaptation of Simulated Annealing to the terminal layout problem in centralized computer networks. We introduce a set of programs implemented under the Windows 3.1™ environment that use the simulated annealing process to improve upon results obtained by certain heuristic resolution algorithms. These programs were created to be user-friendly and offer the user the capability of displaying graphical results in order to visualize the network components and its connections.

1.2 - Problem Statement.

The focus of this thesis is on a subproblem of network topological design typically referred to as the *terminal-layout problem.* Once the number and locations of concentrators are known, and the optimal interconnection configuration of these concentrators to the central computer is established, we must determine how to interconnect every terminal in the cluster of terminals assigned to each concentrator. Terminals can be either connected in a point-to-point or multipoint fashion. Multipoint lines, also known as multidrop lines, consist of many terminals linked together onto one line in order to lower costs. The manner in which the terminals of a multipoint line are linked is known as the line topology. In the terminal layout problem, each terminal is assigned a weight (w_i) that represents the average amount of traffic exchanged between it and the central computer, and the data communication lines that are used for this exchange are constrained by a maximum allowable line weight (*Wmax*) to reflect the capabilities of commercially available lines; their cost is a function of the distance that they must span. The optimal solution will be one which minimizes the total cost of the network subject to the constraint that the weights of all terminals on each line may not exceed the data transmission capability of the line, and to the connectivity constraints imposed by the desired topological architecture. In addition, because of reliability concerns, network designers typically impose a limit on the number of terminals that can share a single multidrop line to limit the disruptive effects of a breakdown.

As we shall see, the terminal-layout problem is NP-complete and can be addressed in two ways: as a 'line cost (distance) minimization' problem or a 'line quantity'

minimization problem. The focus of this thesis is on cost (distance) minimization which comes down to finding a minimum spanning tree with extra constraints on the traffic capacities and on the number of terminals per line. The line minimization approach can also be viewed as a bin-packing problem where terminals with different capacities play the role of a set of objects of various sizes that must be fitted into the smallest possible quantity of equally sized bins (lines) having ample enough storage capacities to at least accept the largest sized object among the set.

There are basically two types of solutions for interconnecting terminals to line-concentrating devices (or directly to the central computer): (1) *exact solutions*, which include branch-and-bound methods, are limited to problems of relatively small size, and (2) solutions obtained using *heuristic techniques*. Heuristics have the advantage of requiring less computer running time while offering solutions that sometimes are near-optimal. However, the solutions obtained by heuristics are usually local minima. Simulated annealing (SA) uses a probabilistic modification method that enables the process not to get trapped too soon in a local minimum while searching to minimize its cost function. By using SA, we attempt to further minimize the cost of multidrop lines for the various topologies (tree-bus-loop) by improving upon the results of simple local search heuristics. To accomplish this, SA must occasionally accept increases in the cost function that allow it to 'jump out' of local minima traps. The analogy is to the physical annealing process of metals, a process by which a substance is first melted at a high temperature and then allowed to cool by lowering the temperature slowly, where the cooling schedule dictates the quality of the final product.

1.3 - Outline of the Thesis.

This thesis is divided into seven chapters. In the first we will state the nature of the problem and familiarize the reader with basic concepts of network design, including a description of the physical network components and their functions. In addition, the characteristics of each type of network topology will be depicted. For the tree architecture, the terminal-layout problem will be defined as a *capacitated minimum spanning tree problem* and explained in detail. Moreover, current algorithms used to resolve the capacitated minimum spanning tree problem will be examined and discussed. Extensions of the terminal layout problem to other topologies such as the bus and the loop architectures will follow.

Chapter two will describe the generalized simulated annealing process and some variants and modifications that make it an extremely flexible tool in optimization. In the third chapter, we shall explain how our programs go about generating a neighbour solution for the SA process and will include illustrated details of the various neighbour generating methods. The fourth chapter gives a general description of our algorithms. Chapter five introduces the data structures used when programming the algorithm, and describes how these were manipulated to generate neighbour solutions. Chapter six describes the SA algorithms in detail using pseudo-code.

Finally, the computational results obtained by running the programs will be presented and analyzed, followed by some concluding remarks. The appendix includes the details of our programs describing our SA applications implemented in Delphi™ for the Windows 3.1™ environment. More specifically, it will illustrate the input and output

capabilities of the programs. This description includes an account of the functions, graphical capabilities, and limitations of the applications produced.

CHAPTER 2 : DESIGN OF CENTRALIZED NETWORKS.

2.1 - An Overview of the Problems Arising in the Design of Centralized Networks.

A *network* is defined as being "two or more computers connected via a communications medium, together with all communications, hardware, and software components. Alternatively, a host processor, together with its attached terminals, workstations, and communications equipment such as transmission media, modems, and so on." [Stamper, 1991, p.600]. Examined topologically, networks can be considered as a three level hierarchical structure. At the first and lowest level we find terminals dispersed geographically at known locations. At the second level in the hierarchy are line-concentrating devices with a limited number of possible sites for locating them. These sites may or may not coincide with the terminal sites. These intermediate line-concentrating devices are generically referred to as 'concentrators' and are connected to the central computer via high speed lines such as fiber optic cables. When designers can identify a cluster containing terminals that are in close proximity to one another but relatively far away from the central computer, the use of a line-concentrating device may be justified. These devices "consolidate low-speed lines into higher-speed lines, taking advantage of the economies of scale of cost versus capacity" [Kershenbaum, 1993, p.179]. When such a device can be placed in the vicinity of the cluster, terminals can be connected to this device rather than spanning the distances needed to link each and every terminal to the central computer. Economies of scale dictate that in some instances the savings such devices generate outweigh the additional costs (acquisition, installation, and connection)

incurred. Finally, at the highest level of the hierarchy we find a single element referred to as the central computer or simply the centre.

A network that is designated as *centralized* is a "network where all communication is to and from a single (central) site. In such networks, the other sites usually have relatively simple equipment not capable of making (message) routing decisions. Also, with all traffic going to a single site, there is little motivation (other than reliability) for including lines between other sites. This leads to a tree topology where there is only one possible path to the center (and hence, between any pair of nodes)" [Kershenbaum, 1993, p.179]. Figure 2.1 illustrates the hierarchical structure of a centralized network.

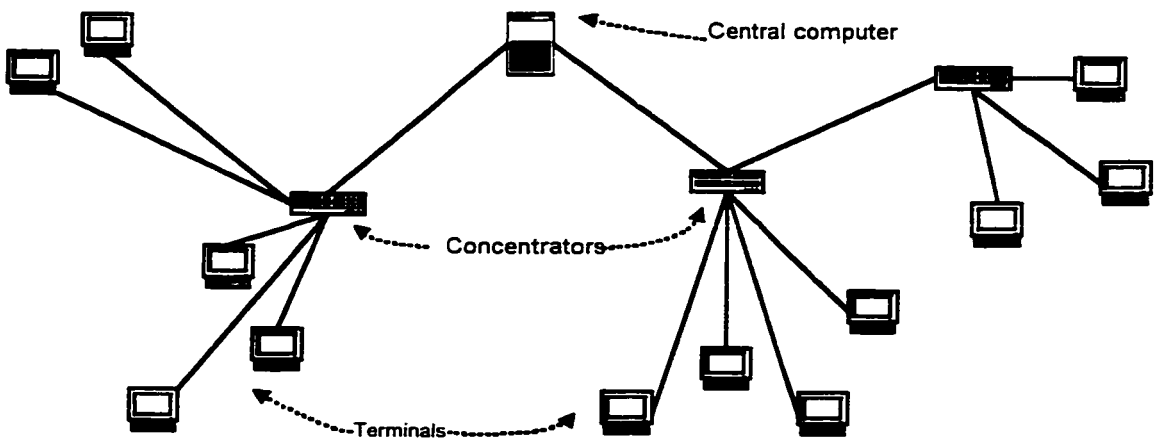


Figure 2.1 A centralized computer network.

The design of centralized networks is made up of three subproblems. These are:

1 - Concentrator Location - Deciding where and how many concentrators to use, if any.

2 - Terminal Assignment - Determining what terminals will be assigned to which concentrator where concentrators are limited in the number of terminals and amount of traffic that they can accommodate.

3 - Terminal Layout - Selecting a cost effective way to connect the terminals to their assigned concentrator with possible configurations being the star, the tree, the bus, and the loop. All of which will be discussed below.

Before addressing each of these problems in detail, we will briefly discuss some of the network components in order to give the reader a rudimentary understanding of centralized network design.

2.2 - Media and Hardware Components.

2.2.1 - Conductive Media.

Today, *wires* constitute the most commonly used transmission medium.. They are available at relatively low prices but are vulnerable to signal distortion or error and have limited transmission rates for long-distance links. Wires can be classified either as *private* (deployed by the user) or *public lines* (provided by the telephone company). Ordinarily, public lines are used where distances are large or geographical factors prevent the installation of private lines. Stamper [1991] notes that if distances are small, the use of private lines allow speeds of up to 80 000 bps. When distances are large, data communications can employ either *switched* or *leased* lines. In addition to using the same equipment as a regular voice telephone call, the use of *switched connections* require *modems* (an acronym for modulator/demodulator). These devices change "a computer signal from digital to analog format for transmission along a medium such as telephone lines, and another modem converts the signal back to digital format at the receiving end" [p.5]. Typical transmission speeds are 2400, 4800, 9600, 14 400, 28 800 bps. Although telephone lines were originally implemented to send analog signals, technological improvements in recent years have enabled telephone companies to devise digital data technology that allows speeds of up to 56 000 bps with switched connections. By simply dialing the telephone number of the other device's line, one of the two devices to be connected sets up the circuit. Commonly, "switched lines are used when the amount of transmitted data is small or when many locations are contacted for short periods" of time. [Stamper, 1991, p.46].

Alternatively, *leased lines* are put to use when "the connection time between locations is long enough to cover the cost of leasing or when speeds higher than those available with switched lines must be attained" [Stamper, 1991, p. 46]. With leased lines, also known as *dedicated lines*, the connection is sustained throughout the day. The cost of leased lines not only depends on the distance the line must cover and its transmission speed, but also on its susceptibility to error. By *conditioning* the leased line, the telephone company can lower error rates and increase transmission speeds. Typically, conditioned leased lines operate at speeds of 64 000 bps, but speeds superior to 2 billion bps are available through today's digital data transmission technology.

Coaxial cable is used in local area networks (LANs) that normally span short distances such as office buildings situated at close proximity. Data transmission over coaxial cables involves using either frequency separation that enables several channels to be transmitted over a single cable at various transmission speeds or by fluctuating the voltage along the channel, which does not allow multiple channels on a single cable but has the advantage of being less expensive than the previous option. Theoretically, bit rates of more than 400 Mbps are feasible although current transmission rates range around 100 Mbps.

A relatively new communication medium is *Fiber optic cable* used by telephone companies instead of long-distance wires and for certain local data communication network implementations. Made up of glass or plastic fibers woven together, the technology consists of directing light pulses through the cable from source to destination allowing speeds over 2 billion bps (2Gbps). Although fiber optics is costly for short distances compared to wire, it becomes cost effective as distances or data transmission

requirements increase. Other types of data transmission mediums also include satellite and microwave technology; the interested reader is referred to Stamper [1991] for more information on conductive and radiated media including their benefits, limitations, and current commercial implementations.

2.2.2 - Terminals.

A terminal is the user's access to the data communication network. It is...

> "an input and/or output device that may be connected to a local or remote computer, called a *host computer.* The terminal is at certain times dependent on the host for either computation or data access or both.
> The phrase 'may be connected' allows for switched connections and devices that have some degree of processing power and are connected to a host on a periodic basis..."[p.128 Stamper, 1991, p. 128] .

Terminals can be loosely classified into a number of categories according to their capabilities. Although overlaps exist, terminals can be usually labeled as dumb, smart, or intelligent. A "*dumb terminal* passively serves for input and/or output and does no additional processing. Since dumb terminals usually have no memory to store entered data, each entered character must immediately be transmitted to the host, unsolicited, and the host must be always ready to accept data from the terminal" [Stamper, 1991, p. 133-134]. *Smart terminals* on the contrary, have memory. They can receive and store data transmitted from the host computer. Entered data can be saved in the smart terminal's memory until the entire record is ready for transmission. Smart terminals are given a name

that both they and the host recognize. This name is commonly referred to as an *address*. "The host can transmit data addressed to that terminal and the terminal will recognize that the data is intended for it and store the data in memory" [Stamper, 1991, p. 135-136]. Addressing further enables *Host control*, where the host can dictate when and what terminal is allowed to send or receive data. Addressing and host control allow smart terminals to share a common medium (the same line). Lines containing several smart terminals are called multipoint lines. They allow cost reductions in ways that will be discussed further on. Stamper [1991] describes an *intelligent terminal* as having the same capabilities as a smart terminal "but in addition can participate in the data-processing requirements of the system." [p. 139] They usually have more memory than smart terminals since part of this is necessary to run programs and store program data. Secondary storage devices such as hard drives are typically fitted into these machines for *downloading* and *uploading* from, and to, the host computer or to store and run stand alone applications.

2.2.3 - Point-to-Point Versus Multipoint Connections.

In a *point-to-point connection*, a terminal is placed at the end of each communication link from the host. Figure 2.2 below shows a group of terminals where each one is directly connected to the host computer.

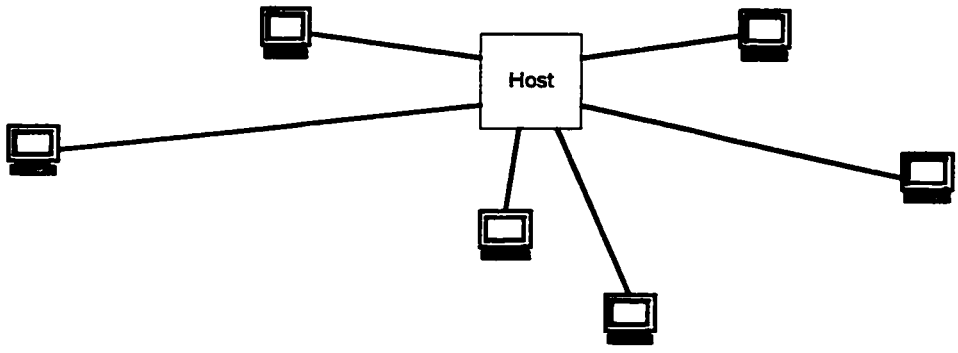


Figure 2.2 Point-to-point connections.

Contention is one simple way of managing the data flow between these terminals. The process consists of a station asking the other party for control of the connection when it is ready to transmit. If the other station is ready to receive, control is simply granted to the requesting device. "Upon completion of the transfer, control is relinquished and the link goes into an idle state, awaiting the next bid for control. A collision may occur when both stations simultaneously bid for the line"[Stamper, 1991, p. 151]. To alleviate this problem, either one station is designated as having priority over the other or each station is asked to wait for a certain time period before reattempting a bid for control. To avoid additional collisions, time-out intervals for each party are assigned different values. *Pure*

contention on the contrary does not require line bids, but allows devices to transmit whenever they are ready. If the message is received correctly, the receiving station returns a positive acknowledgment message. If the sender does not receive this acknowledgment message, or, if a certain period of time has elapsed without any communications at all, the sender assumes that the message was not correctly received and begins retransmission.

In *multipoint connections* also referred to as *multidrop connections*, several terminals may share the same line as depicted in figure 2.3 below. Multidrop lines are made up of several links either placed sequentially, hierarchically, or forming a circuit. The amount of terminals allowed on a multidrop line depends not only on the transmission capacity of commercially available lines and the amount of traffic (messages or bytes per second) of the terminals themselves, but also on the reliability concerns of the designers that may limit the quantity of terminals disposed on each line. When the number of terminals on a line increases, not only does each terminal's average access to the line decrease but in case of a link failure, the potential number of 'downed' terminals in the system rises.

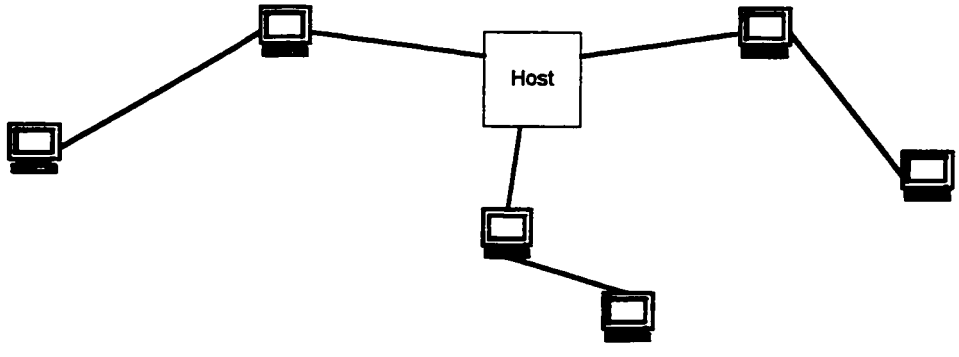


Figure 2.3 Multidrop lines where terminals are configured sequentially (bus topology).

Although contention-like line disciplines also work for multipoint lines, when the number of terminals on a line increases, so too does the number of possible collisions on each channel. In cases where the collision rate is high, the average rate of data transfer diminishes due to the time needed to resolve these collisions. For this reason, different methods were conceived to manage the data flow between devices for multidrop connections. The most common is known as *polling*. One station, usually the host computer (but also possibly the concentrator), is designated as *supervisor* or *primary station*. All other stations are referred to as *secondary stations*. The supervisor assumes total control for managing data flow over the communication link. By asking each secondary station whether it is ready to transmit data, the supervisor can decide when, and to what secondary terminal, the permission to proceed is granted. By referring to its list of terminal addresses, the supervisor sends a short poll message to the chosen terminal. If the latter has nothing to send, it transmits back a negative acknowledgment. Terminals grouped into multipoint lines must either be 'smart' or 'intelligent' since they require some memory in order to store the data they want to send upon awaiting permission to transmit. As we will see below, a concentrator can also be a supervisor managing several multipoint lines.

Selection occurs when the supervisor needs to send data to one or more secondary stations. Similar to polling, it begins by selecting a station by finding its address, and it inquires whether the station is ready to receive data or not. Again, the secondary station responds either positively when it is ready to receive or negatively when, for instance, its memory buffer is full and therefore unable to receive.

2.2.4 - Concentrators and Concentration Techniques.

Although polling and selection allow terminals to share a common communication channel, they however require the use of smart terminals for the protocols necessitate terminals that are addressable and have memory. Furthermore, "polling protocols designed to support high data (transfer) rates impose severe limitations on the number of terminals that can be put on a multidrop line and their distance to the centre. " [Gavish, 1991, p.18]. Alternative line sharing techniques, such as, *multiplexing* and *concentration*, have been developed to respond to these limitations. These techniques divide a communication link into segments, each of which can carry information coming from a separate source. Each segment is separated from other segments by either *time-division multiplexing* (where each data source is allocated a time slot) or *frequency-division multiplexing* (where data sources are transmitted over different frequencies).

One multiplexer at the 'remote site' allows multiple signals to be transmitted over a single link by merging all incoming terminal lines into one line. The combined data is then transmitted over this single link to the host. At the host end a second multiplexer separates the data and distributes it among the outgoing terminal lines. Multiplexers require that the number of lines going into the host side be equal to the number going out to the terminals at the remote side. They make line sharing transparent to the users; in essence they create the illusion and feel of a point-to-point connection.

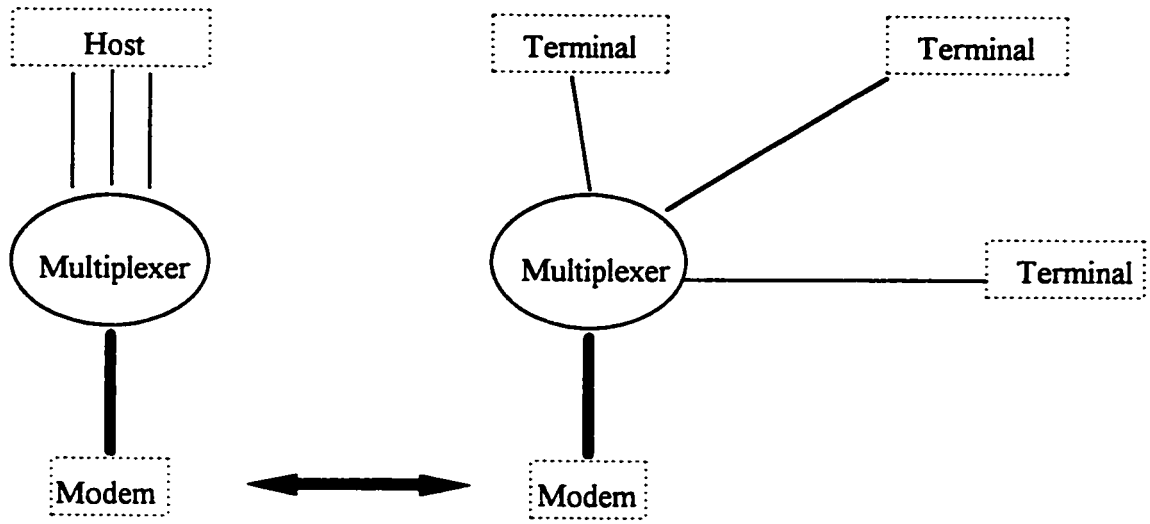


Figure 2.4 General multiplexer configuration.

"A *Concentrator* is also a line sharing device. Its primary function is the same as that of a multiplexer : to allow multiple devices to share a communication circuit. Because a concentrator is a computer, however, it can participate more actively than a multiplexer in any application" [Stamper, 1991, p. 174]. Although in recent years additional functions have been added to multiplexers that have narrowed the differences between multiplexers and concentrators, Stamper describes their main principle differences as follows:

> " **1**. Concentrators are used one at the time; multiplexers are used in pairs.
>
> **2**. A concentrator may have multiple incoming and outgoing lines, with a different number of incoming lines than outgoing lines; a multiplexer takes a certain number of incoming lines into one line and converts back to the same number of outgoing lines.

3. A concentrators is a computer and may have auxiliary storage for use in support of an application.

4. A concentrator may perform some data-processing functions such as device polling and data validation." [p. 174]

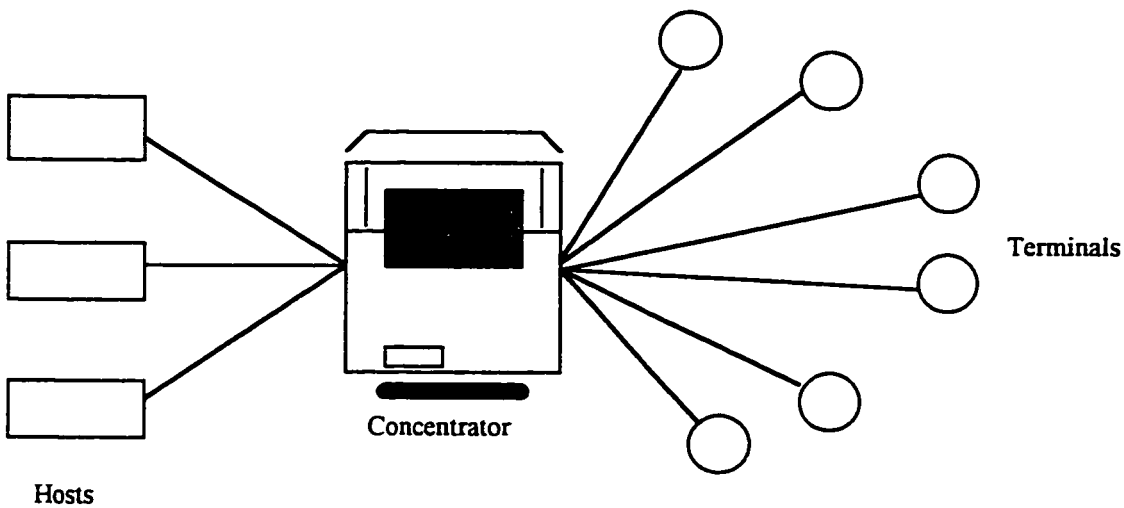


Figure 2.5 A Concentrator Configuration.

Further if the concentrator is equipped with disk drives it can allow intelligent terminals to continue some of their functions even if the link between the concentrator and the host computer is down. Disk storage on a concentrator also provides *store and forward* capabilities; if the communication lines become too busy, the concentrator can store the transmitted data and forward it (retransmit) at a later time. With store-and-forward capabilities, the message one wishes to send is divided up into slices called '*packets*' that contain their destination address. A '*routing table*' is consulted to find the possible itineraries each packet can take to reach the destination independently from one another.

Once all packets arrive at destination they are reassembled by a '*communication protocol*' that insures that the sequence of packets is correctly received at the destination.

2.2.5 - Incentives for Using Multipoint Lines.

In their infancy, computer networks typically served at most a few hundreds of users who accessed a single computer centre that was located at close proximity. Because of these limitations, telecommunications services could effectively be provided by connecting each terminal in a point-to-point fashion via a dedicated low capacity line (*star topology*).

However, as networks evolved and expanded, distances and the number of terminals that they supported grew increasingly larger, and the star configuration's cost became prohibitively high. Besides such economical factors, other considerations include the fact that human beings have significantly slower response times when compared to computers, typically resulting in a very low utilization level of dedicated links. By grouping terminals into a multipoint configuration, it is possible to take advantage of the low line utilization that normally occurs in the star topology. Furthermore, when multidrop lines can be supported by a good polling protocol, their creation allow considerable cost reductions without the users ever perceiving any degradation in service quality.

These cost reductions stem from economies of scale. Since only one line is needed, if modems are used, fewer of them are required. Note that for each point-to-point connection one pair of modems is necessary, while in multipoint configurations one

modem per terminal plus one for each line at the host would suffice. And if the terminals are sufficiently close in proximity, their individual modems can be replaced by a *cluster controller modem* that supports several terminals at once.

2.2.6 - Possible Drawbacks of Using Multipoint Lines.

Multipoint configurations also have their disadvantages. The terminals required on such lines must be 'intelligent' to some degree, which makes them more expensive than the terminals used in point-to-point configurations. However this increase in cost is usually offset by the savings in transmission medium and modems. Another disadvantage is an increase in the users' waiting time when too many terminals share the same medium.

2.2.7 - Design Issues for Multidrop Lines.

Capacity issues come into play since the transmission medium (typically a line) is limited by the amount of data it can carry (referred to as line speed, and commercially available in discrete sizes such as 2400 bps, 4800 bps, 9600 bps, etc.). Consider the following example:[see Kershenbaum, 1993].

> Suppose there are 8 terminals with weights of 100 characters per second and 2 terminals with weights of 4000 char/sec. We consider using 9600 bps (or 1200 char/sec since 8 bytes = 1 character) lines and 56 Kbps lines (or 7000 char/sec). Suppose that, based on delay and stability issues, we are willing to tolerate at most a 70 percent utilization on the lines, therefore

the 9600 bps lines can accept up to 1200*0.70=840 char/sec, while the 56 Kbps can only accept up to 7000*0.70=4900 char/sec. Given the previous information, we can see that the 2 terminals with weights of 4000 char/sec would be individually placed on a pair of point-to-point 56 Kbps lines, all the remaining terminals with weights of 100 char/sec could then 'somehow' occupy a single multipoint 9600 bps line.

Reliability issues can not only be addressed by limiting the number of terminals any line can carry, but reliability is also affected by the choice of the topology of the multipoint lines. Multipoint line topologies are sometimes referred to as *network architecture* or *network topology*. They come in several varieties defined by how the line's terminals are interconnected. The most common topologies are the *tree* (also referred to as *hierarchical*), the *bus*, and the *loop* (also known as the *ring*). Others exist such as the *interconnected network* where every component is connected to every other component with which it must communicate, and *combination network* where various types of topologies are sometimes integrated into one network; these will not be covered in this thesis.

2.3 - Topologies for Multidrop Lines.

2.3.1 - The Hierarchical (Tree) Topology.

The tree structure contains one root node, typically the central computer or a concentrator. At every following level we find the terminals. Each terminal node may have several cascaded terminals attached to it. The tree network is illustrated in the following figure.

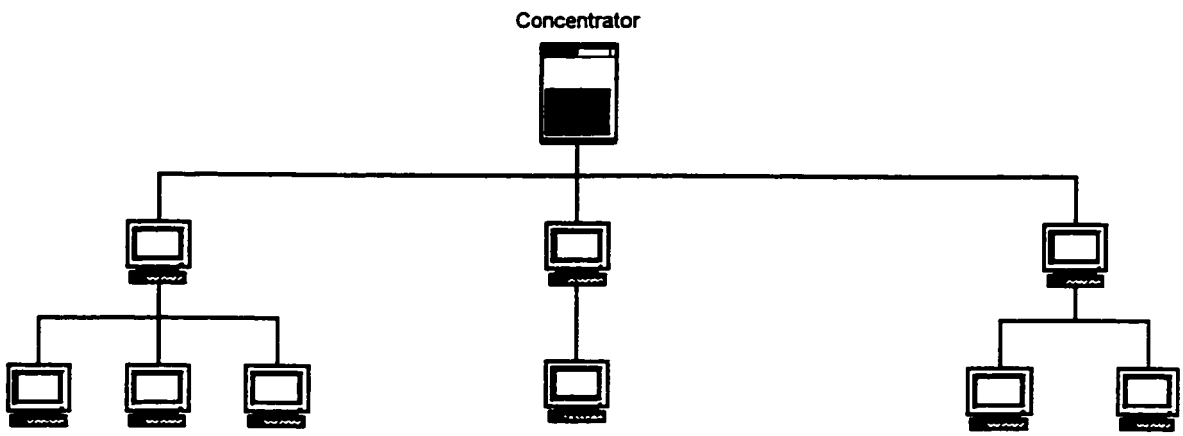


Figure 2.6 A multidrop line configured as a tree with three subtrees.

Each node may have a certain number of child nodes ranging from 0 to a predetermined maximum. Each node in the subtree except for the root node must have a parent node. Trees are typically inexpensive structures, but have the obvious disadvantage that the failure of a single link may disconnect a considerable part of the network.

2.3.2 - The Loop (Ring) Topology.

A loop structure is depicted in figure 2.7 below. A line is a loop when its terminals possess a parent linking it into the network, and only one child. Furthermore, the first and last terminals in the structure must be linked to the central site thus creating a circuit.

The reliability of loops is better than that of trees. Although the cost is usually higher, the reliability issue often warrants the extra expenditure needed in wiring. "All traffic ordinarily travels in one direction around the loop, say, clockwise. If, however, a link breaks, the terminals may have the capability of recognizing this and of temporarily using the remaining portion of the loop in the other direction. This can require a manual switchover and some rearrangement (usually at the software level) at the central site" [Kershenbaum, 1993, p.196]. Loop structures are usually used in local area network implementations that must be highly connected for the sake of reliability.

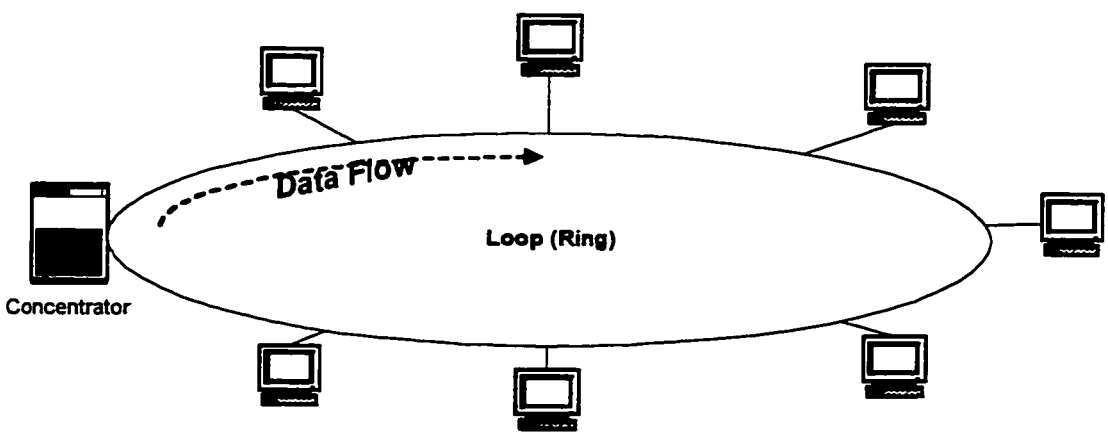


Figure 2.7 A Multidrop Line Using a Loop Configuration.

2.3.3 - The Bus Topology.

Although the bus architecture resembles a loop in which one of the ends is not connected to the concentrator, it is actually a tree topology where the number of child terminals is constrained to 1.

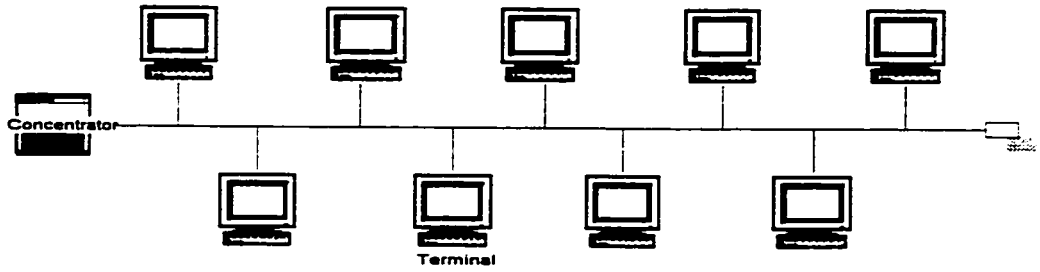


Figure 2.8 A multidrop Line Using a Bus Configuration.

Figure 2.9 illustrates how the various hardware and media components introduced previously may be combined to create a network containing all three topologies discussed above.

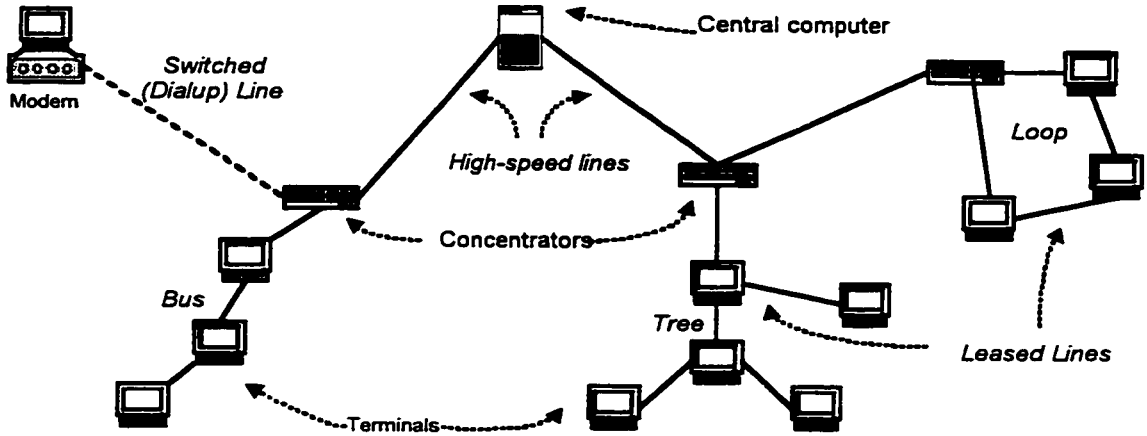


Figure 2.9 A Centralized Network Containing All Three Types of Multipoint Topologies.

2.4 - Centralized Network Design Problems.

2.4.1 - The Concentrator Location Problem.

The concentrator location problem is a version of the plant location problem in Operations Research. Given are network points having demands for a given commodity and several sites for possible plant locations. There exists a fixed cost of opening a plant with a certain capacity at a given site. The shipping costs from plant locations to demand points for each unit of the commodity are known. The problem is to determine at which sites to open plants so as to minimize the total cost consisting of building and commodity shipment costs. Similarly, in the concentrator-location problem there is a limited number of terminals that can be accommodated by each concentrator. Boorstyn and Frank [1976] identify this limit as a function of "the limitations in buffer space, input ports, addressing structure, to the finite capacity of the line from concentrator to central site which restricts the amount of traffic from all the assigned terminals that can be handled by each concentrator, and to the share of resources used by the polling scheme" [p.31]. Because of the complicated nature in which these factors combine to estimate the number of terminals a concentrator can handle, research in this area usually assumes that the concentrator's capacity is already known. In the simple model, either one concentrator 'size' is considered where all concentrators are assumed to have the same capacity constraint or even more simply capacities can be utterly overlooked. The basic problem formulation is:

$$\min z = \sum_{ij} c_{ij}\, x_{ij} + \sum_{j} d_j\, y_j$$

subject to:

$$(1.1) \qquad \sum_{j} x_{ij} = 1 \qquad \forall_i$$

$$(1.2) \qquad \sum_{j} x_{ij} \leq N y_j \qquad \forall_j$$

$$(1.3) \qquad x_{ij}, y_j \in \{0,1\} \qquad \forall_i \forall_j$$

where c_{ij} represents the cost of connecting terminal i to the concentrator situated at location j, d_j represents the cost of placing a concentrator at location j, and N is the total number of terminals. Equations (1.1) ensures that each terminal is associated to exactly one concentrator; (1.2) ensures that the concentrators that have associated terminals are included in the solution,

x_{ij} = 1, if and only if terminal T_i is connected to a concentrator placed at site j.

y_j = 1 , if and only if a concentrator is located at site j.

If a data capacity constraint is imposed on the concentrators, then the following additional constraint must be added:

$$(1.4) \qquad \sum_{i} w_i\, x_{ij} \leq W_j\, y_j \qquad \forall_j$$

where w_i is the weight (amount of data) associated with terminal i, and W_j the maximum transmitted data capacity that the concentrator placed at j can accept. A further restriction can be imposed on the number of terminals that a concentrator can handle, this can be expressed by :

$$(1.5) \qquad \sum_i x_{ij} \leq K\, y_j \qquad \forall_j$$

where K expresses the maximum number of terminals a concentrator can handle and is the same for all concentrators, but it can be allowed to vary with each concentrator.

Well-known algorithms used for locating concentrators are the COM (Center of Mass), ADD, and DROP algorithms. We refer you to [Kershenbaum 1993] for further details.

2.4.2 - The Terminal Assignment Problem.

When a specific set J of concentrators have already been picked, the concentrator-location problem is reduced to the terminal assignment problem. As in the previous problem description, cost is generally a function of the distance separating terminal T_i from concentrator C_j but unlike in the concentrator-location problem the cost of the concentrators can be ignored since it is actually a sunk cost. To minimize the cost of connecting the terminals to the concentrators, the problem is expressed in the following way:

$$\min z = \sum_{ij} c_{ij}\, x_{ij}$$

subject to the constraints:

$$(2.1) \qquad \sum_j x_{ij} = 1 \qquad \forall_i$$

i.e., each terminal must be connected to a concentrator,

$$(2.2) \qquad \sum_i w_i \, x_{ij} \leq W_j \qquad \forall_j$$

i.e., each terminal has a capacity w_i associated to it and the sum of these capacities must not surpass the allowable maximum capacity W_j of the concentrator they are assigned to.

$$(2.3) \qquad x_{ij} \in \{0,1\} \quad \forall_i$$

where x_{ij} = 1, if terminal T_i is connected to a concentrator at site j, and
= 0, otherwise.

Furthermore, if each concentrator is limited by the maximum number K of terminals it can manage, the following additional constraint applies as in the concentrator-location problem:

$$(2.4) \qquad \sum_i x_{ij} \leq K \qquad \forall_j$$

Classical algorithms for solving this problem are the Greedy and Exchange algorithms that are discussed in [Kershenbaum, 1993].

2.4.3 - The Terminal Layout Problem (The Design of Multipoint Lines).

The solutions to both problems stated above generate clusters of terminals associated with each concentrator placed at a certain location. The next step in designing the network is to find the manner in which the terminals in these clusters must be connected to their assigned concentrators via some type of line configuration.

Designing multidrop lines gives rise to a well-known combinatorial optimization problem, the capacitated minimum spanning tree (CMST) problem. The problem consists of finding "trees of minimum total length (cost) subject to the constraint that the aggregate weight of the terminals on any multipoint line does not exceed a given weight, W." [Kershenbaum, 1993, p.182]. Although possible configurations previously described also included the loop and bus configurations, the tree configuration is the basis for many of the algorithms that will be discussed in the following pages and we shall therefore commence with it. But before discussing the problem of optimizing the lines, let us familiarize ourselves with some basic concepts of graph theory that are useful for the analysis and topological design of computer networks.

2.4.3.1 - Basic Notions of Graph Theory.

A *graph* is a collection of nodes connected to one another by *edges*. If we designate by V this collection of nodes and by E the collection of edges, we can denote the resulting graph by $G=(V,E)$. In a graph made up of n nodes, the maximum number of edges is given by $m_{MAX} = n(n-1)/2$. A *directed edge* contains an origin and a destination that are not interchangeable, and serve to model unidirectional communication links, while

non-directed edges can represent two-way links between pairs of nodes. A graph is said to be *directed* when all its edges are directed. An edge is said to be *incident* to a node when the node represents either its origin or its destination. The *degree of incidence* of a node refers to the number of edges incident to that node, and the *degree of a graph* corresponds to the node with the smallest degree of incidence in the entire graph.

A *path* between two nodes i and j designates a sequence of edges starting at *i* and terminating at *j* with no repeated nodes. Each edge is given with a number called its 'length'. The *length of a path* simply designates the sum of all the lengths of the edges that constitute it. When the origin and destination of a path are identical, we call this a *cycle* or *circuit*. A graph is *connected* if each pair of nodes is linked by at least one path. In the case where there exist *k* paths between each pair of terminals in the graph, it is said to be *k-connected*. The interested reader is referred to [Goudran and Minoux, 1984].

2.4.3.2 - The Minimum Spanning Tree Problem and Extensions.

A tree is basically a graph with a degree of connectivity equal to 1 and containing no cycles. A *minimum (length) spanning tree (MST)* is a minimum length tree containing all the nodes of a given graph. In the MST problem, we are given a symmetric matrix V x V of positive values d_{ij} representing the Euclidean distance of edge (i,j). The task is to find a shortest spanning tree. Figure 2.10 below, shows an example of a minimum spanning tree containing two subtrees.

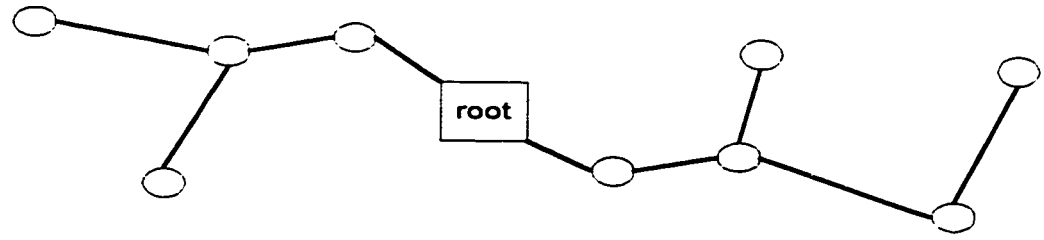


Figure 2.10 A Minimum Spanning Tree

When creating communication networks, minimum spanning trees become extremely meaningful and useful since they are...

> "the basis for many algorithms and design and analysis techniques.(...)They are minimal networks; they provide connectivity without any unnecessary additional links. (...) However, since trees are minimally connected they are also minimally reliable and robust. This is why actual networks are usually more highly connected. Nevertheless, the design of a network often starts with a tree." [Kershenbaum, 1993, p.139].

Algorithms capable of generating MSTs can be used to for centralized network design by simple extension of the MST problem.

2.4.3.3 - Extending the MST to Include Capacities.

When designing multidrop lines, a capacity constraint is needed since the lines that will carry the data to and from the terminals have limited transmission capabilities. Expressing this constraint can be done by considering that terminals have a unit traffic requirement which we can depicted as w_i (weight of terminal i) while a constraint *Wmax* can be placed on the maximum weight any subtree (multipoint line) can carry. This

constraint indicates the maximum data transmission capabilities of the commercially available lines being considered, and the solution to this problem is referred to as a *capacitated minimum spanning tree(CMST)*.

In the CMST, we seek to solve the following problem:

Given :

1. a node set $V=\{v_i \mid i=0, 1, ..., n\}$ representing the terminal locations, where node v_0 is the central site and n is the number of terminals,

2. a symmetric function giving the length (cost) C_{ij} of an arc between any pair of locations,

3. a line capacity constraint ***Wmax***, and a weight (or traffic) w_i associated with each node, where we require that the sum of the weights associated with the nodes on any multidrop line not exceed Wmax. This constraint indirectly limits the number of terminals that can be associated to a multidrop line,

4. no possibility of having a cycle, and

5. that each terminal must have a link connecting it into the configuration.

Minimize:

$$\sum_{i,j} c_{ij}\, x_{ij}$$

where x_{ij} is a 0–1 variable associated with each node pair (i, j) (where $i < j$) and $x_{ij} = 1$ if edge (i,j) is selected in the solution.

It is possible to further constrain the problem described above by incorporating both a limit on line capacity and on the number of terminals any line can handle.

2.4.3.4 -The Complexity of the CMST Problem and Constrained Optimization Techniques.

"There is a general agreement among computer scientists that an algorithm is a practically useful solution to a computational problem only if its complexity grows *polynomially* with respect to the size of the input. For example, algorithms of complexity $O(n)$ or $O(n^3)$ are acceptable in this school of thought. (...) Naturally, algorithms for which the asymptotic complexity is not a polynomial itself but is *bounded* by a polynomial, also qualify. Examples are $n^{2.5}$ and $n \log n$." [Papadimitriou and Steiglitz, 1982, p.164]. On the contrary, an *NP-complete* problem is a computational problem where "the running time of algorithms currently known to guarantee an optimal solution is an exponential function of the size of the problem" [Eglese, 1990, p.271). Papadimitriou [1978] has shown that the CMST falls into this category of problems when $2 < Q < n / 2$, where Q represents the number of terminals a line can carry. The interested reader is referred to Garey and Johnson [1979] for a complete explanation of NP-completeness.

The techniques currently used for solving the CMST problem offer solutions that fall into one of two classes:

1) ***Exact solutions*** using branch & bound and integer programming techniques [Chandy and Lo, 1973], [Gavish, 1982], [Gavish, 1983], [Gavish, 1985], [Kershenbaum and Boorstyn, 1983], and more recently, [Malik and Yu, 1993] and [Gouveia, 1995]

2) ***Near-optimal solutions*** using heuristical methods.

In this thesis, we shall completely focus on the latter due to the limitation on problem size imposed by techniques yielding exact solutions. Gavish [1991] indicates that

the techniques used to obtain exact solutions are limited to problems involving up to 30 terminals. Similarly, Kershenbaum [1974] states that "it is generally felt that these techniques are too slow to be of practical use in the solution of the CMST problem for networks with more than 50 nodes. Even for networks of such moderate size, when the constraints are neither very loose nor very tight, these techniques tend to use too much computer running time to make them practical" [p. 300].

Heuristic procedures offer a respite from this increasing complexity relative to problem size. In practice, a terminal layout problem consists of several hundred nodes that need to be connected. Heuristics have the advantage of providing a sometimes decent solution to large problems with acceptable computational effort. "Running times are on the order of seconds for networks containing several hundred nodes and the quality of the solution is generally within 5% of the optimum obtained using branch and bound techniques" [Kershenbaum, 1974, p.300]. In the following pages, some well known heuristics developed for the CMST will be introduced and their characteristics explained and compared.

2.5 - Heuristics for the CMST problem.

Various heuristics have been proposed for solving the CMST problem. It is possible to divide these algorithms into three categories loosely based on the way they create the subtrees of the CMST. Note that when the a capacity constraint is not present, these algorithms all converge to a minimum spanning tree.

The three classifications are :

1) First Order Greedy Algorithms (FOGA),
2) Second Order Greedy Algorithms (SOGA), and
3) Clustering Algorithms.

The term '*greedy*' refers to algorithms that use local search. They are also known as down-hill search and descending algorithms in optimisation literature. They start with an initial solution i, changes are then made to the current solution while respecting all constraints to generate a new solution j called a *neighbour solution*. The change in cost $\Delta C_{i\,j}$, is then evaluated. If cost is reduced, the current solution is replaced by the neighbour solution, otherwise the current solution is retained. The process is repeated until no additional cost reductions can be generated. The algorithm then terminates at a local minimum. Basically, the algorithms that fall under this designation differ from one another by the sequential order in which links are created. The general format of the greedy algorithm is given in pseudo-PASCAL as follows:

```
Procedure GenericGreedy; (Based on minimization)
Begin
        Initialize (i);
        Repeat
                Generate neighbour solution (i → j);
                Calculate ΔC_ij = C(i) – C(j)
                If ΔC_ij ≥ 0 then i:=j;
        until ΔC_ij ≤ 0 for all j in the neighbourhood of i;
End;
```

In the following pages we will introduce various heuristics developed to resolve the CMST problem. The Esau-Williams algorithm will be given special attention since it provides the initial solution used by the SA programs described in the following chapter.

2.5.1 - First Order Greedy Algorithms.

2.5.1.1 - The Esau-Williams algorithm.

Esau and Williams [1966] used the notion of a 'trade-off value' given by the formula $t_{ij} = d_{ij} - d_{0i}$, where d_{0i} denotes the distance between the terminal i and the central site '0', while d_{ij} represents the distance from terminal i to terminal j.

Initially, each component i is made up of terminal i and the central site, in other words, terminals are considered to be directly connected to the central site in a star configuration. For each pair of components (i , j), the trade-off is calculated

representing the change in cost of removing the central link connecting component i to '0' and forming the link ($i \rightarrow j$). At each iteration, the algorithm finds the best trade-off t_{ij} and merges the component currently containing terminal i with the component currently containing terminal j. Before joining these components into one, the combined traffic is checked to verify that it does not exceed the amount that one line can handle. If the link is accepted and components i and j are merged, then $d_{0i} = d_{0j}$ if $d_{0i} > d_{0j}$ or $d_{0j} = d_{0i}$ if $d_{0i} < d_{0j}$. The algorithm stops when no additional savings can be obtained by combining components together. The following example illustrates the iterations of the algorithms:

> Consider that a network must be created with terminals numbered from 1 to 4 and a concentrator denoted as terminal '0'. The length separating the nodes is given, and represents the cost of linking a pair of nodes. Suppose that Wmax is 2, and that all terminal weights are 1.

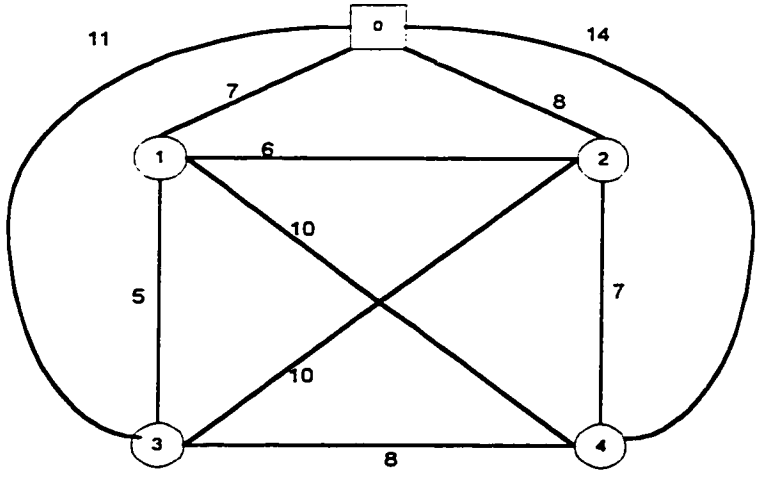


figure 2.11 A CMST problem.

The Esau-Williams algorithm begins by computing the trade-offs.

$$t_{12} = 6 - 7 = -1$$
$$t_{13} = 5 - 7 = -2$$
$$t_{14} = 10 - 7 = 3$$
$$t_{21} = 6 - 6 = 0$$
$$t_{23} = 10 - 6 = 4$$
$$t_{24} = 7 - 6 = 1$$
$$t_{31} = 5 - 11 = -6$$
$$t_{32} = 10 - 11 = -1$$
$$t_{34} = 8 - 11 = -2$$
$$t_{41} = 10 - 14 = -4$$
$$t_{42} = 7 - 14 = -7$$
$$t_{43} = 8 - 14 = -6$$

The link (4,2) is chosen and both components are merged. Since the cost from terminal 4 to the centre has changed from 14 to 8, we must recompute the trade-offs for terminal 4.

$$t_{41} = 10 - 7 = 2$$
$$t_{42} = t_{24} = 0 \text{ (since they are part of the same component)}$$
$$t_{43} = 8 - 7 = 1$$

The link (3,1) is chosen and both components are merged. Since the cost from terminal 3 to the centre has changed from 11 to 7, we must recompute the trade-offs for terminal 3.

$$t_{31} = t_{13} = 0 \text{ (since they are part of the same component)}$$
$$t_{32} = 7 - 7 = 0$$
$$t_{34} = 8 - 7 = 1$$

The link (1,2) is chosen but is rejected since it would create a component of weight 4. The algorithm ends by creating links (1,0) and (2,0), representing the smallest cost from each component to the centre. Figure 2.12 represents the Esau-Williams solution.

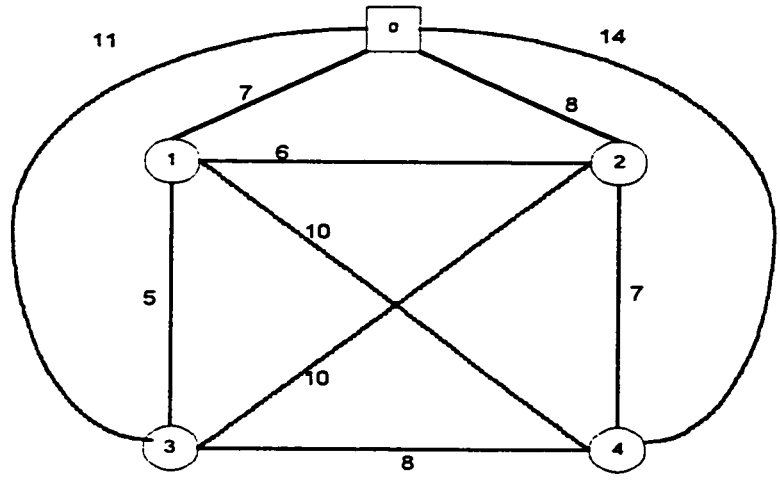


Figure 2.12 The Esau-Williams Solution with Wmax = 2 and all $w_i = 1$.

Note that without a capacity constraint, link (1,2) would be accepted, the result would be a minimum spanning tree.

2.5.1.2 - The Modified Kruskal Algorithm.

The Kruskal algorithm [Kruskal, 1956] begins by sorting all edges (shortest first). While traversing the sorted list, it includes all edges that do not form cycles with edges previously selected. Once all terminals have been linked, the algorithm stops. By modifying the Kruskal algorithm [Boorstyn and Frank,1977], it is possible to apply it to the capacitated minimum spanning tree problem by not including edges that would violate the capacity constraint imposed on the subtrees. Any terminal that violates the constraint is removed from consideration and connected "to the centre by a least cost line. The (modified Kruskal) algorithm has a worst-case time complexity of $O(n^2 \log n)$ for a fully

connected graph of n locations." [Gavish, 1991, p.42]. The algorithm offers worse results than the Esau-Williams algorithm for comparable CPU time.

2.5.1.3 - The Modified Prim Algorithm.

Prim's algorithm [Prim, 1957] viz. [Gavish, 1991] begins by assuming that the central node is the only node in the tree while all other nodes are not. It then finds the *'out-of-tree'* node that is the nearest to the tree and includes it into the tree. The algorithm then updates the distance to the tree for all out-of-tree nodes. The algorithm stops when all nodes have been included into the tree. In the capacitated version of the algorithm, if a subtree reaches the capacity constraint its nodes are eliminated from future consideration. The computational complexity is $O(n^2)$, while offering worse solutions than those provided by the Esau-Williams algorithm for comparable CPU times.

2.5.1.4 - Vogel's Approximation Method (VAM). [Reinfield and Vogel, 1958] viz. [Kershenbaum, 1993] and [Gavish, 1991]

Similarly to the Esau-Williams algorithm, a trade-off function determines values for $\boldsymbol{t}_{ij}$, where $\boldsymbol{t}_{ij} = \boldsymbol{b}_i - \boldsymbol{a}_i - \boldsymbol{d}_{ij}$. The value $\boldsymbol{a}_i$ represents the cost (distance) of connecting terminal i to its nearest neighbour that doesn't violate the capacity restriction, and $\boldsymbol{b}_i$, the cost of connecting to its second nearest neighbour, again without violating the capacity constraint. If terminal i is not immediately connected to its nearest neighbour, there is the risk that the subtree containing the nearest neighbour fills up; and then terminal i would have to connect elsewhere. "By giving preference to (*terminals*) that would

suffer the most by not connecting them to their nearest neighbours, they are considered sooner, making it less likely that they would lose their first choice." [Kershenbaum, 1993, p.193). The quality of the algorithm's results, although slightly worse, are comparable to those offered by Esau-Williams. The CPU time is of the same order as Esau-Williams, but can be considerably greater depending on how the algorithm is implemented as the research by Kershenbaum et al. [1980] revealed. The computational complexity of the algorithm is $O(n^2 \log n)$.

2.5.1.5 - A Unified Algorithm. [Kershenbaum and Chou, 1974].

The algorithm uses the notion of *weight*, that we shall denote as uw_i (not to confuse the reader with the term w_i, denoting the capacity of a terminal as introduced previously in the context of the capacitated minimum spanning tree). A trade-off function for the algorithm is given by $t_{ij} = d_{ij} - uw_i$, where d_{ij} represents the cost (distance) of link (i, j). The algorithm's name is derived from it's ability to reduce itself to other algorithms described above. The difference between all previously introduced heuristics is in the way the uw_i's are defined. For example, if uw_i is set to zero for all i, then the algorithm reduces to Kruskal's algorithm. Similarly, if uw_i is set to d_{0i} then the algorithm reduces to the Esau-Williams algorithm. While Vogel's Approximation Method can be obtained by setting uw_i to $-(b_i - a_i)$.

Kershenbaum and Chou propose the following definition of $uw_i = p_1(p_2 d_i + (1-p_2) b_i)$, where d_i is the cost of connecting terminal i to the centre node and b_i is the cost

of connecting terminal i to its feasible neighbour. p_1 and p_2 are constants such that ($p_1 > 0$) and ($0 \leq p_2 \leq 1$). By changing the values of these parameters, many solutions can be generated, and the best one(least cost) selected. With three to ten different parameter settings, the algorithm offers a 1 to 5 percent improvement on the Esau-Williams algorithm but a three to tenfold increase in computer time. However, if one good set of parameters can be chosen from the start, the CPU time is equivalent to that of the Esau-Williams algorithm. Computational complexity is in the order of $O(n^2 \log n)$.

2.5.2 - Second Order Greedy Algorithms.

First developed by Karnaugh [1976] viz. [Gavish, 1991] a second order greedy algorithm starts with an initial solution provided by one of the first order greedy algorithms, and tries to improve upon its solution. Although the author attained a 1% improvement over the Esau-Williams algorithm, the computational effort demanded an increase of 70 times that of the Esau-Williams.

Kershenbaum et al. [1980] improved upon Karnaugh's idea. According to the authors, the optimal solution for the MST often differs from one of the FOGA heuristic solutions for the CMST by a small percentage. This percentage typically represents a small number of links (M) that were left out of the CMST solution due to the constrained nature of the problem. The heuristic solution is then recalculated a number of times by forcing a number of these links into the solution, and then running the FOGA. The number of times the solution is re-evaluated depends on the amount of links that were originally left-out of the FOGA solution but exists in the MST solution, and how many are

forced at each re-evaluation. Usually, improvements of 1.9 % can be found when the number of left-out links is not greater than 2, but this necessitates 2 to 3 times the CPU time required by the Esau-Williams algorithm.

2.5.3 - Clustering Algorithms.

2.5.3.1 - Sharma's Algorithm. [Sharma and El-Bardai, 1970]

The algorithm partitions the nodes into clusters (or sectors) that do not violate the capacity constraint. Each cluster of nodes is then used to form a subtree by running an ordinary MST algorithm such as Kruskal's. Sharma's algorithm creates these clusters by sweeping out clockwise (or counterclockwise) from the centre node, and including all terminals swept into clusters. A new cluster is started only when the previous cluster can no longer accept the next terminal in the sweep without violating the cluster's capacity constraint. Thus, the quality of the solution depend not only on the direction (clockwise vs. counterclockwise) of the sweep but also on where it starts. Sharma suggests that better solutions (*comparable to Esau-Williams*) can be attained by iterating the start of the sweep at each node, but at a substantial increase in time. The time complexity of the algorithm is $O(n^2 / k)$, where k represents the average number of terminals in a cluster. When the algorithm is repeated for each node, its worst case complexity becomes $O(n^3 / k)$. Kershenbaum [1993, p.191] notes that although Sharma's algorithm is easier to implement than the Esau-Williams algorithm requiring only "an MST algorithm and a sectoring algorithm", the Esau-Williams is "significantly faster if it is implemented

carefully". Furthermore, the main drawback of the algorithm is that it can only be applied to cases where terminal locations are given by their co-ordinate values on a plane.

2.5.3.2 - The McGregor and Shen Algorithm.[McGregor and Shen, 1977]

The algorithm merges the nodes into clusters that do not violate the capacity constraint, these clusters of nodes are then replaced by single nodes. The clusters reflect 'natural' groupings of nodes that are approximated by a single node placed at their centre of mass.

Starting with all nodes connected directly to the centre, two nodes closest together are selected. The nodes are merged only if the merger doesn't violate the line constraint. They are then replaced by a single node at their centre of mass. The weight of the new node is the aggregate weight of all the nodes in the cluster it represents. If the two closest nodes cannot be merged without violating the line constraint, the next closest pair is selected. The algorithm ends when no pair of nodes can effectively be merged.

Similarly to Sharma's algorithm, an MST algorithm can then be run to connect the terminals of each cluster onto a multipoint line.

2.6 -The Terminal Layout as a Bin Packing Problem : An Alternative to the MST Approach.

When one is seeking to minimize the number of lines rather than the total cost (where the total cost is a function of the total length of the spanning tree), the terminal layout problem can be considered as a bin packing problem.

The problem can be viewed as a set of n objects of a given size s_i that must be put into a minimum number of boxes of fixed capacity C, where $i \in \{1,2,...,n\}$. For a feasible solution to exist, one must assume that $s_i \leq C$ for all $i \in \{1,2,...,n\}$. Then, the number of boxes needed is clearly at most n. In the terminal layout problem, terminals can be considered as the objects while the boxes may represent the multidrop data transmission lines.

The problem is formulated as follows:

Given

$N = \{1,...,n\}$

$x_j =$ *1 if box j is used, 0 otherwise.*

$y_{ij} =$ *1 if object i is placed in box j, 0 otherwise.*

Find (4.0) $z = \min \sum_{j \in N} x_j$

subject to:

(4.1) $\sum_{i \in N} s_i y_{ij} \leq C x_j \quad \forall_j \in N$

(4.2) $\sum_{j \in N} y_{ij} = 1 \quad \forall i \in N$

$$(4.3) \qquad y_{ij} \in \{0,1\}, x_{ij} \in \{0,1\} \qquad \forall i \in N \ \forall j \in N$$

Equation 4.0 keeps count of the number of boxes used. Equation 4.1 guarantees the capacity constraint of the boxes. Finally, equation 4.2 insures that every object is placed in exactly one box.

2.7 - Extensions to Other Topologies.

2.7.1 - Extending the CMST to the Loop Topology.

Multipoint lines can also be configured as loops (rings). The advantage offered by this topology is its resistance to failure. Since the bus or tree topologies are minimally connected networks, a single link failure may wreak havoc on a large part of the network by disconnecting some of its users until repairs may be completed. Because of the dire consequences of such a possibility, network designers sometimes opt to configure multipoint lines as loops, reasoning that their reliability justifies the increase in cost due to the additional links.

Loops may be created in two ways. The first approach consists of running an algorithm that solves the CMST. Furthermore, if the resulting subtrees are considered only as partitions, a *Travelling Salesperson (TSP)* algorithm may be used for each partition to find the least lengthy tour starting and ending at the root of the tree. In such a situation, the nodes in each partition are equivalent to the cities the salesperson must visit exactly once before returning home.

The second approach consists of viewing the problem as a *Vehicle Routing Problem (VRP)*. It consists of "finding a set of routes for a fleet of vehicles which have to service a number of stops from a central depot. It is assumed that every vehicle has the same capacity and the number of vehicles is unlimited. The vehicles depart and arrive at the depot. The demand quantity at each stop is known in advance and is deterministic. No single demand quantity exceeds vehicle capacity." [Breedam, 1995, p. 480]. It can be added that the total demand quantity assigned to a given vehicle is no more than the capacity of the vehicle; the number of stops of each route is limited and one is seeking to minimize the overall distance that all the vehicles must cover.

Kershenbaum [1993] proposes the use of a generalization of the Esau-Williams algorithm, called the Clarke-Wright algorithm [Clarke-Wright, 1964, viz. Kershenbaum, 1993]. The algorithm starts by considering each terminal on a separate loop. The algorithm then combines loops when the mergers offer a decrease in cost. Similarly to the Esau-Williams algorithm, a trade-off is associated with link (i, j) given by $t_{i\,j} = C_{i\,j} - C_{i\,0} - C_{j\,0}$, where $C_{j\,0}$ and $C_{i\,0}$ denote the geographical distance between terminals j and i from the central site '0' respectively, while $C_{i\,j}$ represents the distance between terminal i and terminal j. Kershenbaum [1993] adds that the solution obtained by the Clarke-Wright algorithm can then be improved by running a TSP algorithm for every resulting loop.

2.7.2 - Extending the CMST to the Bus Topology.

Recall that a centralized network connected under the bus topology is actually a tree where the nodes of each subtree have been restricted to have a maximum of 2 edges. Thus, any algorithm available to solve the CMST can be further constrained to create multipoint lines that follow a bus architecture.

CHAPTER 3 : AN OVERVIEW OF SIMULATED ANNEALING.

3.1 - Origins of Simulated Annealing.

In thermodynamics, "*physical annealing* is a process in which a solid is heated until all particles randomly arrange themselves in the liquid state, followed by a slow cooling.(...) At each temperature, the solid is allowed to reach thermal equilibrium, where energy levels follow the Boltzmann distribution." [Vidal, 1993, p. 47].

It is essential for the temperature to be lowered in small decrements and the system allowed to settle at each temperature level for "if the cooling is too rapid, the material does not have time to reach equilibrium. Instead, various defects become frozen into the structure." [Greene and Supowit, 1986, p. 221].

Simulated annealing was first introduced to model the physical annealing of solids when Metropolis et al. [1953] simulated a small displacement in individual atoms for each iteration of the simulation while monitoring the change in system energy the displacement produced. When the change corresponded to a decrease in energy, the resulting change was accepted, while increases in energy were only accepted with a certain probability. At each temperature level, a sufficiently large number of iterations were realized to attain thermal equilibrium, and the acceptance function guaranteed that the system was governed by the Boltzmann distribution (also known as the Gibbs distribution).

Kirkpatrick et al. [1983], and Cerny [1985] independently proposed that the simulated annealing process could be applied to optimization problems by comparing the energy states of the solid to an objective function to be minimized. In the analogy, "the

different states of the substance correspond to different feasible solutions to the combinatorial optimization problem, and the energy of the system corresponds to the function to be minimized." [Eglese, 1990, p. 273].

The general simulated annealing algorithm can be described in pseudo-PASCAL as follows:

```
Procedure Simulated Annealing;(for minimizing a function)
Begin
    Initialize the parameters(k, T0, ε, α);(set T:= T0, and 0 < α < 1)
    Repeat
        For m := 1 to k do
        Begin
            Generate a neighbour state (state i→state j);
            Calculate ΔCij = C(state i) - C(state j);
            if ΔCij ≤ 0      then make j the current state;
            else
            if exp(-ΔCij / T) > random [0,1[
                             then make j the current state;
        End;
        T:= T * α ;
    Until T ≤ ε (or until some other stopping criteria is reached);
End;
```

In any implementation of the algorithm an *annealing schedule* must be specified. This schedule stipulates the initial temperature (T_0) setting, the reduction rate (α, also known as alpha) in temperature (T), the repetition (k) which denotes the number of changes to be attempted at each temperature level, and finally a stopping criterion (ε, later referred to as 'epsilon') in order to terminate the program.

As we can see from the SA description above, the process involves accepting or rejecting a number of k neighbouring states at each temperature level (T), while dropping the temperature gradually. For the new state j to be a *neighbour* state of i, it must be reachable in exactly one move form state i, and it must be reversible. Furthermore, if Sj denotes a set of neighbour states reachable in exactly one move from i, then any state of Sj must be capable of being reached from any other in some number of moves. Moves to new states are generally selected at random, although we shall see later on how a systematic approach identifying all possible neighbour states before selecting one was implemented in one version of our SA programs for the CMST problem. If the selected new state represents a decrease in cost then the new state becomes the current state, while states generating increases in cost are not necessarily rejected. Because of the formulation of the probabilistic acceptance function, given as $\mathbf{exp}(-\Delta C_{ij} / T) > \mathbf{random}[0,1[$, note that at high temperature levels it is possible to accept new states that produce relatively large increases in cost. As the system cools, the function becomes more likely to accept states generating small increases in cost rather than larger ones. When the temperature approaches zero, the majority of cost increasing moves are rejected.

Its potential to accept occasional increases in the cost function is what differentiate it from simple local search algorithms since it has the "ability to migrate through a sequence of local extrema in search of the global solution and to recognize when the global extremum has been located" [Bohachevsky, Johnson, and Stein, 1986, p. 209]. Contrarily to simple local search (greedy) algorithms, the SA's sporadic increases in the cost function allow it to 'jump out' of local minima traps. Consider the following figure depicting the cost function of an optimization problem, to illustrate the point:

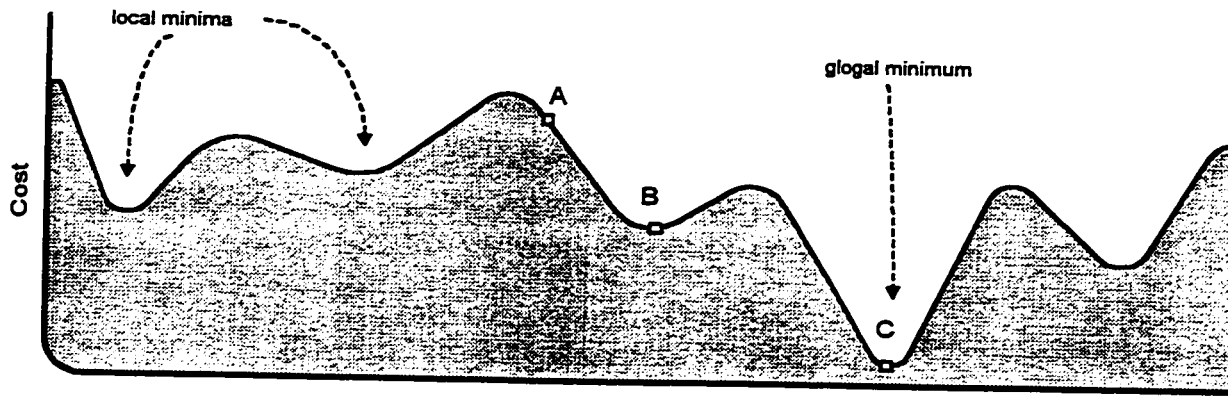


Figure 3.1 An Optimization Problem

Algorithms based on the greedy approach begin by placing themselves at a certain location on the function (point *A* for example), they then start to descend the "hill" until they reach the lowest point (point *B*). The problem that usually occurs is that the values found that minimize the function do so only locally, as depicted by *B*. As point *C* shows, there might exist a better solution but a greedy algorithm will not find it.

3.2 - The Annealing Schedule.

Good annealing schedules are those that combine short execution times for the program and near optimal solution qualities for the results. Often good schedules depend on a the right 'blend' of parameter values in the program, and thus finding such a schedule is an intuitive process of trial_and_error referred to as 'tuning' the algorithm. Below, we shall stress the effects of parameter settings on the behaviour of the SA program. These considerations aid in establishing the right annealing schedule, by guiding the search towards better parameter value settings.

Large initialization values in temperature (T_0) for the program is synonymous to heating up a substance until all its particles are randomly distributed in the liquid form of the substance. Essentially, when T_0 is 'large enough', all neighbouring states have the potential of being accepted. If T_0 is set too high, the program might waste a lot of time attempting to lower the cost of large 'uphill' moves that were accepted early on. On the other hand, if T_0 is set too low, the program may not have the momentum to get out of a local minimum.

Recall that 'α' is the parameter required for reducing the temperature parameter, T. When "a proportional temperature function is used, i.e. $T(t+1) = \alpha\, T(t)$ where α is a constant (...and t denotes time,...) typical values of α used in practice lie between 0.8 and 0.99. Such a function provides smaller decrements in T as zero temperature is approached." [Eglese, 1989, p. 275]. This enables the program to relatively consider more neighbour solutions near the freezing point where lower cost solutions are accepted primarily. Of course, for a given value T_0, the closer the value of α to 1 the larger the

running time of the program. This is because the amount of temperature decrements becomes larger, thus increasing the number of iterations for the program.

It is possible to counter this effect by lowering the number of repetitions (k). Recall that k *affects* the number of neighbouring states that will be considered at each T. The term 'affects' must be stressed, for the value of k is not necessarily equal to the number of neighbour states considered at each T. Only when one neighbour state is considered per repetition is this the case. Instead, at each repetition, it is possible to generate a set of neighbour states before selecting the best among them. This of course, increases the time needed for the program at each iteration but has the advantage of increasing its ability to jump out of local minima traps as the system is cooling. Apart from the ones described above, a variety of schemes have been developed to select the number of repetitions at each T, such as '*repeat until a certain number of acceptances or rejections have occurred* ' at each temperature value. The stopping criteria can be set to stop once the system reaches a certain temperature (ε), which represents the final value of T. Others include stopping the process once a fixed number of temperature decrements have occurred, or after no improvement has occurred over a number of successive temperature decreases.

3.3 - A Variation on the Generalized Simulated Annealing Model.

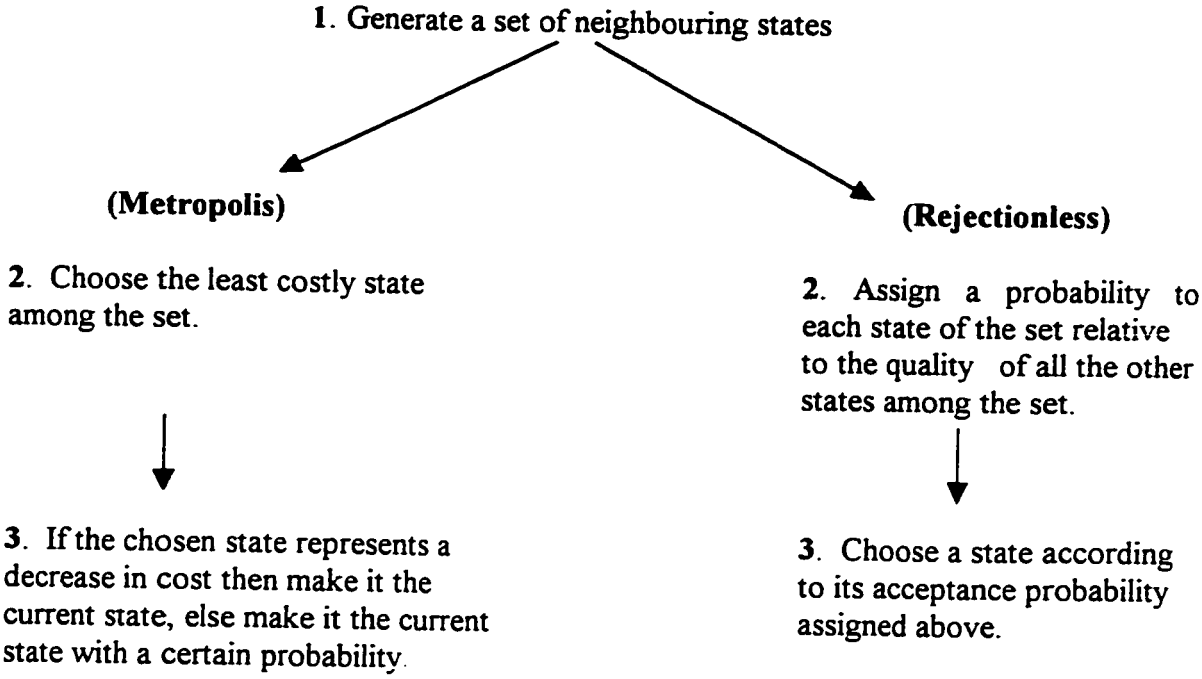
Greene and Supowit [1986] note that "although the Metropolis method is simple, effective and easily programmed, it has one major drawback: at low temperatures, the running time is quite high because many candidates are rejected before each move to a different state." [p. 222]. The authors modified the Metropolis method by assigning probabilities to a set of m neighbouring states chosen randomly. Labeled as the *rejectionless method* by the authors, one of the neighbour states is always accepted to become the current state based on a certain probability value.

The rejectionless method works as follows. For each possible state i, where $1 \leq i \leq m$, a value α_i is calculated that corresponds to the probability of selecting that state under the Metropolis method. Therefore, if state i corresponds to a decrease in cost, α_i is assigned a value of **1.00**. If instead, state i induces an increase in cost, then $\alpha_i = \mathbf{exp}(-\Delta C_i / T)$, where ΔC_i corresponds to change in cost if state i is accepted as the current solution. The probability of accepting neighbour states i under the rejectionless method is then given by:

$$p_i = \frac{\alpha_i}{\sum_{j=1}^{m} \alpha_i}$$

Finally, the cumulative probabilities F_i are calculated before the pseudo-random generator is called. Assuming that F_i = b and F_{i-1} = a, then i is accepted if the pseudo-random generator provides a number in the interval]a, b].

The Metropolis and Rejectionless methods vary from one another in the following way:



It is important to note that, the rejectionless method, although necessitating more computer memory (since it must store the cost effects of all states considered), has the advantage of always changing the current state to one of the neighbouring states i being considered. This is especially beneficial over the Metropolis method when the temperature approaches zero where the Metropolis method spends most of its time rejecting neighbouring states.

3.4 - Various Modifications to SA.

Below, we will demonstrate the flexibility of SA by discussing certain modifications that can be made to the general SA model in order to improve the effectiveness of the basic algorithm.

3.4.1 - *Storing the best solution obtained.*

In implementations of SA, it is possible that the algorithm accepts states that increase the cost function even as the temperature becomes quite low. "It is therefore possible in any single SA run for the final solution to be worse than the best solution found during the run". [Eglese, 1989, p. 276]. In such occurrences, any previous states although offering lower energy configurations are lost. To remedy this, it is plausible to specify that SA continuously remember the lowest state it has found thus far.

3.4.2 - *Starting with a good initial solution.*

It is possible to improve SA by providing it with a good initial state (solution) which allows to shorten the running time of the process. Such initial states may be provided by a heuristic algorithm such as Kruskal's for the CMST problem. In doing so, it should be noted that the SA process must be initialized with a lower initial temperature (T_0), otherwise the process may negate the beneficial effects of such a strategy by perturbing the initial state by a large degree.

3.4.3 - *Combining different stopping criteria.*

A common stopping criteria in SA is to specify a temperature (ε) close to zero. However, as the temperature of the SA process approaches zero, the probability of acceptance of even the slightest cost increasing state becomes very low. With most of its time spent rejecting lower grade solutions, the process continues to search for savings that it may be no longer be able to attain. In other words, if the process finds itself in a local minimum when the current temperature is in the vicinity of zero, the SA may not have the momentum to escape from this trap. This time consuming task can be shortened by combining an alternative stopping criterion to the original one that tells the SA to stop trying to escape after a certain number of sequential temperature iterations without improvement.

For a more in depth look at various modifications to SA, the interested reader is referred to [Eglese, 1990], where the author summarizes the 'speed-up' techniques introduced and tested by various researchers.

3.5 - SA Research into the Terminal Layout Problem.

To our knowledge, only one research study[1] exists where simulated annealing was applied to minimizing the cost of multipoint lines. Starting with all terminals connected via point-to-point connections as an initial solution, the authors reported having achieved "amazingly good" results (close to the optimum) in spite of the short CPU-time per run, typically a few minutes on a 386-16 MHz machine. However, their testing was limited to data sets of at most 63 terminals. Nevertheless, their approach was later used to design a real network in Denmark consisting of 72 terminals and a CPU in the Copenhagen area.

[1] See Anderson, K., Vidal, R., and Iversen, V., "Design of a Teleprocessing Communication Network using Simulated Annealing," in Applied Simulated Annealing, ed. René V.V. Vidal (Berlin: Springer-Verlag, 1993), 201-215.

3.6 - How We Generated Neighbour Solutions.

In this section, we review how our SA algorithms proceed to generate neighbour solutions for the three topologies.

Note that the term 'neighbouring state' introduced in the previous sections of this chapter will be used interchangeably with 'neighbour solution', which is intuitive and more appropriate when referring to an optimization problem such as the CMST.

3.6.1 - Generating a Neighbour Solution for the Bus and Loop Topologies.

The process of generating a neighbour solution for these topologies begins by selecting 2 terminals denoted Ti and Tj, where Ti ≠ Tj. Furthermore, let us denote Ti-1 as the *predecessor* of Ti on a line I originating at the centre. In other words, if nodes Ti-1 and Ti denote the end points of a common edge in a line rooted at the concentrator, then the path from the concentrator to node Ti-1 is shorter than from the concentrator to Ti. Similarly, Ti+1 denotes the *successor* of Ti.

By changing the link(s) that connect each of these two terminals to their respective lines we can consider two simple *'moves'* and one pairwise *'interchange'*. Of course, the terms 'moving' and 'interchanging' is figurative since the terminal locations are geographically fixed. The process is actually one of including or excluding terminals from lines by adding and dropping the links that connect them to their respective lines. By evaluating the change in total distance (cost) of ...

(1) including terminal Ti to the line containing terminal Tj at position Tj+1,

(2) including terminal Tj to the line containing terminal Ti at position Ti+1, and

(3) swapping both terminals

up to three potential neighbours can be generated.

Each new configuration produced in one of these ways that does not violate the topology nor the maximum line weight (Wmax) and maximum number of terminals per line constraints, is considered to be a valid neighbour solution. When a link is dropped, its cost is retrieved from the cost matrix and subtracted from the current cost, similarly when a link is created its cost must be added to the current cost. Through this amalgamate of subtractions and additions, the neighbour solution's cost is obtained. Needless to say that the greater the decrease in total cost generated by a neighbour solution, the higher its desirability.

The following figure illustrates how three neighbour solutions can be created from a pair of terminals (Ti and Tj) by manipulating the links connecting these terminals.

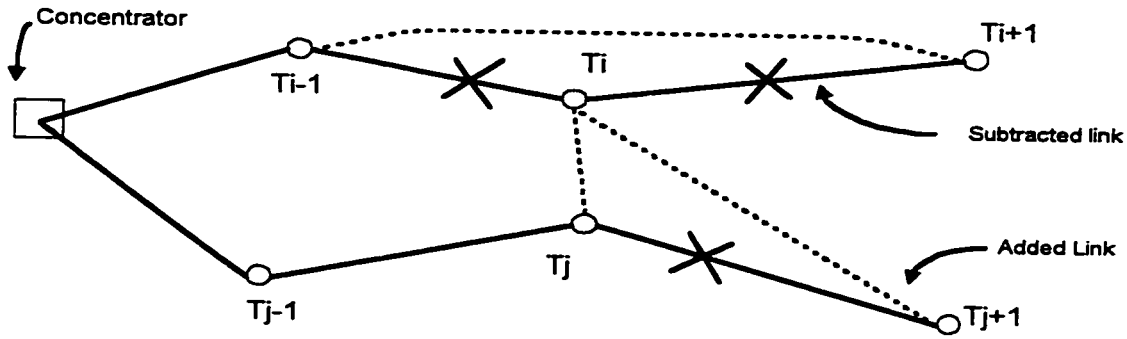


figure 3.2 Placing terminal Ti following terminal Tj.

The steps of figure 3.2 are as follows:

Step 1: If Ti is not the concentrator then goto step 2 else goto step 5.

Step 2: If the line containing Tj can accept the weight of Ti without violating Wmax then goto step 3 else goto step 5.

Step 3: Remove Ti from the line that contains it by subtracting the cost of the links (Ti-1,Ti) and (Ti,Ti+1)* and adding the cost of the link (Ti-1,Ti+1)* to the current cost.

Step 4: Connect Ti to the line containing Tj by adding the cost of the links (Tj,Ti) and (Ti,Tj+1)* and subtracting the cost of the link (Tj,Tj+1)*

Step 5 : Stop.

(*) : If this link exists.

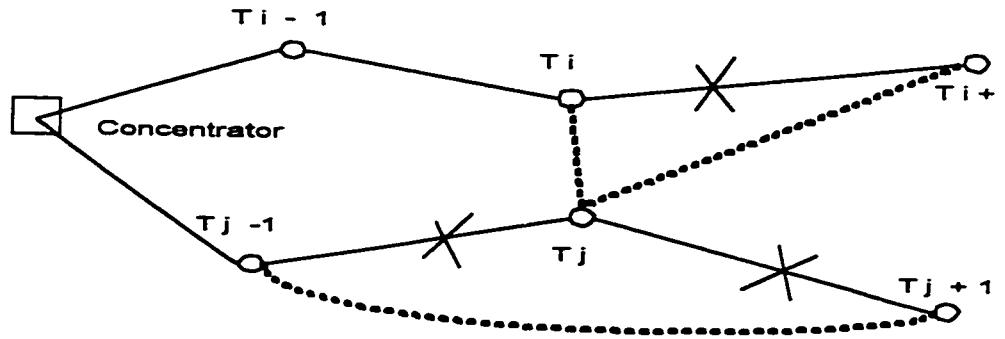


figure 3.3 Placing terminal Tj following terminal Ti.

The steps of figure 3.3 are as follows:

Step 1: If Tj is not the concentrator then goto step 2 else goto step 5.

Step 2: If the line containing Ti can accept the weight of Tj without violating Wmax then goto step 3 else goto step 5.

Step 3: Remove Tj from the line that contains it by subtracting the cost of the links (Tj-1,Tj) and (Tj,Tj+1)* and adding the cost of the link (Tj-1,Tj+1)* to the current cost.

Step 4: Connect Tj to the line containing Ti by adding the cost of the links (Ti,Tj) and (Tj,Ti+1)* and subtracting the cost of the link (Ti,Ti+1)*

Step 5 : Stop.

(*) : If this link exists.

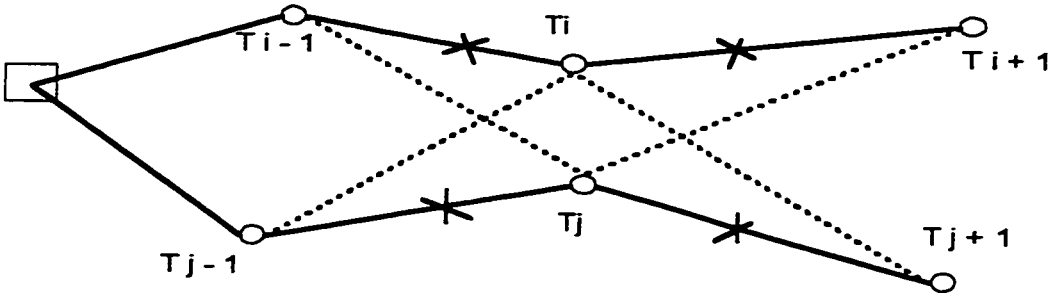


figure 3.4 Swapping terminal Ti and Tj,

The steps of figure 3.4 are as follows:

Step 1: If neither Ti nor Tj is the concentrator then goto step 2 else goto step 4.

Step 2: If the sum of the terminal weights on the line containing Ti minus the weight of Ti plus the weight of Tj is smaller than or equal to Wmax and

If the sum of the terminal weights on the line containing Tj minus the weight of Tj plus the weight of Ti is smaller than or equal to Wmax then goto step 3 else goto step 4.

Step 3: Subtract the cost of links (Ti-1,Ti), (Ti,Ti+1)*, (Tj-1,Tj), and (Tj,Tj+1)* from the current cost. Add the cost of links (Tj-1,Ti), (Ti-1,Tj), (Ti,Tj+1)*, and (Tj,Ti+1)* to the current cost.

Step 4: Stop.

(*) : If this link exists.

3.6.2 - Generating a Neighbour Solution for the Tree Topology.

For the tree topology, neighbour solutions are obtained by manipulating the order in which the Esau-Williams links terminals to one another to create multipoint lines. As the Esau-Williams algorithm proceeds, subtrees may fill up by reaching the maximum line weight constraint (Wmax) and/or a node may not accept any additional child nodes without violating the maximum number of terminals per line constraint. From this point on, the solution will start degenerating away from a minimum spanning tree since the algorithm is unable to make the best multipoint connections for the remaining terminals. Typically, links created towards the end of the run tend to be non optimal.

By forcing the Esau-Williams algorithm to link a node before it would normally be considered for connection it is possible to generate a neighbour solution with a lower cost.

To illustrate the point, consider the following Esau-Williams solution with Wmax is 3, maximum number of terminals per line is 3, and all terminals having a weight of 1 as depicted in figure 3.5 below.

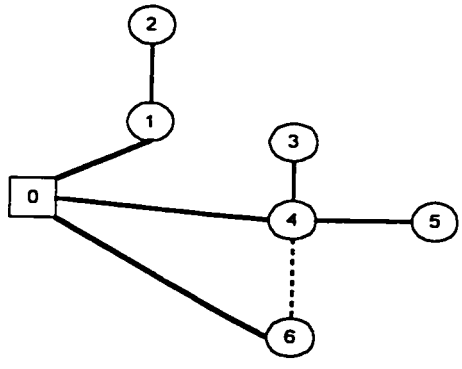


Figure 3.5 The Esau-Williams solution showing a capacitated minimum spanning tree.

Initially, all terminals are connected directly to the central node '0'. With each iteration of the Esau-Williams algorithm a new link is created merging terminals into components. By saving the order in which the algorithm links nodes together, a hierarchy of connections can be recorded. For the example above the connections were made as follows:

Iteration	Connection		Connection Cost
	From	To	
1	2	1	$20
2	5	4	$10
3	3	4	$5
4	1	0	$20
5	4	0	$40
6	6	0	$50
			$145

Note that replacing the link (0-6) by a link from terminal 6 to terminal 4 would yield an additional decrease in cost but cannot be done without violating both the *WMax* and the maximum number of terminals per line constraints.

By forcing the Esau-Williams algorithm to consider terminal 6 higher up in the hierarchy of connections a lower cost solution can be obtained as shown in figure 3.6 below.

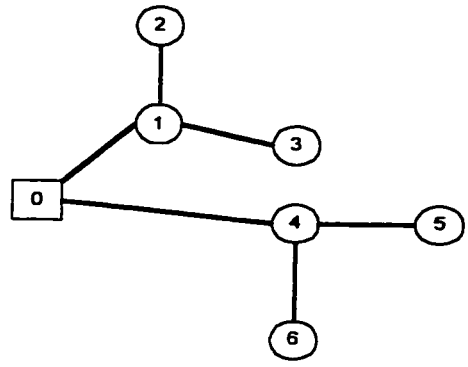


Figure 3.6 The modified Esau-Williams solution showing a capacitated minimum spanning tree with a lower cost.

This modified version of the Esau-Williams algorithm can be illustrated as follows:

Iteration	Connection		Connection Cost
	From	To	
1	2	1	$20
2	**6(forced)**	**4**	$25
3	5	4	$10
4	**3(*)**	**1**	$10
5	1	0	$20
6	4	0	$40
			$125

(*) : Since a connection from terminal 3 to terminal 4 would now violate the constraints, terminal 3 must be linked to terminal 1.

If we let **F** denote the iteration at which the forced connection was made, this method of generating neighbour solutions eventually breaks down to :

(1) For iteration = 1 to **F** -1, keep the Esau-Williams links, and destroy all links created from point **F** onward,

(2) Force a random yet unconnected node to be considered for connection at point **F**, and then

(3) Run the Esau-Williams algorithm for all remaining yet unconnected nodes.

CHAPTER 4 - DESCRIPTION AND IMPLEMENTATION OF OUR ALGORITHMS.

Seven SA programs were produced, three for the bus, three for the loop, and one for the tree topology. All of these applications start by using a heuristic algorithm to generate an initial solution. For the bus and tree topologies, the initial solution is obtained by running the Esau-Williams algorithm previously explained in 2.5.1.1. As for the loop, the initial solution is given by the Clarke-Wright algorithm introduced in 2.7.1.

Once an initial solution to the problem has been generated, the SA subroutine then attempts to improve upon it. The steps of each application will be given below starting with the ones implemented for the bus and loop topologies, followed by the one for the tree topology.

4.1 - Algorithms for the Bus and Loop Topologies.

For each of the topologies, three SA algorithms were developed. Each one uses a different method to select the number of neighbour solutions that will be created and evaluated.

The general format of the applications developed for the bus and loop topologies can be summarized in the following steps:

(1) Given the INPUT data.

A set of terminals represented by their (x,y) coordinates relative to a concentrator placed at point (0,0), that can be used to create a cost matrix,
or

A symmetric cost matrix representing the distances between all terminals (where 1 unit in distance is equivalent to 1 unit in cost).

(2) Calculate the INITIAL SOLUTION.

The Esau-Williams for the Bus Topology.
The Clarke-Wright for the Loop Topology.

(3) Run the SIMULATED ANNEALING Process. At each iteration repeat one of the following ...

Method 1 :

A set of randomly generated neighbour solutions, then choose the best neighbour among the generated set. If this solution's cost is smaller than the current cost then accept it as the current solution, otherwise accept it with a certain probability.

Method 2 :

A set containing all neighbour solutions derived by manipulating the links in 2 randomly selected lines, then choose the best neighbour among the generated set. If this solution's cost is smaller than the current cost then accept it as the current solution, otherwise accept it with a certain probability.

Method 3 :

Consider all possible neighbours that can be derived from the current solution and assign a probability to each possible neighbour . Then choose one based on its probability value, and make it the current solution.(Modification of the rejectionless method introduced by Greene and Supowit, 1986)

...until a stopping criterion has been reached.

(4) If the Simulated Annealing process has improved on the initial solution then display the improvement.(OUTPUT)

Although the implementation of any one of the methods shares similarities with the other two, all differ enough to warrant independent explanation in the following pages. This is because the data structures that are used to implement them are not only dependent on the type of network topology (Bus versus Loop) but also on the relevant information needed by each method to generate its set of neighbour solutions.

4.1.1. - Method (1) : Considering a Set of Neighbour Solutions Obtained from Randomly Selected Terminals.

The following steps describe how method 1 goes about generating a random set of neighbour solutions.

Step 1. Specify the number of times (m) you wish to repeat the process.

Step 2. From the current solution, select 2 terminals randomly among all terminals including the centre.

Step 3. Consider two moves and a pairwise interchange for the terminals, to generate up to a maximum of 3 neighbour solutions, and select the best among them.

Step 4. If any one of the solutions generated in step 3 is the best neighbour solution found so far then remember this neighbour solution.

Step 5. If steps 2 to 4 have been repeated m times then goto step 6, otherwise goto step 2.

Step 6. Stop.

With method (1) up to (3 * m) neighbour solutions are generated before the best among them is chosen.

4.1.2. - Method (2) : Considering a Set of All Possible Neighbour Solutions from Two Randomly Selected Lines.

The following steps describe how method 2 goes about finding a neighbour solution.

Step 1. Select 2 lines randomly among all lines from the current solution.

Step 2. For each pair of terminals, consider two moves and a pairwise interchange among the two lines, to generate up to a maximum of 3 neighbour solutions for every pair, and select the best among them.

Step 3. Stop.

With method (2), the maximum number of neighbour solutions that are generated depends on the number of terminals in each line. Since up to 3 neighbour solutions can be generated for each pair (Ti, Tj), the maximum number of neighbour solutions to be evaluated is given by:

3 * (Number of Terminals in line I - 1) * (Number of Terminals in line J - 1)

since no neighbours can be generated with the pair (0,0).

4.1.3 - Method (3) : Considering All Possible Neighbour Solutions.

Intuitively, before implementing this method it was believed that it would yield a better neighbour solution than simply selecting random pairs of terminals since all possible pairs could be considered before selecting the best one among them. Once implemented though under the 'regular' simulated annealing process (the Metropolis method), testing

revealed that it could eventually become trapped in a local minimum. To understand how this could occur let us recall that the simulated annealing process using the metropolis method automatically accepts a neighbour solution if its cost is lower but does not necessarily reject a neighbour solution with a higher cost. The following figure illustrates the circular loop the process may get trapped in.

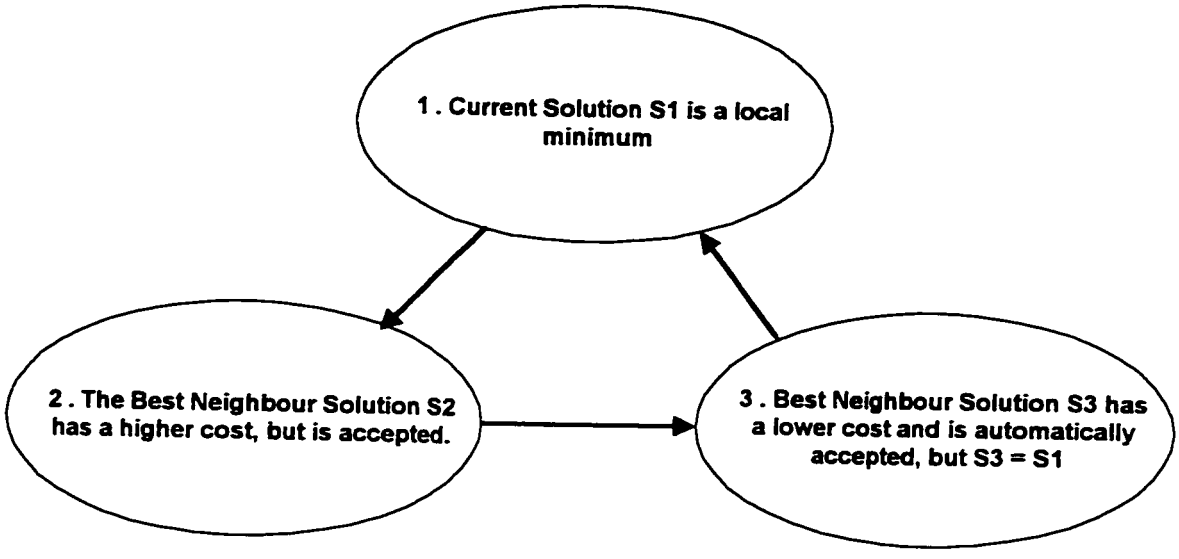


figure 4.1 A local minima trap.

By implementing this selection method under a variant of the simulated annealing process 'with rejectionless moves'(Greene and Supowit, 1986) instead, it was possible to alleviate this problem.

Recall that instead of automatically selecting the best solution among all neighbours and then considering whether to accept it or not, the rejectionless method assigns a probability to each solution. The better a solution, the higher its probability of being selected, and although all lower grade neighbour solutions have smaller probabilities

assigned to them, they may still be selected. Once a neighbour is selected it becomes the current solution.

The following steps describe how Method 3 finds the best neighbour solution among all possibilities:

Step 1. Evaluate all possible neighbour solutions by considering all possible moves and pairwise interchanges for the current solution.

Step 2. Assign a probability to each solution based on its quality relative to all other solutions.

Step 3. Select a solution generated in step 2 based on its probability.

Step 4. Stop.

4.2 - An Algorithm for the Tree Topology.

The SA process implemented for the tree topology is given in the following general steps:

step 1. Generate an initial solution using of the Esau-Williams algorithm and remember the order in which the connections were made.

step 2. Initialize the SA parameters.

step 3. Choose a connection from the current solution representing a link from terminal i to terminal j but this may not be the optimal connection for terminal i.

step 4. Run the modified Esau-Williams algorithm which is forced to consider terminal i the moment it would normally be considered for connection into the network.

step 5 If the modified Esau-Williams algorithm generates a lower cost then goto step 7. If a lower solution was not found then goto step 6.

step 6. Accept the solution with a certain probability generated by the function $\mathbf{exp}(-\Delta C_i / T)$, where ΔC_i corresponds to the change in total cost if the link connecting terminal i into the network has changed. If the solution is rejected then goto step 9.

step 7. Make the modified Esau-Williams solution the current solution.

step 8. If the stopping criteria has been reached then goto step 10.

step 9. Adjust the SA parameters(decrement the repetition counter or reset the repetition counter and the decrement the temperature). Goto step 3.

step 10. Stop.

4.3 - Data Structures for the Bus and Loop Topologies.

4.3.1 - Data Structures Used to Represent the Bus and Loop Network Topology in all Three Methods.

To illustrate the data structure that was used to represent the bus and loop topologies consider the following figure illustrating a bus network made up of 7 terminals divided up into 3 bus lines.

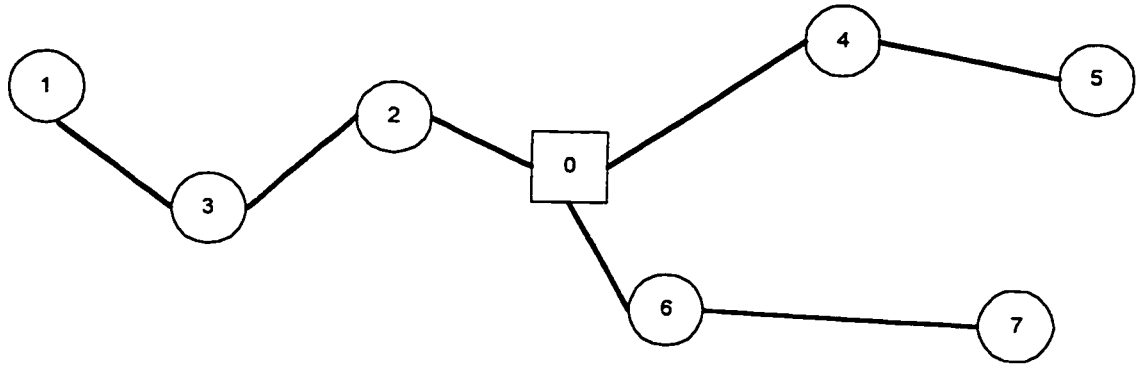


figure 4.2 Illustration of a bus network.

This preceding network is represented using a linked list as follows :

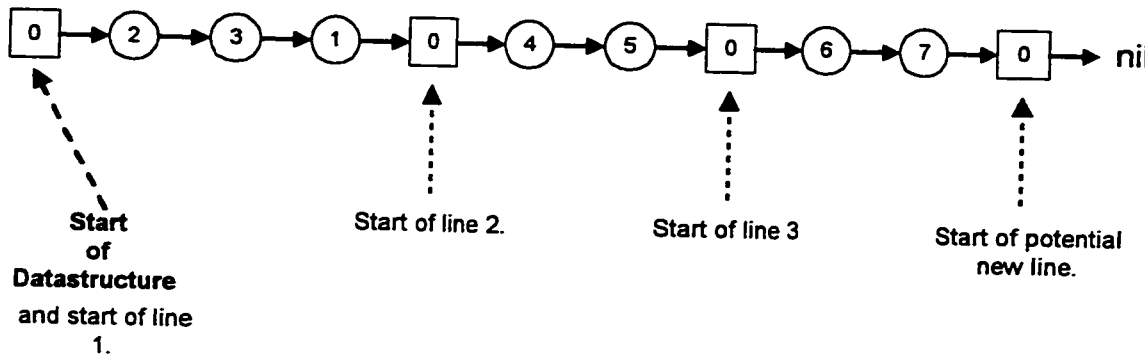


figure 4.3 Data structure representing a bus network.

Each data structure element contains the following fields:

Terminal_Number	: integer;
Weight	: integer; (if the data structure element is '0' the weight field value represents the cumulative weight of all the terminals on that line, if the data structure is non zero and therefore a terminal, the weight field value simply reveals the weight of the terminal.
Number_of_Terminals_in_Line	: integer; (used only if data structure element is '0').
Next	: pointer;

The concentrator is represented by '0' and thus a beginning of a line. The arrows represent pointers to the next element of the linked list.

Each linked list (line) starts with an element representing the concentrator node ([0]). This element denoted with a Terminal_Number equal to 0, contains information on the number of terminals in the line, the weight of the line, and uses a pointer to the next element (terminal) in the list.

Each element representing a terminal node on the line ((*i*)), contains the terminal number (*i*), the weight of the terminal, and use a pointer to represent the neighbouring terminal on the line. Finally, the Number_of_Terminals field for data structure elements representing terminals is redundant and therefore set to zero.

Finally, an additional data element positioned at the end of the data structure represents a potential new line that can be created by linking a terminal directly to the concentrator.

The following describes the process of finding 2 random terminals for **method 1**:

1. Line_I := A line randomly selected from 1 to Lastline
2. Line_J := A line randomly selected from 1 to Lastline
3. Position Ti at element [0] in linked list representing Line_I
4. Position Tj at element [0] in linked list representing Line_J
5. Move Ti a random number of times down Line_I, where the number of times is a random value not greater than Number_of_Terminals_in_Line I.
6. Move Tj a random number of times down Line_J, where the number of times is a random value not greater than Number_of_Terminals_in_Line J.

For **method 2** the following figure illustrates the process of combining all terminals from line I and J to create the pairs of terminals that will generate the neighbour solutions.

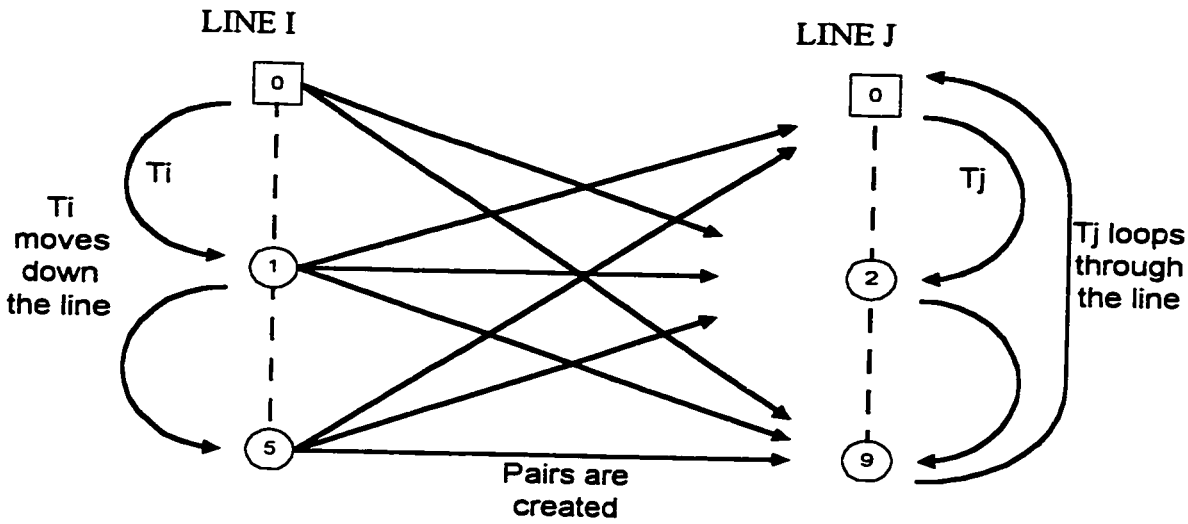


Figure 4.4 Neighbour solution generating process used by method 2.

With the pairs (T_0,T_2),(T_0,T_9),(T_1,T_0),(T_1,T_2),(T_1,T_9),(T_5,T_0),(T_5,T_2), and (T_5,T_9), a set of 16 potential neighbour solutions can be evaluated so as to determine whether they do not violate the maximum line weight constraint (Wmax) or not:

For pair (T_0,T_2) : 1. moving T_2 to line I between the concentrator and T_1.

For pair (T_0,T_9) : 2. moving T_9 to line I between the concentrator and T_1.

For pair (T_1,T_0) : 3. moving T_1 to line J between the concentrator and T_2.

For pair (T_1,T_2) : 4. moving T_1 to line J between T_2 and T_9.

5. moving T_2 to line I between T_1 and T_5.

6. swapping T_1 and T_2.

For pair (T_1,T_9) : 7. moving T_1 to line J following T_9.

8. moving T_2 to line I between T_1 and T_5.

9. swapping T_1 and T_9.

For pair (T_5,T_0) :

10. moving T_5 to line J between the concentrator and T_2.

For pair (T_5,T_2) :

11. moving T_5 to line J between T_2 and T_9.
12. moving T_2 to line I following T_5.
13. swapping T_5 and T_2.

For pair (T_5,T_9) :

14. moving T_5 to line J following T_9.
15. moving T_2 to line I following T_5.
16. swapping T_5 and T_9.

From the set of acceptable neighbours, only the lowest cost solution will be considered for acceptance or rejection by the simulated annealing process.

As for **method 3**, since all possible pairs of terminals are considered for this method, no search procedure is needed.

The following describes the steps of this method:

Step 1 : Set Ti equal to Start, set Tj equal to Ti.NextElement.

Step 2 : If Ti $\neq$ nil then goto step 3 else goto step 5.

Step 3 : Evaluate the neighbour solutions obtained by

1) placing Ti following Tj (If Ti $\neq$ 0),

2) placing Tj following Ti (If Tj $\neq$ 0), and

3) swapping Ti and Tj (If both Ti and Tj $\neq$ 0)

and move Tj to the next element in the data structure.

Repeat this step until Tj reaches the end of the line (Tj = nil).

Step 4 : Move Ti to Ti.NextElement.

Move Tj to Ti.NextElement.

Goto step 2.

Step 5 : Stop.

Graphically, the iterations of method 3 can be represented as follows:

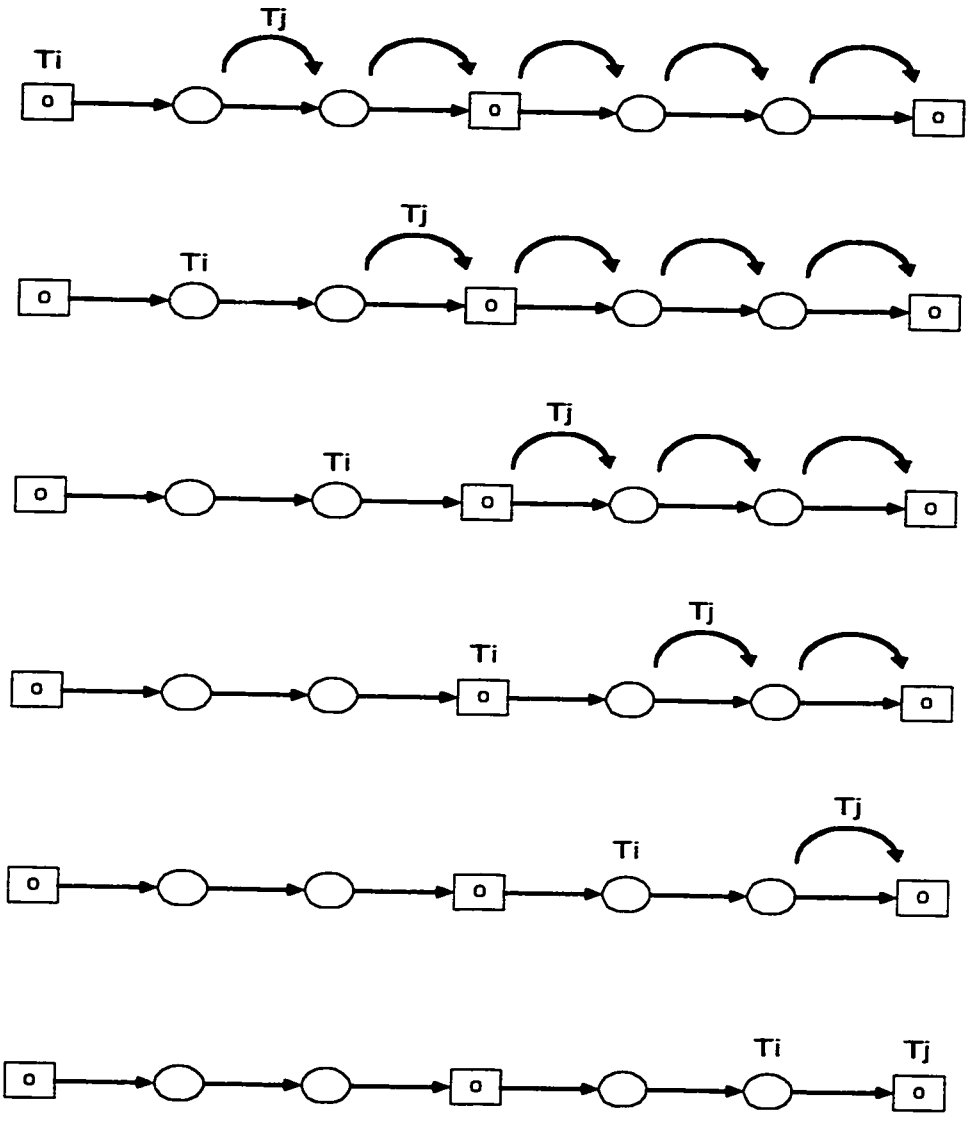


Figure 4.5 Neighbour solution generating process used by method 3.

4.3.2 - An Additional Data Structure Needed for Method 3.

Recall that in method 3 we use the 'rejectionless method'. All possible neighbour solutions must be evaluated before one is chosen depending on its probability relative to all other neighbour solutions. This chosen neighbour solution is never rejected and becomes the current solution of the SA process. Therefore, a linked list was needed to hold the relevant information of all neighbour solutions before one could be chosen. Each element of the list represents a potential neighbour solution. Recall that up to 3 neighbour solutions can be generated for each pair of terminals. Let us denote (Z) as being the set of possible neighbour solutions. Each element z is represented as a record, where the fields of each element contain the parameters needed by the program to easily and promptly change the current solution to the neighbour solution in the case where it would be selected. This linked list is created at each repetition (k) of the simulated annealing process, once a neighbour solution is selected the list is then disposed (erased) to free up the computer's memory. Each element of the list contains the following fields :

```
NeighborInfoType=RECORD
        LineI           : NodePointerType;
        LineJ           : NodePointerType;
        Ti              : NodePointerType;
        Tj              : NodePointerType;
        NeighbourType   : Byte;
        NeighbourCost   : LongInt;
        α_i             : Real;
```

F_i : Real;

Following : NeighborInfoPointerType;

END;

Fields LineI and LineJ are pointers to elements in the data structure representing the topology, these elements represent the beginning of the lines containing *Terminal i* and *Terminal j* respectively.

Fields Ti and Tj are pointers to the *terminals i* and *j* in the data structure representing the topology.

The field 'NeighbourType' holds a value of 1,2, or 3, representing the type of exchange that is needed to modify the current solution to become that selected neighbour solution (where the value 1 corresponds to putting Ti at position Tj+1 on line J, the value 2 corresponds to putting Tj at position Ti+1 on line I , and the value 3 corresponds to swapping the terminals).

The 'NeighbourCost' field expresses the cost of the particular neighbour solution.

The pointer 'Following' simply points to the *next neighbour solution* in the list of solutions.

The field 'α_i' holds the probability value calculated by the 'regular' simulated annealing acceptance probability function (α_i = exp((Neighbour Cost -Current Cost)/Temperature) for solutions where the neighbour cost is superior to the current cost. For neighbours having a lower cost than the current solution α_i is assigned a value equal to 1. Recall that

in this instance, the neighbour solution would be automatically accepted under the 'regular' simulated annealing process proposed by Metropolis et al. [1953].

Once all feasible neighbour solutions have been identified, all values for α_i are summed up with one 'pass' through the linked list. Each element z of the list is then assigned a value for 'p_i' by the function

$$p_i = \frac{\alpha_i}{\sum_{j=1}^{m} \alpha_i}$$

Finally, the cumulative probabilities F_i are calculated before the pseudo-random generator is invoked. Consider the following figure showing an example to illustrate how probabilities are assigned. Three neighbour solutions are considered, the current cost is \$60, and temperature is currently equal to *10*.

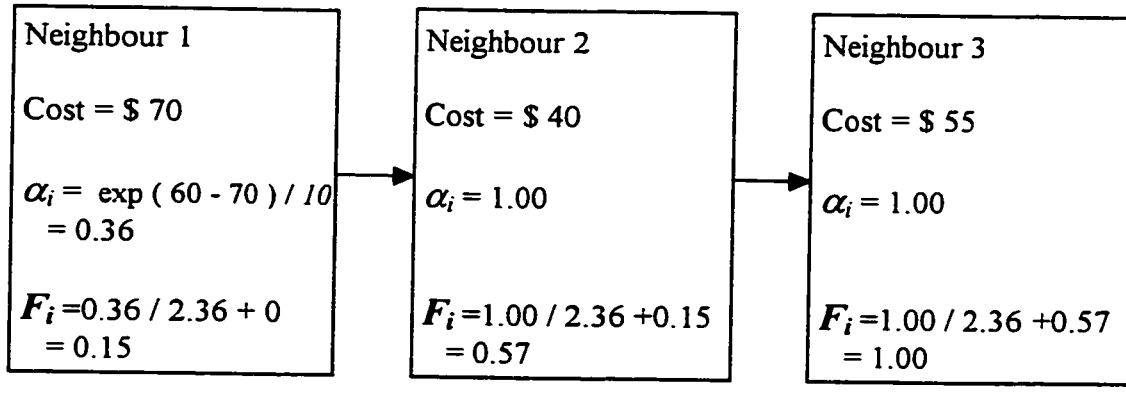


figure 4.6 Assigning probabilities to neighbour solutions for method 3.

A random probability value is generated. If the value ranges from 0 to 0.15 then neighbour 1 is selected, if it is larger than 0.15 but smaller than 0.57 then neighbour 2 is selected, and if the random probability is above 0.57 then neighbour 3 is chosen.

4.3.3 - Data Structure to Represent the Tree Network Topology.

Recall that the Esau-Williams algorithm initially assumes a star topology where each node represents a component. The algorithm then merges components together according to a certain trade-off function. An array of n records provides a simple way of keeping track of how these components are merged at each iteration of the algorithm. Each record [i], where i represents the terminal number and can have a value of 1 to n, where n represents the number terminals not including the central node contains the following fields:

```
component        : integer;
SumOfW           : integer;
NbOfChildNodes   : integer;
```

Each 'component' field is initialized with a value equal to i, the number of the sole terminal that it initially contains. 'SumOfW' represents the aggregate weight of all the terminals in the component where terminal i is included. Initially, this value is set to equal the weight of terminal i since it is the only terminal of the component. 'NbOfChildNodes' denotes the number of child terminals in the component and is initially set to 1.

Two arrays of size n record the hierarchy of connections generated by the Esau-Williams algorithm and its modified version. An additional two arrays store the current SA solution and the best SA solution found so far. Each element of these arrays contain the following two fields.

```
From  : integer;
To    : integer;
```

The SA process implemented for the tree topology is given in the following steps with an example of a network consisting of 6 terminals numbered from 1 to 6, where '0' represents the central site:

step 1. Generate an initial solution using the Esau-Williams algorithm and store the hierarchy of its connections in the EWSolution array.

example:

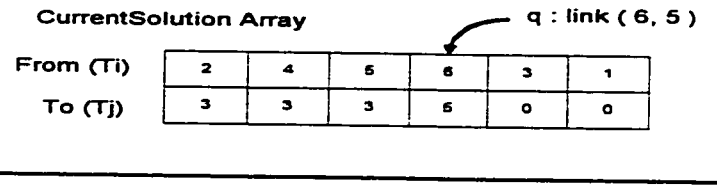
EwSolution Array

From (Ti)	2	4	5	6	3	1
To (Tj)	3	3	3	5	0	0

step 2. Copy the elements of the EWSolution array into the CurrentSolution array.

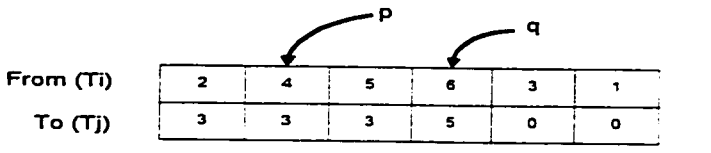
step 3. Choose an element 'q' from the CurrentSolution array that represents the link (terminal i_q - terminal j_q) but is not the optimal connection for terminal i_q into the network.

example:



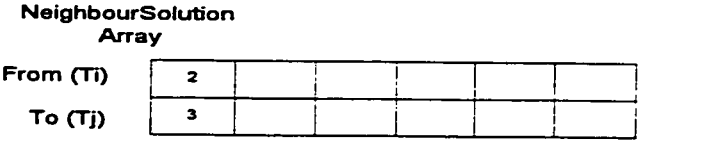
step 4. Select an element 'p' from the CurrentSolution array, where $p<q$.

example:



step 5. Copy all elements preceding element p from the CurrentSolution array into the NeighbourSolution array, making these links part of the neighbour solution.

Example:



step 6. Make terminal i_p = terminal i_q in the NeighbourSolution array, and find the best available link (terminal i_p - terminal j_p) that does not violate any constraints.

example:

NeighbourSolution Array

i_p=6, j_p=3

From (Ti)	2	6				
To (Tj)	3	3				

step 7. Run the Esau-Williams algorithm after forcing the link in step 6 for the terminals that have not yet been linked.

example:

NeighbourSolution Array

From (Ti)	2	6	4	5	3	1
To (Tj)	3	3	5	0	0	0

step 8. If the links in the NeighbourSolution array represents a lower cost solution than those in the CurrentSolution array then goto step 10. If a lower cost solution was not found then goto step 9.

step 9. Accept the solution with a certain probability generated by the function $\exp(-\Delta C_i / T)$, where ΔC_i corresponds to the change in total cost if the link connecting terminal i into the network has changed. If the solution is rejected then goto step 12.

step 10. Make the neighbour solution the current solution by copying the elements of the NeighbourSolution array into the CurrentSolution array.

step 11. If the new solution is the least costly solution found so far then save it as the best SA solution.

step 12. If the stopping criteria has been reached then goto step 14.

step 13. Adjust the SA parameters (decrement the repetition counter or reset the repetition counter and decrement the temperature). Goto step 3.

step 14. Stop.

4.4 - Pseudo-Code Describing the Algorithms.

4.4.1 - For SA Implemented For the Bus and Loop Topologies.

In the following pages we use pseudo-PASCAL to show the simulated annealing programs that were developed for the various topologies.

The terms used in the pseudo-code below were chosen carefully in order to be self-explanatory to readers with no background in programming languages.

In addition, note that an alternative stopping criterion was introduced for the bus and loop programs in order to shorten the execution times. The alternative criterion (called 'max_no_improve') consists of stopping the process after no improvement has occurred over a number of successive temperature decreases. As we shall see later, testing revealed that this strategy was effective once the temperature neared zero where most of the program's time is spent rejecting lower grade solutions.

4.4.1.1 - For SA Implemented Under Method (1).

Recall that method (1) consists of finding the best neighbour solution from a specified number of pairs of terminals. For each pair, up to three neighbour solutions are generated by considering two moves and an exchange.

For this method the steps of the Simulated Annealing process can be described in Pseudo-PASCAL as follows:

Begin (Method 1)

step 1. Start with the initial topology;

step 2. *Initialization of Parameters.* Assign values to the simulated annealing parameters (Temperature, Alpha, Epsilon, Number_of_repetitions, and max_no_improve. Assign a value to number_of_terminal_pairs_to_consider).
Set *r* equal to 0.

step 3. **While** (Temperature > than Epsilon) **or** (*r* <= max_no_improve) **do**
Begin
Set k equal to Numbers_of_repetitions;
Set m equal to number_of_terminal_pairs_to_consider;
While k > than 0 **do**
Begin
Repeat (m) times
Begin
find_two_Random_Terminals_Ti_and_Tj
and
Consider_All_Possible_Neighbours_For_Ti_and_Tj;
End

Select_the_Best_Neighbour_from_the_set_of_Solutions;

If Best_Neighbour_Solution's_Cost is smaller than the
Current_Solution's_Cost
then Change_To_Selected_Neighbour
else If we accept with a certain probability
then Change_To_Selected_Neighbour;

if the cost has not gone down then increment (*r*);

decrement (k);
End;

Temperature:=Temperature*Alpha;
End;

End(Method 1).

4.4.1.2 - For SA Implemented Under Method (2).

For this method the steps of the Simulated Annealing process can be described in Pseudo-PASCAL as follows:

```
Begin (Method 2)

step 1. Start with the initial topology;

step 2. Initialisation of Parameters. Assign values to the simulated annealing parameters
        (Temperature, Alpha, Epsilon, Numbers_of_repetitions, and max_no_improve).
        Set r = 0.

step 3. While (Temperature > than Epsilon) or (r<=max_no_improve) do
        Begin
              Set k equal to Numbers_of_repetitions;
              While k > than 0 do
              Begin
                    Select_Two_Random_Lines_I_and_J
                    Evaluate_All_Possible_Neighbours_for_each
                                Combination_of_Ti_and_Tj
                    Select_the_Best_Neighbour_from_the_set_of_Solutions;

                    If Best_Neighbour_Solution's_Cost is smaller than the
                          Current_Solution's_Cost
                          then Change_To_Selected_Neighbor
                    else If we accept with a certain probability
                          then Change_To_Selected_Neighbor;

                    if the cost has not gone down then increment (r) ;

                    decrement (k);
              End;

              Temperature:=Temperature*Alpha;
        End;

End(Method 2).
```

4.4.1.3 - For SA Implemented Under Method (3).

For this method the steps of the Simulated Annealing process can be described in Pseudo-PASCAL as follows:

Begin (Method 3)

step 1. Start with the initial topology;

step 2. *Initialization of Parameters.* Assign values to the simulated annealing parameters (Temperature, Alpha, Epsilon, Numbers_of_repetitions, and Max_no_improve). Set $r = 0$;

step 3. **While** (Temperature > than Epsilon) **or** (r<=Max_no_improve) **do**
Begin
Set k equal to Numbers_of_repetitions;
While k > than 0 **do**
Begin
Evaluate_All_Possible_Neighbours
Assign_a_Probability_of_being_Selected_to_each_Neighbour
Select_a_Neighbour_Solution_Based_on_its_Probability
Change_To_Selected_Neighbor

if the cost has not gone down then increment(r);

decrement (k);
End;
Temperature:=Temperature*Alpha;
End;

End(Method 3).

4.4.2 - For SA Implemented For the Tree Topology.

Recall that for the tree topology we store the hierarchy of connections obtained while generating solutions in arrays. As the following example illustrates, link (2,3) in element 1 of the array is the first link created, while (1,0) at element 6 in the array is the last.

example:

EwSolution Array

From (Ti)	2	4	5	6	3	1
To (Tj)	3	3	3	5	0	0

Note that the terminals considered towards the end of the process by the Esau-Williams algorithm have no choice but to connect to the '0'. Often enough, this is because the multipoint lines being created around them are being filled up and cannot accept additional terminals without violating the capacity constraint.

The method that was developed for generating neighbour solutions for the tree topology consists of changing the order in which the Esau-Williams algorithm creates its links. For example, terminal 1 instead of being considered for connection at iteration 6 of the Esau-Williams algorithm as illustrated above, would be linked anywhere between iterations 1 to 5. Thus, instead of having to link terminal '1' to the centre '0', a possibility would now exist to find a lower cost link to a previously unavailable multipoint line.

The following steps in Pseudo-PASCAL describe the process:

Begin (Tree topology)

step 1. Start with the Esau-William topology constrained by maximum line weight (Wmax) and by the number of child nodes any terminal node may have;

step 2. *Initialization of Parameters.* Assign values to the simulated annealing parameters (Temperature, Alpha, Epsilon, Numbers_of_repetitions).

step 3. **While** (Temperature > than Epsilon) **do**

Begin
Set k equal to Numbers_of_repetitions;

While k > than 0 **do**
Begin
Select_a_Random_Link_(i_q, j_q)_at _Element_q _of_the_CurrentSolution_Array;
Select_an_Element_ p_From_the_CurrentSolution_Array;(where p<q)
Copy_all_Elements_Preceeding_ p_to_the_NeighbourSolution_Array;
Find_the_Best_Link for terminal i_q ;
Run_Esau-Williams_for_the_Remaining_Terminals;

If Neighbour_Solution's_Cost is smaller than the
Current_Solution's_Cost **then** Change_To_Neighbor
else If we accept with a certain probability **then** Change_To_Neighbor;

decrement (k);
End;

Temperature:=Temperature*Alpha;
End;

CHAPTER 5 - COMPUTATIONAL RESULTS AND ANALYSIS.

Data sets representing 50, 100, 150, 200, and 250 terminals were created. In all, 20 sets were produced, 4 in each size category. All data sets were generated at random to represent the xy-coordinates[1] of the terminals relative to a concentrator positioned at (0,0). In addition, each terminal (T_i) was randomly[2] assigned a weight (w_i), with an average w_i for all data sets equal to 4.

For the loop and bus topologies, the problems were constrained by imposing a limit on the maximum number of terminals any line could carry in addition to a restriction on the maximum data transmission capacity of each line (Wmax) which was the same for each line. For the bus topology, the 'maximum number of terminals per line' constraint was ignored and replaced by a limit on the number of child nodes any one terminal in the tree could have with exception of the concentrator.

For the loop and bus topologies, the 'maximum number of terminals per line' constraint was set to 12 after making sure that this value actually did constrain the Esau-Williams and Clarke-Wright algorithms somewhat. For the tree topology, the 'maximum number of child nodes' was limited to 3. For all topologies the 'maximum line weight' constraint (Wmax) was set to 32, thus, on average we would expect resulting topologies consisting of about 8 terminals per line (Wmax / average w_i = 32 / 4). All programs were run by specifying a random seed value of 2222 on a 486 DX-2 66MHz computer with 8 megs of ram.

---

[1] Random values for x were limited to [-309, 309], and [-174, 174] for y.
[2] Random weight assignments were generated between [1, 8].

The following pages include tables illustrating the results obtained from our simulated annealing algorithms. The tables include (among other relevant information) the percentage of average savings obtained and the SA running time for the given annealing schedule. The schedules were obtained by comparing the results obtained from various combinations of parameters settings. From this process of trial_and_error, the annealing schedule offering the most savings for each problem size was selected. The following tables summarize the results obtained using the selected schedules.

5.1 - Results for the Bus Topology (Method 1).

For the bus topology, the results of our simulated annealing algorithm using method 1 (randomly selecting a specified number of terminal pairs at each repetition) are given below.

For the four data sets representing the xy-coordinate locations of 50 terminals the results are...

table 5.1 Results for the bus topology (method 1) for data sets consisting of 50 terminals.

n = 50	Esau Williams Cost	Simulated Annealing Cost	Savings	Esau Williams Time	Simulated Annealing Time	*Average Improvement %*	*Annealing Schedule (Parameters)*
50_1	3094	2873	221	0	12 sec		*Temperature = 14*
50_2	3027	2795	232	0	29 sec	**5.87 %**	*Alpha = .990*
50_3	2947	2758	189	0	35 sec		*Epsilon = .01*
50_4	3324	3238	86	0	31 sec		*Repetition = 80*
							Pairs Considered = 40
							Alternate Stopping Criteria = 30

For the four data sets representing the xy-coordinate locations of 100 terminals the results are...

Table 5.2 Results for the bus topology (method 1) for data sets consisting of 100 terminals.

n = 100	Esau Williams Cost	Simulated Annealing Cost	Savings	Esau Williams Time	Simulated Annealing Time	*Average Improvement %*	*Annealing Schedul (Parameters)*
100_1	5435	4867	568	1 sec	3 min 12 sec	**9.59 %**	*Temperature = 13*
100_2	4905	4208	697	0	6 min 26 sec		*Alpha = .998*
100_3	4996	4882	114	0	1 min 44 sec		*Epsilon = .01*
100_4	5194	4605	589	0	8 min 53 sec		*Repetition = 95*
							Pairs Considered = 5
							Alternate Stopping Criteria = 110

For the four data sets representing the xy-coordinate locations of 150 terminals the results are...

Table 5.3 Results for the bus topology (method 1) for data sets consisting of 150 terminals.

n = 150	Esau Williams Cost	Simulated Annealing Cost	Savings	Esau Williams Time	Simulated Annealing Time	*Average Improvement %*	*Annealing Schedule (Parameters)*
150_1	6365	5848	517	3 sec	9 min 11 sec	**6.21 %**	*Temperature = 15*
150_2	7284	6597	687	3 sec	5 min 40 sec		*Alpha = .993*
150_3	6690	6415	275	3 sec	9 min 04 sec		*Epsilon = .01*
150_4	6897	6684	213	3 sec	4 min 21 sec		*Repetition = 100*
							Pairs Considered = 50
							Alternate Stopping Criteria = 150

For the four data sets representing the xy-coordinate locations of 200 terminals the results are...

Table 5.4 Results for the bus topology (method 1) for data sets consisting of 200 terminals.

n = 200	Esau Williams Cost	Simulated Annealing Cost	Savings	Esau Williams Time	Simulated Annealing Time	*Average Improvement %*	*Annealing Schedule (Parameters)*
200_1	8281	7829	452	8 sec	9 min 16 sec	**7.17 %**	*Temperature = 10* *Alpha = .991* *Epsilon = .01* *Repetition = 180* *Pairs Considered = 85* *Alternate Stopping Criteria = 20*
200_2	9004	8241	763	9 sec	6 min 53 sec		
200_3	8778	7799	979	10 sec	6 min 16 sec		
200_4	8910	8598	312	10 sec	3 min 12 sec		

For the four data sets representing the xy-coordinate locations of 250 terminals the results are...

Table 5.5 Results for the bus topology (method 1) for data sets consisting of 250 terminals.

n = 250	Esau Williams Cost	Simulated Annealing Cost	Savings	Esau Williams Time	Simulated Annealing Time	*Average Improvement %*	*Annealing Schedule (Parameters)*
250_1	10143	9305	838	17 sec	32 min 22 sec	**6.45 %**	*Temperature = 7* *Alpha = .993* *Epsilon = .01* *Repetition = 200* *Pairs Considered = 60* *Alternate Stopping Criteria = 120*
250_2	10132	9495	637	18 sec	31min 28 sec		
250_3	10101	9403	698	18 sec	32 min 26 sec		
250_4	10249	9802	447	19 sec	29 min 08 sec		

5.2 - Results for the Bus Topology (Method 2).

For the bus topology, the results of our simulated annealing algorithm using method 2 (randomly selecting two lines and considering all possible pairs of terminals that can be combined from these two at each repetition) are given below.

For the four data sets representing the xy-coordinate locations of 50 terminals the results are...

Table 5.6 Results for the bus topology (method 2) for data sets consisting of 50 terminals.

N = 50	Esau Williams Cost	Simulated Annealing Cost	Savings	Esau Williams Time	Simulated Annealing Time	*Average Improvement %*	*Annealing Schedu (Parameters)*
50_1	3094	2863	231	0	23 sec	**5.74 %**	*Temperature = 7*
50_2	3027	2961	66	0	24 sec		*Alpha = .999*
50_3	2947	2647	300	0	23 sec		*Epsilon = .01*
50_4	3324	3210	114	0	19 sec		*Repetition = 50*
							Alternate Stopping Criteria = 180

For the four data sets representing the xy-coordinate locations of 100 terminals the results are...

Table 5.7 Results for the bus topology (method 2) for data sets consisting of 100 terminals.

n = 100	Esau Williams Cost	Simulated Annealing Cost	Savings	Esau Williams Time	Simulated Annealing Time	*Average Improvement %*	*Annealing Schedul (Parameters)*
100_1	5435	4814	621	0	57 sec	**9.83 %**	*Temperature = 11*
100_2	4905	4441	464	0	58 sec		*Alpha = .999*
100_3	4996	4366	630	0	1 min 09 sec		*Epsilon = .01*
100_4	5194	4890	304	0	33 sec		*Repetition = 80*
							Alternate Stopping Criteria = 110

For the four data sets representing the xy-coordinate locations of 150 terminals the results are...

Table 5.8 Results for the bus topology (method 2) for data sets consisting of 150 terminals.

n = 150	Esau Williams Cost	Simulated Annealing Cost	Savings	Esau Williams Time	Simulated Annealing Time	*Average Improvement %*	*Annealing Schedule (Parameters)*
150_1	6365	6246	119	3 sec	36 sec	**10.20 %**	*Temperature = 20*
150_2	7284	6306	978	3 sec	2 min 45 sec		*Alpha = .999*
150_3	6690	5840	850	3 sec	1 min 18 sec		*Epsilon = .01*
150_4	6897	6066	831	3 sec	1 min 43 sec		*Repetition = 92*
							Alternate Stopping Criteria = 200

For the four data sets representing the xy-coordinate locations of 200 terminals the results are...

Table 5.9 Results for the bus topology (method 2) for data sets consisting of 200 terminals.

n = 200	Esau Williams Cost	Simulated Annealing Cost	Savings	Esau Williams Time	Simulated Annealing Time	*Average Improvement %*	*Annealing Schedu (Parameters)*
200_1	8281	7373	908	8 sec	2 min 11 sec	**10.54 %**	*Temperature = 12*
200_2	9004	8000	1004	9 sec	1 min 46 sec		*Alpha = .999*
200_3	8778	7887	891	9 sec	2 min 38 sec		*Epsilon = .01*
200_4	8910	8027	883	10 sec	1 min 26 sec		*Repetition = 110*
							Alternate Stopping Criteria = 220

For the four data sets representing the xy-coordinate locations of 250 terminals the results are...

Table 5.10 Results for the bus topology (method 2) for data sets consisting of 250 terminals.

n = 250	Esau Williams Cost	Simulated Annealing Cost	Savings	Esau Williams Time	Simulated Annealing Time	*Improvement %*	*Annealing Schedule (Parameters)*
250_1	10143	9179	964	18 sec	2 min 04 sec	**7.16 %**	*Temperature = 11*
250_2	10132	9048	1048	19 sec	4 min 16 sec		*Alpha = .999*
250_3	10101	9659	442	19 sec	1 min 19 sec		*Epsilon = .01*
250_4	10249	9795	454	20 sec	3 min 16 sec		*Repetition = 100*
							Alternate Stopping Criteria = 220

5.3 - Results for the Bus Topology (Method 3).

For the bus topology, the results of our simulated annealing algorithm using method 3 (a systematic approach of considering all possible pairs of terminals to create potential neighbour solutions) are given below.

For the four data sets representing the xy-coordinate locations of 50 terminals the results are...

Table 5.11 Results for the bus topology (method 3) for data sets consisting of 50 terminals.

n = 50	Esau Williams Cost	Simulated Annealing Cost	Savings	Esau Williams Time	Simulated Annealing Time	*Average Improvement %*	*Annealing Schedule (Parameters)*
50_1	3094	2795	232	0	37 sec		*Temperature = 6*
50_2	3027	2780	247	0	57 sec		*Alpha = .900*
50_3	2947	2757	190	0	1 min 40 sec	**5.92 %**	*Epsilon = .01*
50_4	3324	3260	64	0	1 min 03 sec		*Repetition = 50*
							Alternate Stopping Criteria = 2

For the four data sets representing the xy-coordinate locations of 100 terminals the results are...

Table 5.12 Results for the bus topology (method 3) for data sets consisting of 100 terminals.

n = 100	Esau Williams Cost	Simulated Annealing Cost	Savings	Esau Williams Time	Simulated Annealing Time	*Average Improvement %*	*Annealing Schedule (Parameters)*
100_1	5435	4962	473	1 sec	4 min 16 sec	**4.99 %**	*Temperature = 6* *Alpha = .990* *Epsilon = .01* *Repetition = 100* *Alternate Stopping Criteria = 2*
100_2	4905	4762	123	0	4 min 51 sec		
100_3	4996	4877	119	0	7 min 12 sec		
100_4	5194	4885	309	0	8 min 21 sec		

For the four data sets representing the xy-coordinate locations of 150 terminals the results are...

Table 5.13 Results for the bus topology (method 3) for data sets consisting of 150 terminals.

n = 150	Esau Williams Cost	Simulated Annealing Cost	Savings	Esau Williams Time	Simulated Annealing Time	*Average Improvement %*	*Annealing Schedule (Parameters)*
150_1	6365	6159	206	3 sec	34 min	**4.05 %**	*Temperature = 7* *Alpha = .900* *Epsilon = .01* *Repetition = 50* *Alternate Stopping Criteria = 15*
150_2	7284	6889	395	4 sec	25 min 55 sec		
150_3	6690	6554	136	3 sec	51 min 45 sec		
150_4	6897	6531	366	3 sec	39 min 43 sec		

For the four data sets representing the xy-coordinate locations of 200 terminals the results are...

Table 5.14 Results for the bus topology (method 3) for data sets consisting of 200 terminals.

n = 200	Esau Williams Cost	Simulated Annealing Cost	Savings	Esau Williams Time	Simulated Annealing Time	*Average Improvement %*	*Annealing Schedule (Parameters)*
200_1	8281	7762	519	8 sec	33 min 18 sec	**5.11 %**	*Temperature = 4* *Alpha = .960* *Epsilon = .01* *Repetition = 50* *Alternate Stopping Criteria = 4*
200_2	9004	8586	418	9 sec	47 min 09 sec		
200_3	8778	8321	457	10 sec	38 min 34 sec		
200_4	8910	8518	392	10 sec	38 min 19 sec		

For the four data sets representing the xy-coordinate locations of 250 terminals the results are...

Table 5.15 Results for the bus topology (method 3) for data sets consisting of 250 terminals.

n = 250	Esau Williams Cost	Simulated Annealing Cost	Savings	Esau Williams Time	Simulated Annealing Time	*Average Improvement %*	*Annealing Schedule (Parameters)*
250_1	10143	9734	409	18 sec	30 min 35 sec	**3.79 %**	*Temperature = 2* *Alpha = .992* *Epsilon = .01* *Repetition = 50* *Alternate Stopping Criteria = 1*
250_2	10132	9653	479	19 sec	21 min 18 sec		
250_3	10101	9540	561	19 sec	33 min 33 sec		
250_4	10249	10158	91	20 sec	5 min 39 sec		

5.4 - Results for the Loop Topology (Method 1).

For the loop topology, the results of our simulated annealing algorithm using method 1 (randomly selecting a specified number of terminal pairs at each repetition) are given below.

For the four data sets representing the xy-coordinate locations of 50 terminals the results are...

Table 5.16 Results for the loop topology (method 1) for data sets consisting of 50 terminals.

n = 50	Clarke Wright Cost	Simulated Annealing Cost	Savings	Clarke Wright Time	Simulated Annealing Time	*Average Improvement %*	*Annealing Schedule (Parameters)*
50_1	4851	4131	720	0	36 sec	**10.17 %**	*Temperature = 10* *Alpha = .999* *Epsilon = .01* *Repetition = 22* *Pairs of terminals = 25* *Alternate Stopping Criteria = 250*
50_2	4942	4433	509	0	26 sec		
50_3	4177	4099	78	0	23 sec		
50_4	5053	4425	628	0	55 sec		

For the four data sets representing the xy-coordinate locations of 100 terminals the results are...

Table 5.17 Results for the loop topology (method 1) for data sets consisting of 100 terminals

n = 100	Clarke Wright Cost	Simulated Annealing Cost	Savings	Clarke Wright Time	Simulated Annealing Time	*Average Improvement %*	*Annealing Schedule (Parameters)*
100_1	8776	7549	1227	2 sec	2 min 07 sec	**12.18 %**	*Temperature = 10* *Alpha = .999* *Epsilon = .01* *Repetition = 35* *Pairs of terminals = 35* *Alternate Stopping Criteria = 250*
100_2	7750	6741	1009	1 sec	2 min 11 sec		
100_3	7938	7446	492	2 sec	2 min 07 sec		
100_4	8439	7176	1279	2 sec	2 min 24 sec		

For the four data sets representing the xy-coordinate locations of 150 terminals the results are...

Table 5.18 Results for the loop topology (method 1) for data sets consisting of 150 terminals.

n = 150	Clarke Wright Cost	Simulated Annealing Cost	Savings	Clarke Wright Time	Simulated Annealing Time	*Average Improvement %*	*Annealing Schedule (Parameters)*
150_1	10622	9417	1205	8 sec	2 min 35 sec	**11.57 %**	*Temperature = 10* *Alpha = .999* *Epsilon = .01* *Repetition = 55* *Pairs of terminals = 35* *Alternate Stopping Criteria = 200*
150_2	12414	10751	1663	10 sec	6 min 10 sec		
150_3	10937	10019	918	9 sec	3 min 36 sec		
150_4	11848	10332	1516	8 sec	4 min 03 sec		

For the four data sets representing the xy-coordinate locations of 200 terminals the results are...

Table 5.19 Results for the loop topology (method 1) for data sets consisting of 200 terminals.

n = 200	Clarke Wright Cost	Simulated Annealing Cost	Savings	Clark Wright Time	Simulated Annealing Time	*Average Improvement %*	*Annealing Schedule (Parameters)*
200_1	14335	12755	1580	23 sec	5 min 54 sec	**10.35 %**	*Temperature = 16*
200_2	15425	13455	1970	27 sec	5 min 18 sec		*Alpha = .990*
200_3	15156	13808	1348	28 sec	5 min 59 sec		*Epsilon = .01*
200_4	15468	14116	1352	29 sec	6 min 16 sec		*Repetition = 45*
							Pairs of terminals = 60
							Alternate Stopping Criteria = 120

For the four data sets representing the xy-coordinate locations of 250 terminals the results are...

Table 5.20 Results for the loop topology (method 1) for data sets consisting of 250 terminals.

n = 250	Clarke Wright Cost	Simulated Annealing Cost	Savings	Clarke Wright Time	Simulated Annealing Time	*Average Improvement %*	*Annealing Schedule (Parameters)*
250_1	17746	15496	2250	55 sec	20 min 51 sec	**11.88 %**	*Temperature = 18*
250_2	17344	15210	2134	56 sec	19 min 34 sec		*Alpha = .992*
250_3	17294	15281	2013	53 sec	13 min 49 sec		*Epsilon = .01*
250_4	17984	16038	1946	58 sec	17 min 50 sec		*Repetition = 55*
							Pairs of terminals = 100
							Alternate Stopping Criteria = 110

5.5 - Results for the Loop Topology (Method 2).

For the loop topology, the results of our simulated annealing algorithm using method 2 (randomly selecting two lines and considering all possible pairs of terminals that can be combined from these two at each repetition) are given below.

For the four data sets representing the xy-coordinate locations of 50 terminals the results are...

Table 5.21 Results for the loop topology (method 2) for data sets consisting of 50 terminals.

n = 50	Clarke Wright Cost	Simulated Annealing Cost	Savings	Clarke Wright Time	Simulated Annealing Time	*Average Improvement %*	*Annealing Schedule (Parameters)*
50_1	4851	4164	687	0	31 sec		*Temperature = 14*
50_2	4942	4218	724	0	30 sec		*Alpha = .998*
50_3	4177	4080	97	0	24 sec	**10.27 %**	*Epsilon = .01*
50_4	5053	4607	446	0	19 sec		*Repetition = 75*
							Alternate Stopping Criteria = 100

For the four data sets representing the xy-coordinate locations of 100 terminals the results are...

Table 5.22 Results for the loop topology (method 2) for data sets consisting of 100 terminals.

n = 100	Clarke Wright Cost	Simulated Annealing Cost	Savings	Clarke Wright Time	Simulated Annealing Time	*Average Improvement %*	*Annealing Schedule (Parameters)*
100_1	8776	7533	1243	2 sec	48 sec		*Temperature = 22*
100_2	7750	7362	388	2 sec	17 sec	**12.25 %**	*Alpha = .998* *Epsilon = .01*
100_3	7938	6331	1607	2 sec	44 sec		*Repetition = 55*
100_4	8439	7646	793	2 sec	34 sec		*Alternate Stopping Criteria = 100*

For the four data sets representing the xy-coordinate locations of 150 terminals the results are...

Table 5.23 Results for the loop topology (method 2) for data sets consisting of 150 terminals.

n = 150	Clarke Wright Cost	Simulated Annealing Cost	Savings	Clarke Wright Time	Simulated Annealing Time	*Average Improvement %*	*Annealing Schedule (Parameters)*
150_1	10622	8948	1674	8 sec	1 min 24 sec		*Temperature = 12*
150_2	12414	11165	1249	10 sec	35 sec	**13.09 %**	*Alpha = .998* *Epsilon = .01*
150_3	10937	9887	1050	9 sec	1 min 22 sec		*Repetition = 55*
150_4	11848	9819	2029	8 sec	1 min 30 sec		*Alternate Stopping Criteria = 150*

For the four data sets representing the xy-coordinate locations of 200 terminals the results are...

Table 5.24 Results for the loop topology (method 2) for data sets consisting of 200 terminals.

n = 200	Clarke Wright Cost	Simulated Annealing Cost	Savings	Clark Wright Time	Simulated Annealing Time	*Average Improvement %*	*Annealing Schedule (Parameters)*
200_1	14335	12235	2100	23 sec	4 min 55 sec		*Temperature = 25*
200_2	15425	13311	2114	26 sec	1 min 08 sec	**13.51 %**	*Alpha = .998*
200_3	15156	12912	2244	28 sec	4 min 24 sec		*Epsilon = .01*
200_4	15468	13771	1697	29 sec	1 min 53 sec		*Repetition = 75*
							Alternate Stopping Criteria = 200

For the four data sets representing the xy-coordinate locations of 250 terminals the results are...

Table 5.25 Results for the loop topology (method 2) for data sets consisting of 250 terminals.

n = 250	Clarke Wright Cost	Simulated Annealing Cost	Savings	Clarke Wright Time	Simulated Annealing Time	*Average Improvement %*	*Annealing Schedule (Parameters)*
250_1	17746	15518	2228	55 sec	1 min 29 sec		*Temperature = 10*
250_2	17344	14713	2631	56 sec	1 min 51 sec	**12.50 %**	*Alpha = .999*
250_3	17294	15136	2158	53 sec	1 min 38 sec		*Epsilon = .01*
250_4	17984	16204	1780	58 sec	3 min 04 sec		*Repetition = 35*
							Alternate Stopping Criteria = 250

5.6 - Results for the Loop Topology (Method 3).

For the loop topology, the results of our simulated annealing algorithm using method 3 (a systematic approach of considering all possible pairs of terminals to create potential neighbour solutions) are given below.

For the four data sets representing the xy-coordinate locations of 50 terminals the results are...

Table 5.26 Results for the loop topology (method 3) for data sets consisting of 50 terminals.

n = 50	Clarke Wright Cost	Simulated Annealing Cost	Savings	Clarke Wright Time	Simulated Annealing Time	*Average Improvement %*	*Annealing Schedule (Parameters)*
50_1	4851	4491	360	0	1 min 08 sec	**7.91 %**	*Temperature = 4*
50_2	4942	4503	439	0	4 min 11 sec		*Alpha = .999*
50_3	4177	4091	86	0	1 min 10 sec		*Epsilon = .01*
50_4	5053	4433	620	0	1 min 43 sec		*Repetition = 70*
							Alternate Stopping Criteria = 4

For the four data sets representing the xy-coordinate locations of 100 terminals the results are...

Table 5.27 Results for the loop topology (method 3) for data sets consisting of 100 terminals.

n = 100	Clarke Wright Cost	Simulated Annealing Cost	Savings	Clarke Wright Time	Simulated Annealing Time	*Average Improvement %*	*Annealing Schedule (Parameters)*
100_1	8776	7809	967	2 sec	9 min 35 sec	**7.92 %**	*Temperature = 3* *Alpha = .999* *Epsilon = .01* *Repetition = 70* *Alternate Stopping Criteria = 6*
100_2	7750	7288	462	2 sec	6 min 41 sec		
100_3	7938	7416	522	2 sec	5 min 58 sec		
100_4	8439	7783	656	2 sec	18 min 04 sec		

For the four data sets representing the xy-coordinate locations of 150 terminals the results are...

Table 5.28 Results for the loop topology (method 3) for data sets consisting of 150 terminals.

n = 150	Clarke Wright Cost	Simulated Annealing Cost	Savings	Clarke Wright Time	Simulated Annealing Time	*Average Improvement %*	*Annealing Schedule (Parameters)*
150_1	10622	9393	1229	9 sec	21 min 40 sec	**7.15 %**	*Temperature = 3* *Alpha = .999* *Epsilon = .01* *Repetition = 75* *Alternate Stopping Criteria = 5*
150_2	12414	11556	858	10 sec	23 min 30 sec		
150_3	10937	10344	593	9 sec	18 min 58 sec		
150_4	11848	11251	597	8 sec	19 min 17 sec		

For the four data sets representing the xy-coordinate locations of 200 terminals the results are...

Table 5.29 Results for the loop topology (method 3) for data sets consisting of 200 terminals.

n = 200	Clarke Wright Cost	Simulated Annealing Cost	Savings	Clark Wright Time	Simulated Annealing Time	*Average Improvement %*	*Annealing Schedul (Parameters)*
200_1	14335	13273	1062	23 sec	15 min 49 sec		*Temperature = 1*
200_2	15425	13754	1671	27 sec	22 min 18 sec	**7.14 %**	*Alpha = .960* *Epsilon = .01*
200_3	15156	14350	806	28 sec	24 min 28 sec		*Repetition = 25*
200_4	15468	14693	775	30 sec	16 min 33 sec		*Alternate Stopping Criteria = 10*

For the four data sets representing the xy-coordinate locations of 250 terminals the results are...

Table 5.30 Results for the loop topology (method 3) for data sets consisting of 250 terminals.

n = 250	Clarke Wright Cost	Simulated Annealing Cost	Savings	Clarke Wright Time	Simulated Annealing Time	*Average Improvement %*	*Annealing Schedule (Parameters)*
250_1	17746	16234	1512	55 sec	40 min 36 sec		*Temperature = 1*
250_2	17344	16279	1165	57 sec	28 min 17 sec	**8.83 %**	*Alpha = .950* *Epsilon = .01*
250_3	17294	15729	1565	54 sec	43 min 01 sec		*Repetition = 25*
250_4	17984	16026	1958	59 sec	38 min 28 sec		*Alternate Stopping Criteria = 10*

5.7 - Results for the Tree Topology.

For the tree topology, the results of our simulated annealing algorithm are given below. Recall that for the tree, the SA algorithm consists in altering the order in which the Esau-Williams algorithm selects its links.

For the four data sets representing the xy-coordinate locations of 50 terminals the results are...

Table 5.31 Results for the tree topology for data sets consisting of 50 terminals.

n = 50	Esau Williams Cost	Simulated Annealing Cost	Savings	Esau Williams Time	Simulated Annealing Time	*Average Improvement %*	*Annealing Schedule (Parameters)*
50_1	3021	2988	33	0	1 min 09 sec	**1.11 %**	*Temperature = 3*
50_2	2910	2907	3	0	1 min 10 sec		*Alpha = .900*
50_3	2701	2640	61	0	56 sec		*Epsilon = .99*
50_4	3221	3186	35	0	1 min 11 sec		*Repetition = 35*

For the four data sets representing the xy-coordinate locations of 100 terminals the results are...

Table 5.32 Results for the tree topology for data sets consisting of 100 terminals.

n = 100	Esau Williams Cost	Simulated Annealing Cost	Savings	Esau Williams Time	Simulated Annealing Time	*Average Improvement %*	*Annealing Schedule (Parameters)*
100_1	5236	5190	60	2 sec	13 min 02 sec		*Temperature = 10*
100_2	4789	4726	84	1 sec	11 min	**1.45 %**	*Alpha = .880*
100_3	4812	4756	50	2 sec	11 min 47 sec		*Epsilon = .99*
100_4	5095	4971	124	2 sec	12 min 23 sec		*Repetition = 20*

For the four data sets representing the xy-coordinate locations of 150 terminals the results are...

Table 5.33 Results for the tree topology for data sets consisting of 150 terminals.

n = 150	Esau Williams Cost	Simulated Annealing Cost	Savings	Esau Williams Time	Simulated Annealing Time	*Average Improvement %*	*Annealing Schedule (Parameters)*
150_1	6291	6231	60	8 sec	22 min 15 sec		*Temperature = 7*
150_2	7347	7263	84	12 sec	30 min 48 sec	**1.37 %**	*Alpha = .870*
150_3	6572	6522	50	9 sec	24 min 46 sec		*Epsilon = .50*
150_4	6952	6775	177	9 sec	23 min 11 sec		*Repetition = 9*

For the four data sets representing the xy-coordinate locations of 200 terminals the results are...

Table 5.34 Results for the tree topology for data sets consisting of 200 terminals.

n = 200	Esau Williams Cost	Simulated Annealing Cost	Savings	Esau Williams Time	Simulated Annealing Time	*Average Improvement %*	*Annealing Schedule (Parameters)*
200_1	8235	8169	66	26 sec	1 hr 56 min 30 sec		*Temperature = 6*
200_2	8805	8730	75	32 sec	2 hrs 21 min 32 sec	**1.05 %**	*Alpha = .900* *Epsilon = .50*
200_3	8628	8528	100	32 sec	2 hrs 24 min 15 sec		*Repetition = 12*
200_4	8747	8628	119	34 sec	2 hrs 32 min 16 sec		

For the four data sets representing the xy-coordinate locations of 250 terminals the results are...

Table 5.35 Results for the tree topology for data sets consisting of 250 terminals.

n = 250	Esau Williams Cost	Simulated Annealing Cost	Savings	Esau Williams Time	Simulated Annealing Time	*Average Improvement %*	*Annealing Schedule (Parameters)*
250_1	10006	9926	80	1 min 05 sec	1 hr 15 min 23 sec		*Temperature = 18*
250_2	9985	9895	90	1 min 05 sec	1 hr 16 min 15 sec	**0.69 %**	*Alpha = .700* *Epsilon = .50*
250_3	10039	9943	96	1 min 05 sec	1 hr 16 min 50 sec		*Repetition = 3*
250_4	10132	10122	10	1 min 08 sec	1 hr 20 min 21 sec		

5.8 - Overall Average Improvement on the Initial Solutions.

The following table shows the overall average improvement obtained by each program considering the results from all 20 data sets.

Table 5.36 Average improvement for each heuristic.

	Method 1	Method 2	Method 3
Bus[1]	7.06 %	8.69 %	4.77 %
Loop[2]	11.23 %	12.33 %	7.79 %
Tree[3]	1.13 %		

5.9 - Analysis of Results.

Before testing began, intuitively at least, we believed that for the bus and loop topologies, method 3 would yield better results. This was due to the fact that method 3 was presumed to have a distinct advantage by systematically considering all possible neighbour solutions before selecting one among them. Furthermore, the higher running times of method 3 when compared to the other two methods were not unexpected, but it

[1] For the Bus topology the improvement is over the Esau-Williams solution.
[2] For the Loop topology the improvement is over the Clarke-Wright solution.
[3] For the Tree topology the improvement is over the Esau-Williams solution.

was presumed that the additional savings that the method would offer would offset this drawback. Yet, as the results clearly show, for the 20 data sets tested and the annealing schedules that were selected, method 3 offers the least improvement among all three methods, and this for both topologies. In addition, if running time is also taken into consideration, this method is clearly disadvantageous. From the table above, we can see that for both the bus and loop topologies, method 2 offers the most savings. In addition, for the loop topology under the SA process using method 2, the SA solution offered a 20.24% improvement over the Clarke-Wright algorithm for data set '100_3'(see table 22, p.112). Surprisingly, for both topologies, it is also method 2 that offers significantly smaller running times when compared to the other two methods.

For the tree topology, although an improvement over the initial solution was obtained, the computational results are disappointing. Not only is the running time quite high when compared to our other SA programs for the same data sets, the overall average cost improvement is only of 1.13 % over the Esau-Williams solution.

A possible explanation for the relatively small improvements obtained for the tree topology is that good annealing schedules for the data sets that were tested were not found. Another possibility for the lower improvement percentages is that we are somewhat erroneously evaluating the quality of the tree solutions. Indeed, by comparing them to the greater improvements obtained for the bus and loop topologies, the improvements for the tree topology may do appear disproportionate to those obtained for the other two topologies. However, the relatively small amount of improvement for the tree topology may simply be due to the high quality of the initial solution itself. In fact, in the tree topology, a multipoint line is made up of terminals that were interconnected in a

less restrictive manner when compared to the other two topologies. In other words, the Esau-Williams solutions obtained for our tree topology problems may be relatively close to exact solutions, in such a case, large improvements should not be expected.

For the bus and loop topologies, the addition of an alternate stopping criterion proved to be beneficial in reducing the running times of the programs. Recall that this alternate criterion stops the SA process if a specified number of consecutive temperature decreases occur without procuring any savings. The selection of a value for this criterion depends mainly on the value of alpha. We noticed that when alpha was set to a value very close to 1, say, 0.999, the value for this alternate criterion needed also to be set quite high since at high alpha settings the temperature decrements very slowly. In other words, this has the effect of increasing the total number of temperature decrements in the annealing process. As this number increases so too does the average number of temperature decrements between the occurrences of lower cost neighbour solutions.

Another observation that was made was the importance in finding a good value for the initial temperature. Testing showed that lower temperature settings, generally less than 20, provided the best results. This, we believe, is due to the fact that the initial solutions provided for our SA programs are already quite good. High initial temperature settings deteriorate these initial solutions greatly from the start of the SA process. In such situations, as the temperature approaches zero the programs probably spend most of their time fixing their previous cost increasing alterations.

5.10 - Limitations of this Research.

The major limitation of our research is that although improvements were obtained over well-known heuristic solutions (Clarke-Wright and Esau-Williams algorithms) that are used as input to our programs, we have not been able to evaluate how close our results come to the exact solution for lack of a lower bound on the optimum. Furthermore, we did not grade the quality of the solutions obtained by our SA algorithms relative to other existing CMST and VRP heuristic solutions apart from those used to obtain our initial solutions.

Moreover, only one value of *Wmax* (the maximum amount of traffic a line can effectively carry) is considered. This is analogous to designing a network with only one type of data transmission medium (only copper wire for example). In reality, a multitude of data transmission media exist as described in chapter 1. However, in this aspect we are not alone; none of the previous CMST research that was collected and reviewed in this thesis took this factor into account. In the real world, we believe that any heuristic solution (as implemented in existing research on the terminal layout problem) should probably only reflect an upper bound on the true cost of a real centralized network. This is because an appropriate value for *Wmax* must be specified so that any line can at least accept the terminal with the largest amount of traffic ($Wmax \geq w_i$ for all i). Once a solution is obtained with this value of *Wmax*, any multipoint line that has slack *could* be down-graded to a lower gauge line that would still accept all the terminals already on that multipoint line. Since the cost of data transmission media is largely dependent on its transmission capability or what researchers call *Wmax* (also generically known as the

'speed' of a line), the change to a lower gauged line could provide additional savings. Considering that the classical formulation of the CMST problem is already very difficult to solve to optimality in a reasonable amount of time, the additional consideration of multiple media types would complicate the problem greatly. For the loop topology for example, considering multiple *Wmax* values would be synonymous to considering a vehicle routing problem with a fleet of transport vehicles having different payload capacities.

CHAPTER 6 - CONCLUSION.

6.1 - Conclusion.

We developed programs using simulated annealing in an attempt to improve upon the quality of the solutions obtained from well-known heuristics used for the terminal layout problem. Given the geographical locations of the terminals assigned to a concentrator, the problem consists in creating multipoint lines in order to minimize the cost of centralized computer networks. Specifically, instead of independently connecting terminals to a concentrator, where each terminal monopolizes one line, cost reductions are obtained by allowing several terminals to share the same data transmission line. Three types of multipoint line topologies were considered; the tree, the bus, and the loop.

This research has shown that simulated annealing is a very flexible tool when applied to the terminal layout problem. We demonstrate that SA yields improvement on solutions generated by the Esau-Williams and Clarke-Wright algorithms for all the data sets we tested. In fact, in some cases, considerably better solutions were obtained. However, for some of our programs and with data sets consisting of a large number of terminals, typically above 150, the CPU time was quite high when compared to the time required to calculate the initial solutions.

The results disclosed in this thesis were obtained by selecting the best solutions obtained through a process of trial_and_error by varying each program's annealing schedule. Although some of our algorithms exhibited better quality solutions than others,

we are certain that additional computational experiments using various annealing schedules could yield even better results in all our SA algorithms.

6.2 - Recommendations.

Further research into the CMST should include the development of other effective solution procedures. To our knowledge, other techniques such as tabu search and genetic algorithms have not been applied to the CMST. Moreover, an in depth investigation of the full potential of the application of SA to this problem is needed for it has shown itself to be well adapted to this type of problem. In addition, a need exists for establishing a standardized set of test problems for the CMST. The lack of such data sets make comparisons of new CMST heuristics difficult.

Bibliography:

Bohachevsky, I.O., Johnson, M.E. and Stein, M.L: "Generalized Simulated Annealing for Function Optimization", *Technometrics*, 28, 1986, p. 209-217.

Boorstyn, R.R., and Frank, H.: "Large Scale Network Topological Optimization", *IEEE Trans. Comm.*, Com-25, 1977, p. 29- 47.

Breedam, A.V. : "Improvement Heuristics for the Vehicle Routing Problem Based on Simulated Annealing", *European Journal of Operational Research*, 86, 1995, p 480- 490.

Cerny, V : "Thermodynamical Approach to the Travelling Salesman Problem : An Efficient Simulation Algorithm", *Journal of Optimization Theory and Applications*, 45, 1985, p. 41-51.

Chandy, K.M., and Lo, T. : "The Capacitated Minimum Spanning Tree", *Networks*, Vol. 3, no.2, 1973, p. 173- 182.

Clarke, G. and Wright, J.: "Scheduling for Vehicles from a Central Depot to a Number of Delivery Points", *Operations Research*, no. 12, 1964, p. 568- 581.

Eglese, R.W : "Simulated Annealing: A Tool for Operational Research", *European Journal of Operational Research*, 46, 1990, p. 271-281.

Elias, D., and Ferguson, M.J. : "Topological Deesign of Multipoint Teleprocessing Networks", *IEEE Transactions Comm.*, COM- 22, 1974, p. 1753- 1762.

Esau, L.R., and Williams, K.C. : "On Teleprocessing System Design", Part II, *IBM System Journal*, Vol. 5, no. 3, 1966, p. 142-147.

Garey, M.R., and Johnson, D.S. : *Computers and Intractability - A Guide to the Theory of NP-Completeness*, Freeman, San Fransisco, 1979.

Gavish. B. : "Topological Design of Centralized Computer Networks - Formulations and Algorithms", *Networks*, Vol. 12, 1982, p. 355- 377.

Gavish. B. : "Formulations and Algorithms for the Capacitated Minimal Directed Spanning Tree", *J. ACM*, Vol. 30, 1983, p. 118- 132.

Gavish. B. : "Augmented Lagrangian Based Algorithms for Centralized Network Design", *IEEE Transactions on Comm.*, Vol. Com-33, no. 12, 1985, p. 1247- 1257.

Gavish. B. : "Topological Design of Telecommunication Networks - Local Access Design Methods", *Annals of Operations Research*, Vol. 33, 1991, p. 17- 71.

Gondran, M. and Minoux, M. : *Graphs and Algorithms*, Chichester, West Sussex, NY, Wiley, 1984.

Greene, J.W., and Supowit, K.J. : "Simulated Anneealing Without Rejected Moves", *IEEE Transactions on Computer-Aided Design*, 5, 1986, p. 221- 228.

Gouveia, L.: "A 2nd Constraint Formulation for the Capacitated Minimal Spanning Tree Problem", *Operations Research*, Vol. 43, no. 1, 1995, p. 130- 141.

Karnaugh, M. : "A New Class of Algorithms for Designing Multipoint Network Optimization", *IEEE Transactions Comm.*, COM-24, 1976, p. 500- 505.

Kershenbaum, A.: "Computing Capacitated Minimal Spanning Trees Efficiently", *Networks*, Vol.4, 1974, p. 299-310.

Kershenbaum, A., and Boorstyn. R.R. : "Centralized Teleprocessing Network Design", *Networks*, Vol. 13, 1983, p. 279- 293.

Kershenbaum, A., and Chou, W.: "A Unified Algorithm for Designing Multidrop Teleprocessing Networks", *IEEE Transactions on Communications*, com-22, no. 11, 1974, p.1762-1771.

Kershenbaum, A., Boorstyn. R.R., and Oppenheim, R. : "Second Order Greedy Algorithms for Centralized Teleprocessing Network Design", *IEEE Transactions Comm.*, COM- 28, 1980, p. 1835- 1838.

Kershenbaum, A.: *Telecommunications Network Design Algorithms*, McGraw-Hill, NY, 1993.

Kirkpatrick, S., Gelatt, Jr., C.D., and Vecchi, M.P.: "Optimization by Simulated Annealing", *Science*, 220, 1983, p. 671-680.

Kruskal, J.B. : "On the Shorteest Spanning Subtree of a Graph and the Travelling Salesman Problem", *Proc. of American Math. Society*, Vol. 7, 1956, p. 48- 50..

Malik, K. and Yu, G.: "A Branch and Bound Algorithm for the Capacitated Minimum Spanning Tree Problem", *Networks*, 23, 1993, p. 525-532.

McGregor, P.M., and Shen, D. : "Network Design : An Algorithm for Access Facility Location Problems", *IEEE Transactions Comm.*, COM- 25, 1977, p. 61- 73.

Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., and Teller, A.H. : "Equation of State Calculations by Fast Computing Machines", *J. Chem. Phys.*, Vol. 21, no. 6, 1953, p.1087- 1092.

Papadimitriou, C.H. : "The Complexity of the Capacitated Tree Problem", *Networks*, Vol. 8, 1978, p. 217- 230.

Papadimitriou, C.H., and Steiglitz, K: *Combinatorial Optimization:Algorithms and Complexity*, Prentice-Hall, inc., Englewood Cliffs, New Jersey, 1982.

Prim, R.C. : "Shortest Connection Networks and Some Generalizations", *Bell System Technical Journal*, Vol. 36, 1957, p. 1389- 1401.

Reinfield, N.V. and Vogel, W.R.: *Mathematical Programming*, Prentice-Hall, Englewood Cliffs, NJ, 1958.

Sharma, R.L., and El-Bardai, M.T. : "Suboptimal Communications Network Synthesis", *Proc. International Conference on Communications*, p. 19.11- 19.16, June 1970.

Stamper, D.A. : *Business Data Communications*, 3rd Edition, The Benjamin/ Cummings Publishing Company, Inc., Redwood City, CA, 1991.

Vidal, René : *Applied Simulated Annealing*, Springer-Verlag Berlin Heidelberg, 1993.

APPENDIX

INPUT AND OUTPUT CAPABILITIES OF THE PROGRAMS.

The programs were implemented for the Windows 3.1™ environment. With the aid of Delphi 1.0™ and after a fairly simple transformation of the original Pascal code, it became possible to design programs that would offer users the ability to easily load data sets of up to 256 terminals into the programs and offer users graphical capabilities in order to display the solutions obtained. This graphical output capability was especially desired so that the results could be verified visually. It also provides the user with a quick way to visually discern the differences between the initial solution and that provided by the SA algorithm.

Inputting data can be done in a variety of ways. Typically the programs allow either data files containing xy-coordinates or symmetric cost matrices to be loaded. In addition, it is possible for the user to create a random set of coordinate values representing the positions of up to 256 terminals relative to a concentrator positioned at point (0,0). It must be noted that only the data sets consisting of coordinate values (either from file or randomly generated) can be used to display the topology in graphical format. In other words, loading data that is represented in a cost matrix does not allow graphical display of the results, although text output is still available .

In the following pages the capabilities of the programs will be introduced and explained. When possible an illustration of the actual window used by the program will be given to familiarize the reader during the explanation. Since all the programs have the same format the following applies to all although we have chosen to review the program implemented for the bus topology using method 2.

To begin, when the program is first loaded the following screen appears.

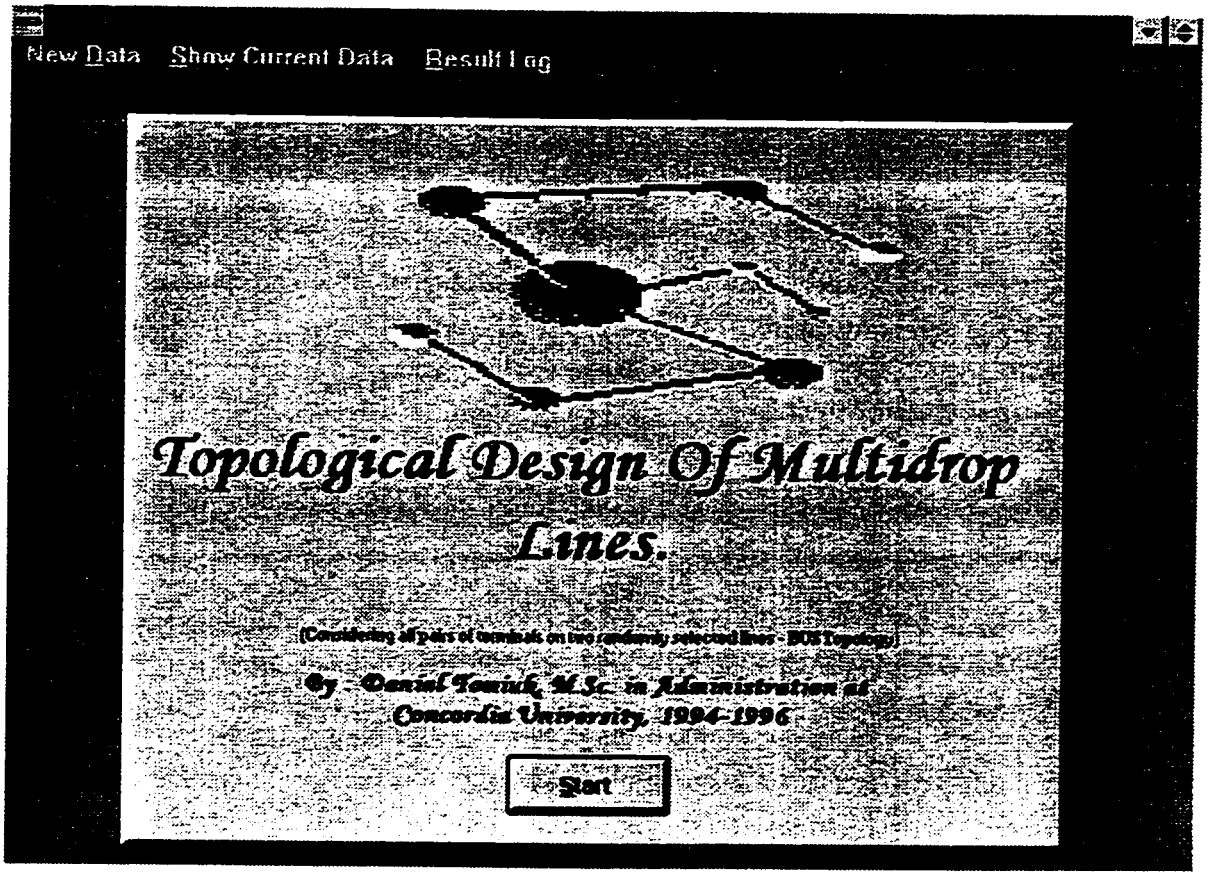


figure A1. Example of the initial window.

The **menu** contains the following items '**New Data**', '**Show Current Data**', '**Result Log**', '**About**', and '**Quit**' although at this point only the last two items of the menu are enabled. Once the user clicks on the start button, the introduction screen disappears and the '**New Data**' menu item is enabled.

By clicking on this menu item the user is capable of loading a data set into the program. As mentioned above, data sets may be randomly generated by the program. If this is the case, the following screen allows the user to specify the size of the data set and

let the program assign random weight values ranging from 1 to the specified maximum weight as shown below.

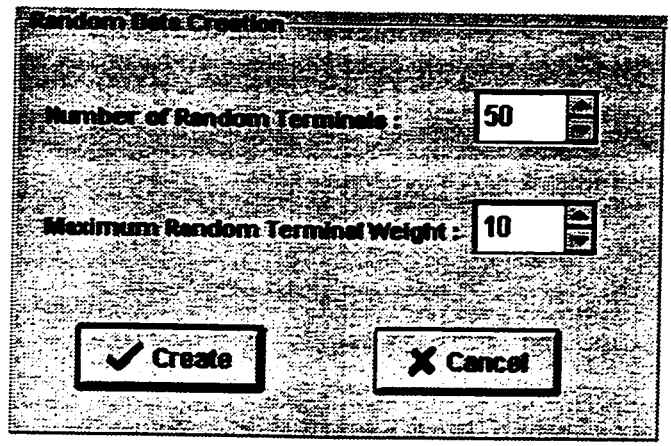


figure A2 Creating a random data set of 50 terminals.

Instead, if the user decides to load an existing data set the following screen allows him/her to do so.

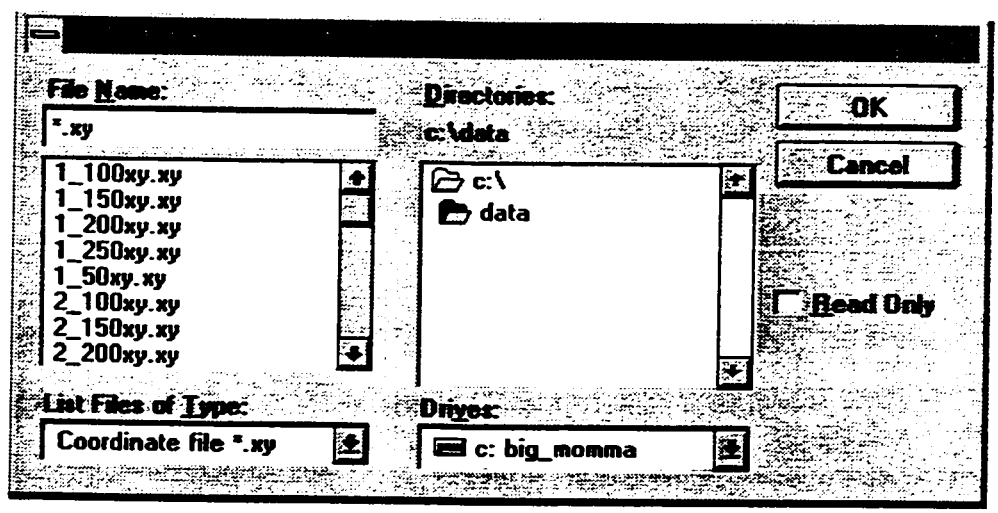


figure A3 Loading an existing data file.

Once a data set has been selected and loaded into the program, a window appears allowing the user to enter the parameters needed to calculate the initial solution and those required by the SA process.

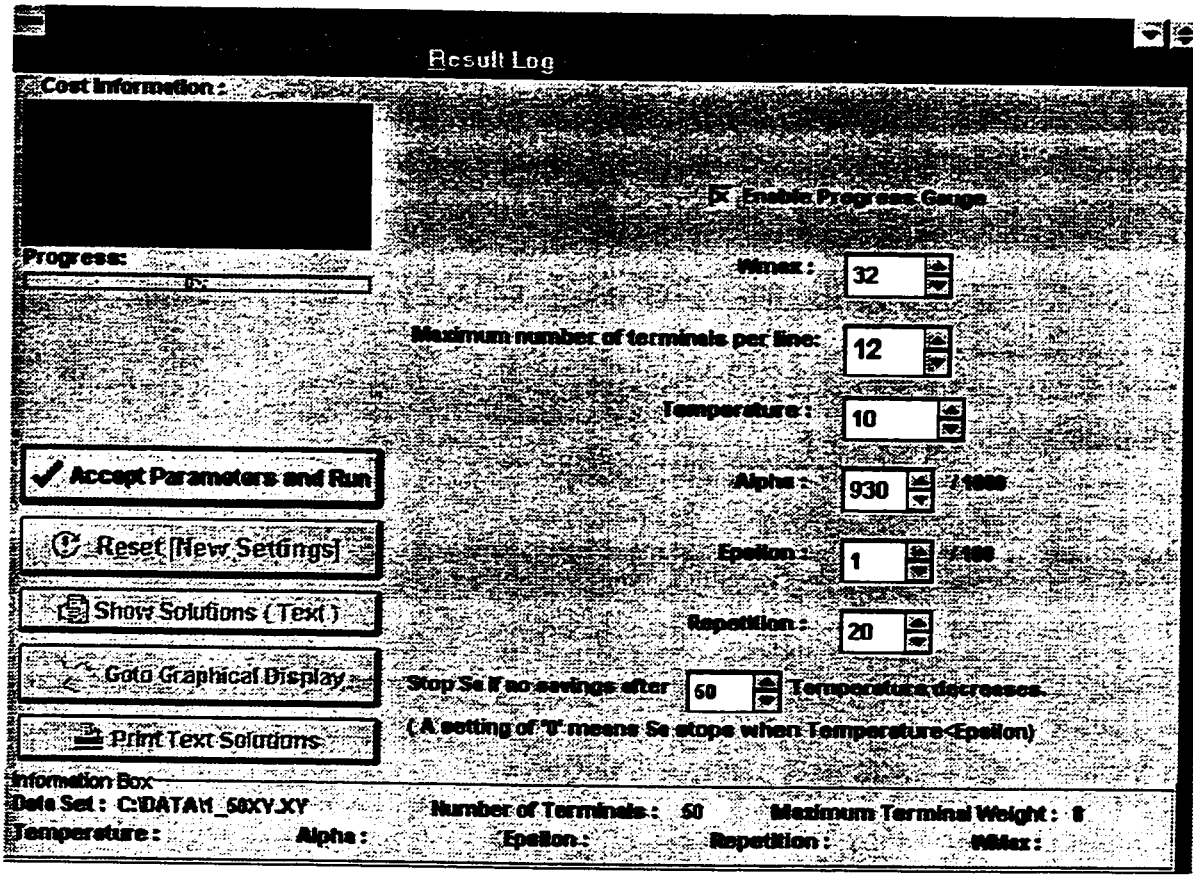


figure A4 Program parameters.

Note that the menu item '**Show Current Data**' is now enabled. By clicking this item, the user can examine the xy-coordinate values and the weight assigned to each terminal of the data set that is currently loaded (not shown).

The screen shown above allows the user to manipulate the parameters of both the topological problem itself ('*Wmax*' and the '*maximum number of terminals per line*') and the SA parameters (annealing schedule). The cost information box situated at the top left

of the screen displays the cost of the different solutions and the savings obtained. The progress gauge allows the user to visualize the speed at which the simulated annealing process is working. The information box found at the bottom of the screen displays a summary of the data set currently loaded, the constraint settings, and the annealing schedule. Once an annealing schedule has been specified, the user clicks on the Accept Settings and Run button to start the simulated annealing process.

When the user is satisfied with his/her parameter value assignments, he/she drags the mouse and clicks on the '**Accept Parameters and Run**' button. The program calculates the initial solution, and begins the simulated annealing process. If the progress bar was enabled, the user is given a visual indication of how the program is proceeding. When the results are obtained, the cost information box displays a summary of the findings while the text and graphical output buttons are enabled as shown below.

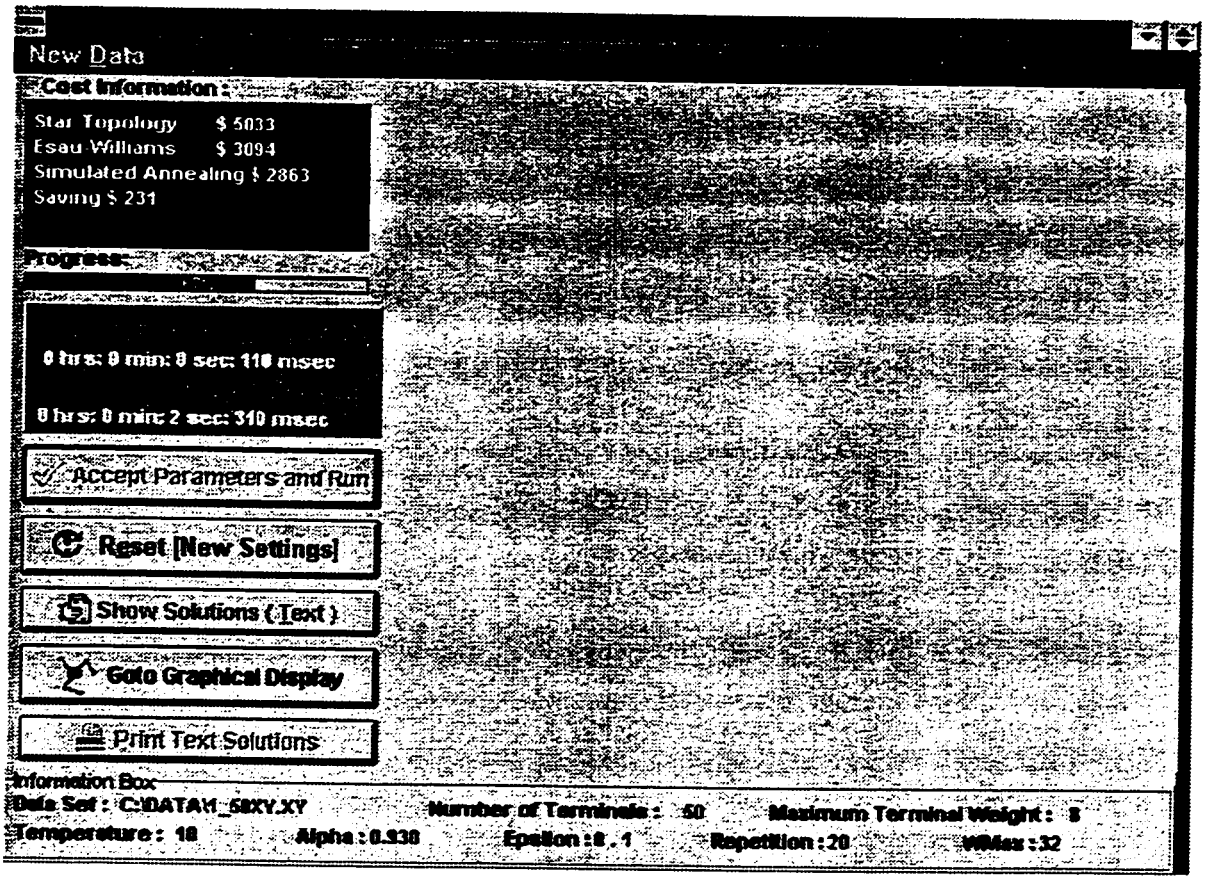


figure A5. An example of the summary of results.

Once the process terminates and a lower cost topological configuration is found, the user may display the textual result as shown below. This text output displays what terminals can be found on what line of the topology. The value in parentheses represents the connection cost between the terminals on the line. Additionally, these text solutions may be printed.

Results given in text format can be seen below.

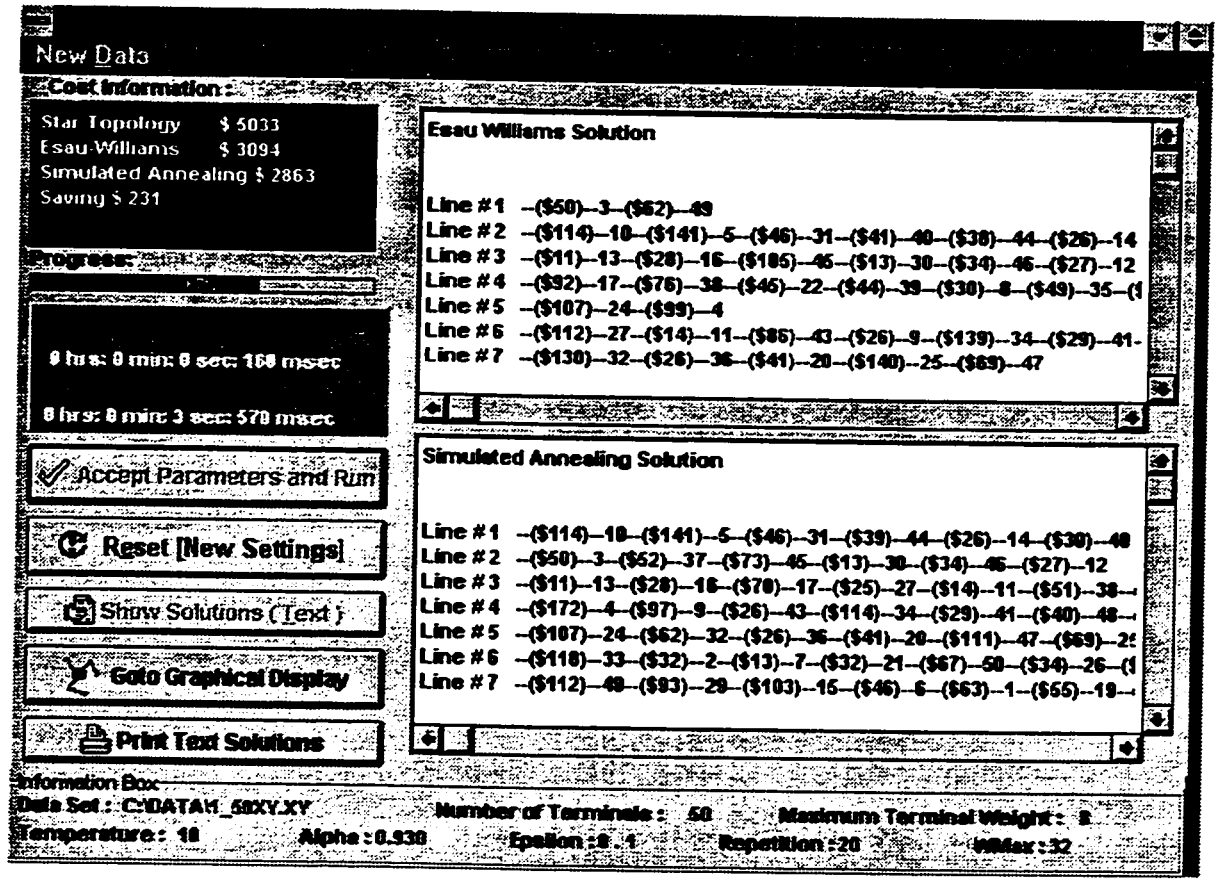


figure A6. An example of text results.

Note that the SA process has finished and a cost reduction obtained but that the progress gauge only registers 67% completion. This is because a value of 50 was specified for the alternative stopping criterion. Recall that an alternative stopping criteria value of 50 informs the program to stop execution if no saving has been obtained for 50 consecutive temperature decreases.

Scroll bars enable the user to view the entire solution if it doesn't fit into the provided solution display area.

For the loop and bus topologies, text results are displayed line by line where each represents the terminals found on a data communication line in the resulting topology. In parentheses we find the costs of the links between the terminals of the line. Finally at the bottom of each text solution, the total sum of the topology is given (as shown below).

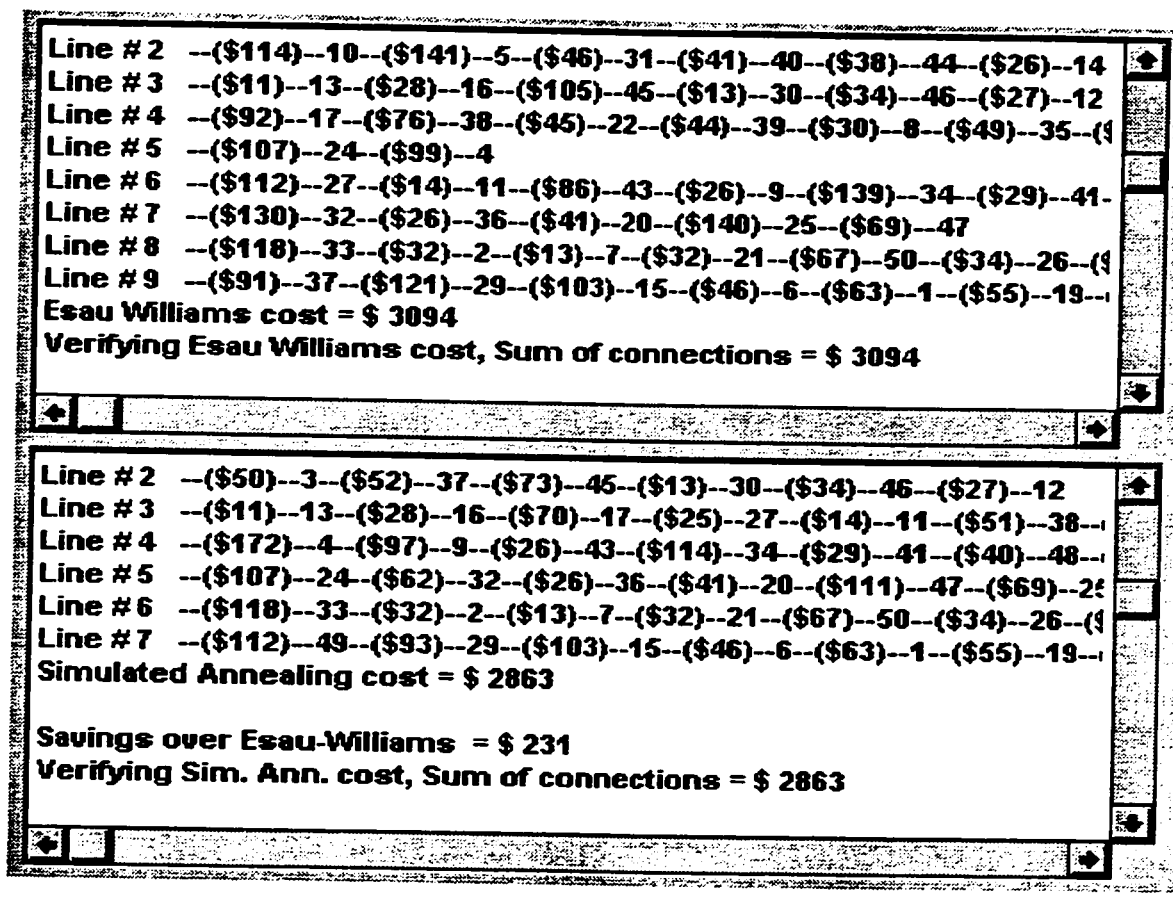


figure A7. Format of text results for the loop and bus topologies.

For the tree topology, an example of text result format is given as follows.

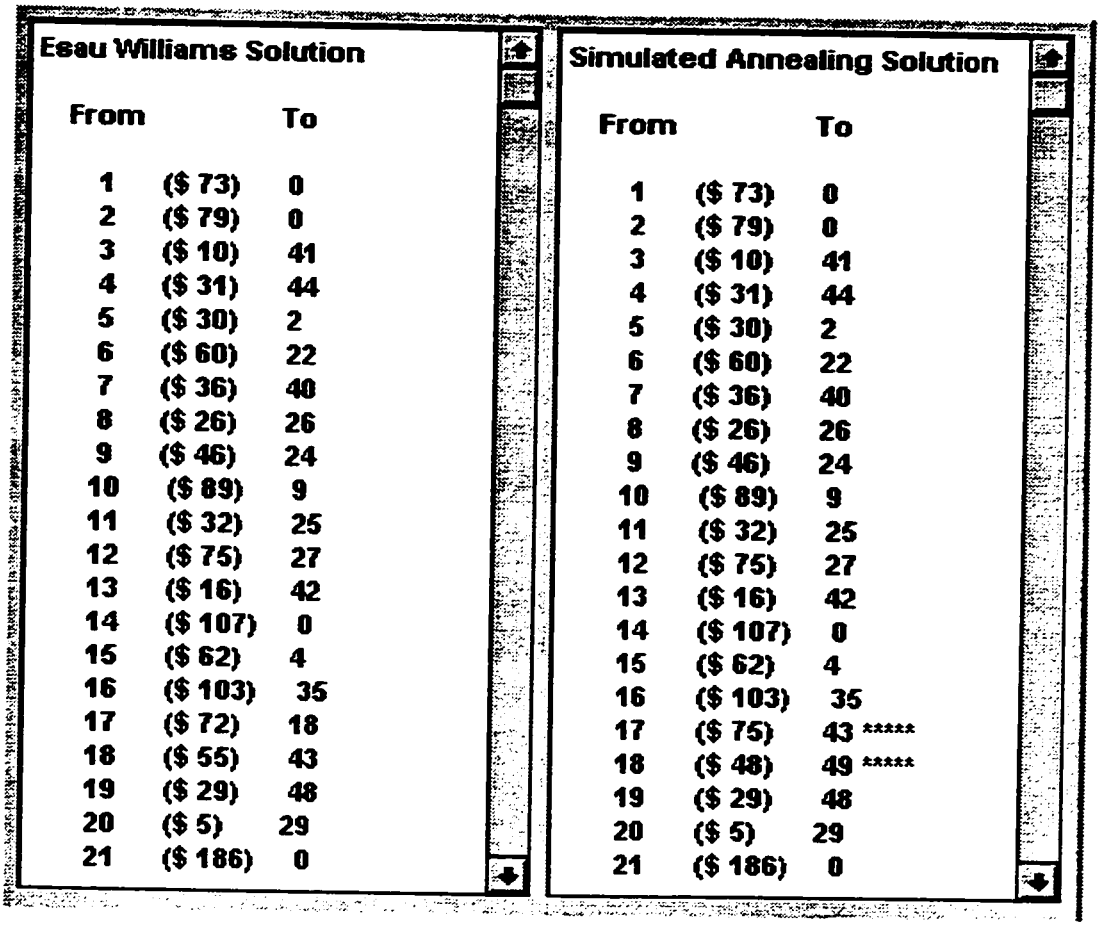


figure A8. Format of text results for the tree topology.

Note that the asterisks beside a terminal in the simulated annealing solution denotes a difference from the Esau-Williams solution.

The graphical display capabilities for the results are illustrated below. The larger node found initially at the centre of the display represents the concentrator. The links interconnecting the terminals represent the communication lines of the network.

For the Esau-Williams solution obtained for the bus topology above, the graphical solution is given as follows.

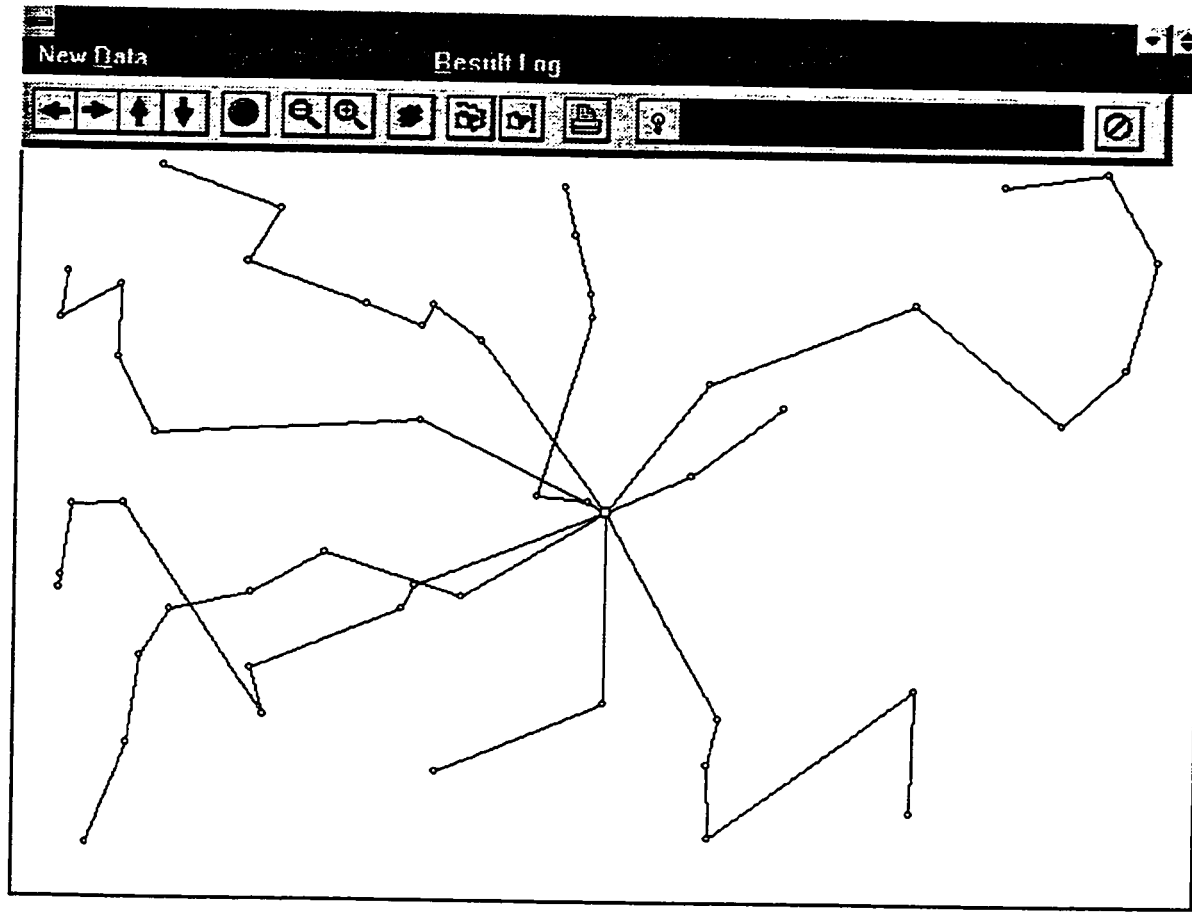


figure A9. Example of a graphical result for the bus topology showing the Esau-Williams solution.

Similarly, the simulated annealing solution can be displayed in the same manner.

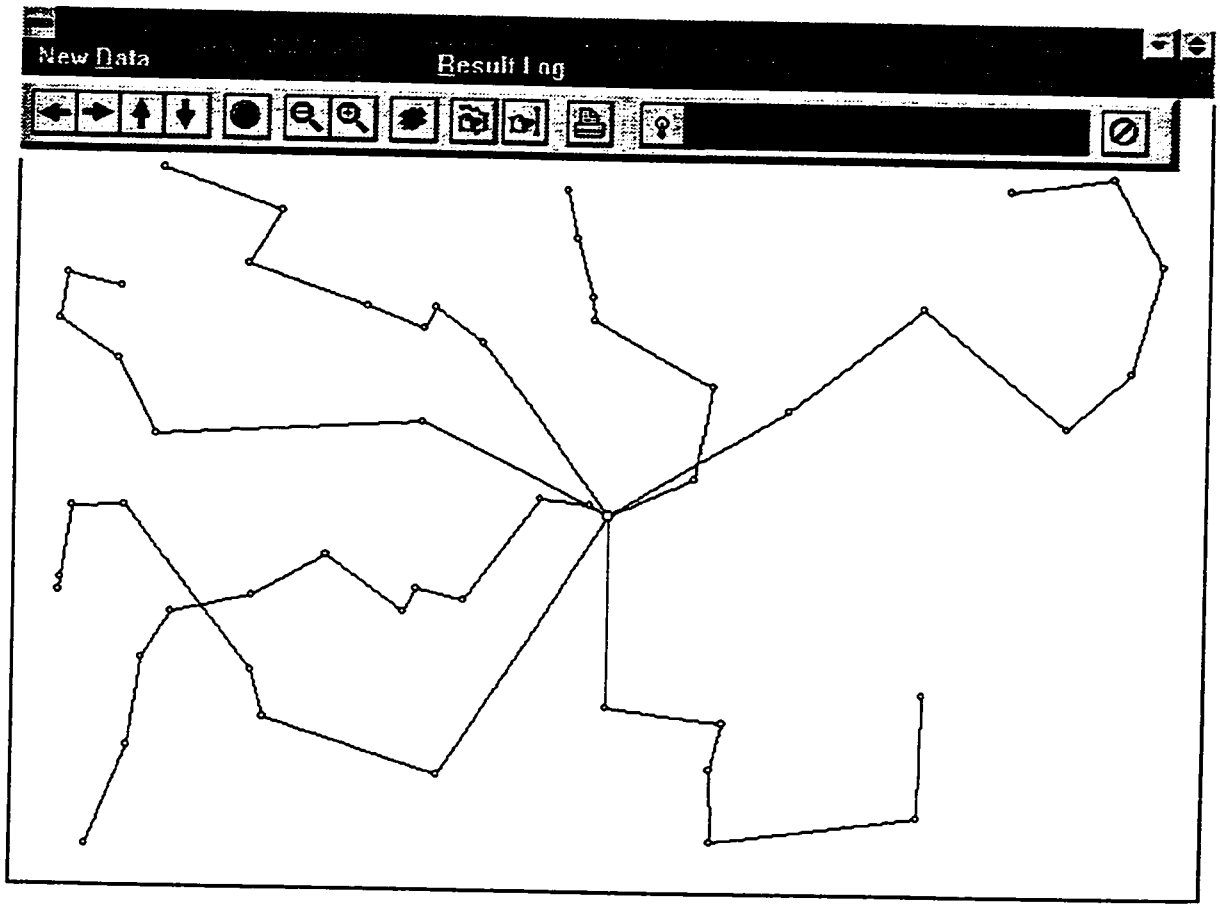


figure A10. Example of a graphical result for the bus topology showing the SA improvement.

In addition, both solutions can be superimposed to note the differences as shown below.

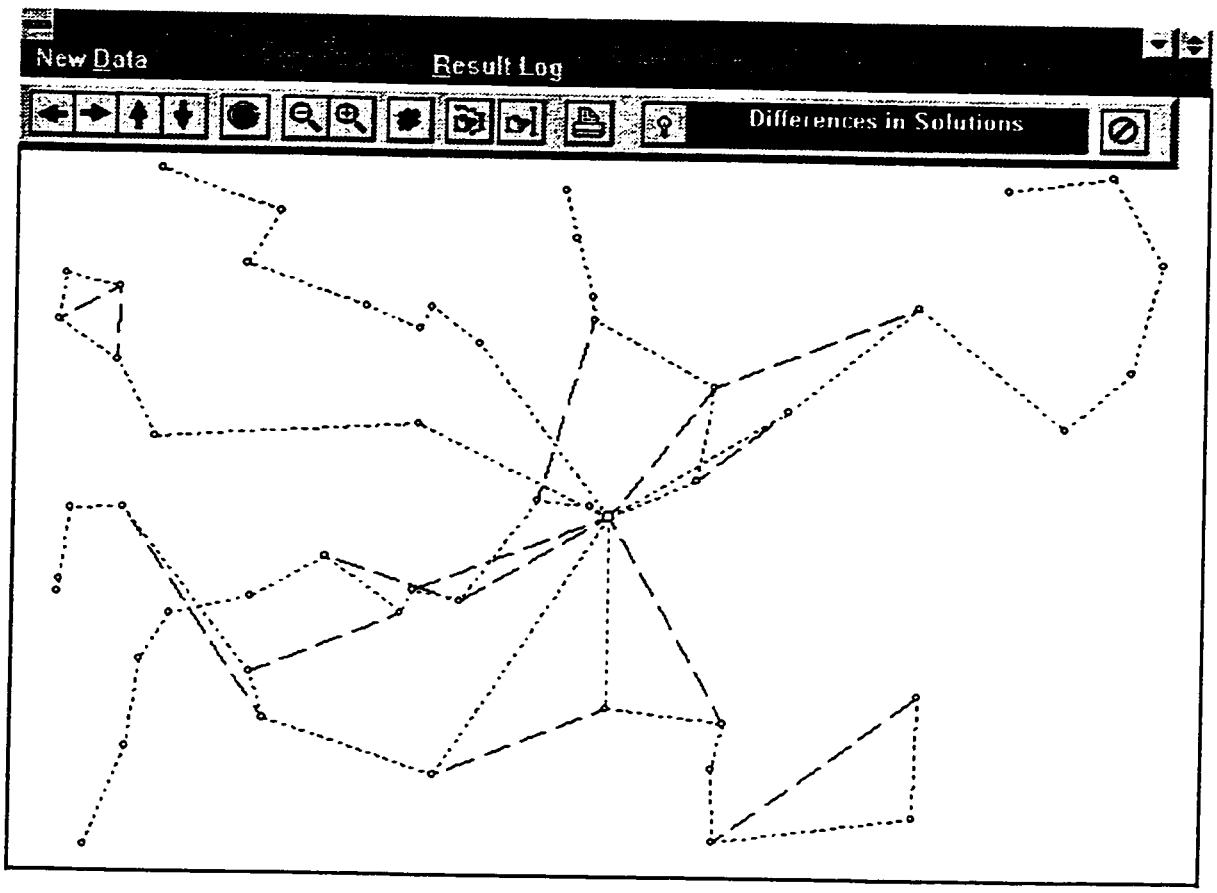


figure A11. Example of the program's capability to display the differences between the initial and SA solutions.

In the illustration above, the dashes represent the Esau-Williams solution and the dotted lines superimposed on top of it represent the SA solution.

A tool bar appears that allows the user to easily manipulate the display. It consists of various buttons. From left to right, we first find 4 arrow buttons and a bull's eye.

figure A12. Buttons of the toolbar used in positioning the graphical display.

The arrows allow the user to move the graphical display in 4 different direction. This capability is particularly useful when the data set consists of a large number of terminals and they do not fit the screen all at once. The bull's eye allows the user to automatically center the display back onto the concentrator (the default setting).

The next set of tools are given below. The magnifying glasses allows the user to zoom in or out (increasing or decreasing the display size). The '#' button numbers all the terminals of the solution. The next two buttons enable the user to either find a specific terminal or a specific line in the topology.

figure A13. The zoom buttons, terminal numbering buttons, and the 'find' terminal and line buttons.

Finally, printing capabilities were added in order to give the user the ability to output a hard copy of the solutions.

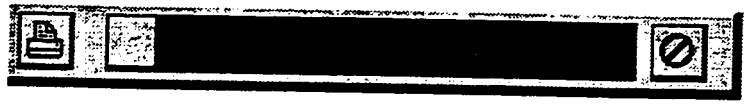


figure A14. The print button, the solution button, and the exit button.

The middle button above toggles between the initial solution display and the SA solution display, while the last button to the right exits the graphics mode and returns the user to the previous screen.

Finally, examples of the graphical output for the other topologies are given below and in the following pages.

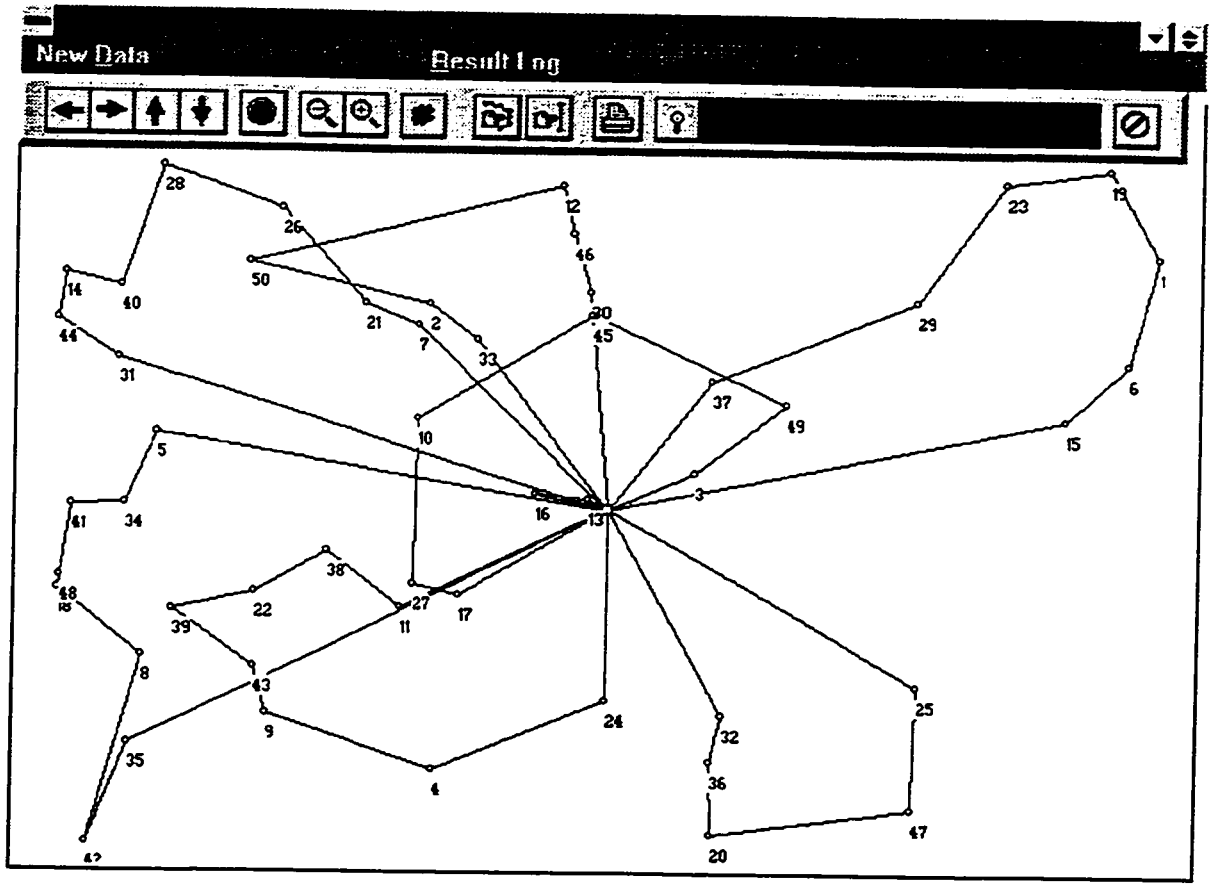


figure A15. Example of graphical output for the loop topology.(initial solution)

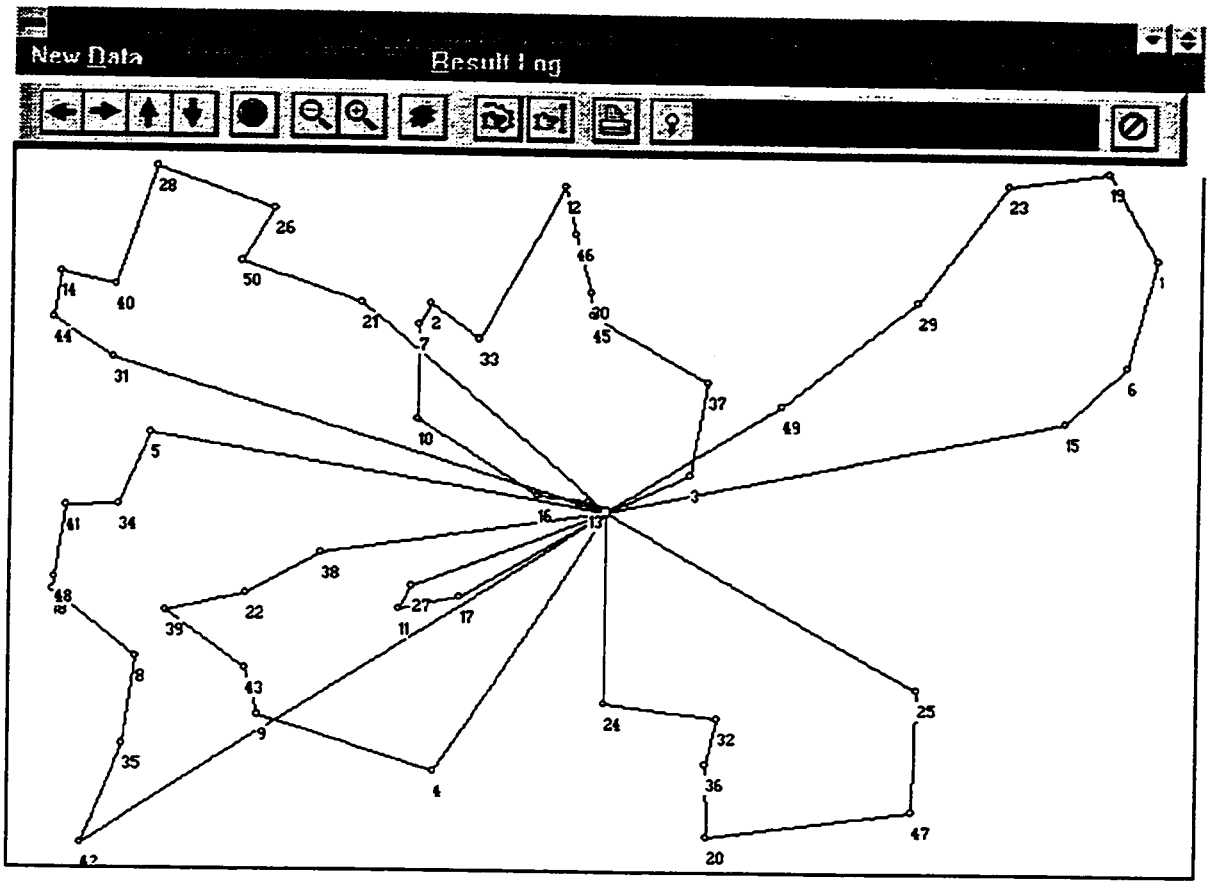


figure A16. Example of graphical output for the loop topology.(SA solution)

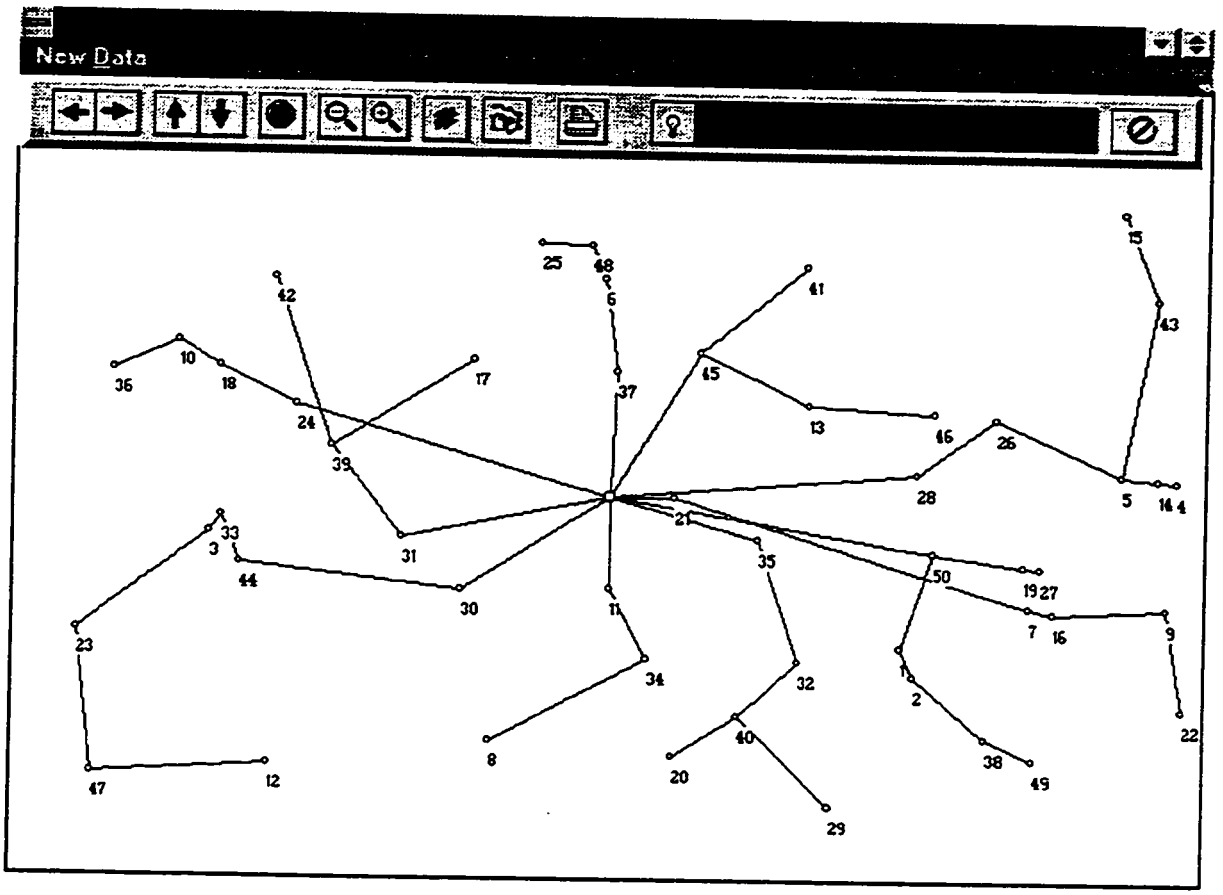


figure A17. Example of graphical output for the tree topology.(initial solution)

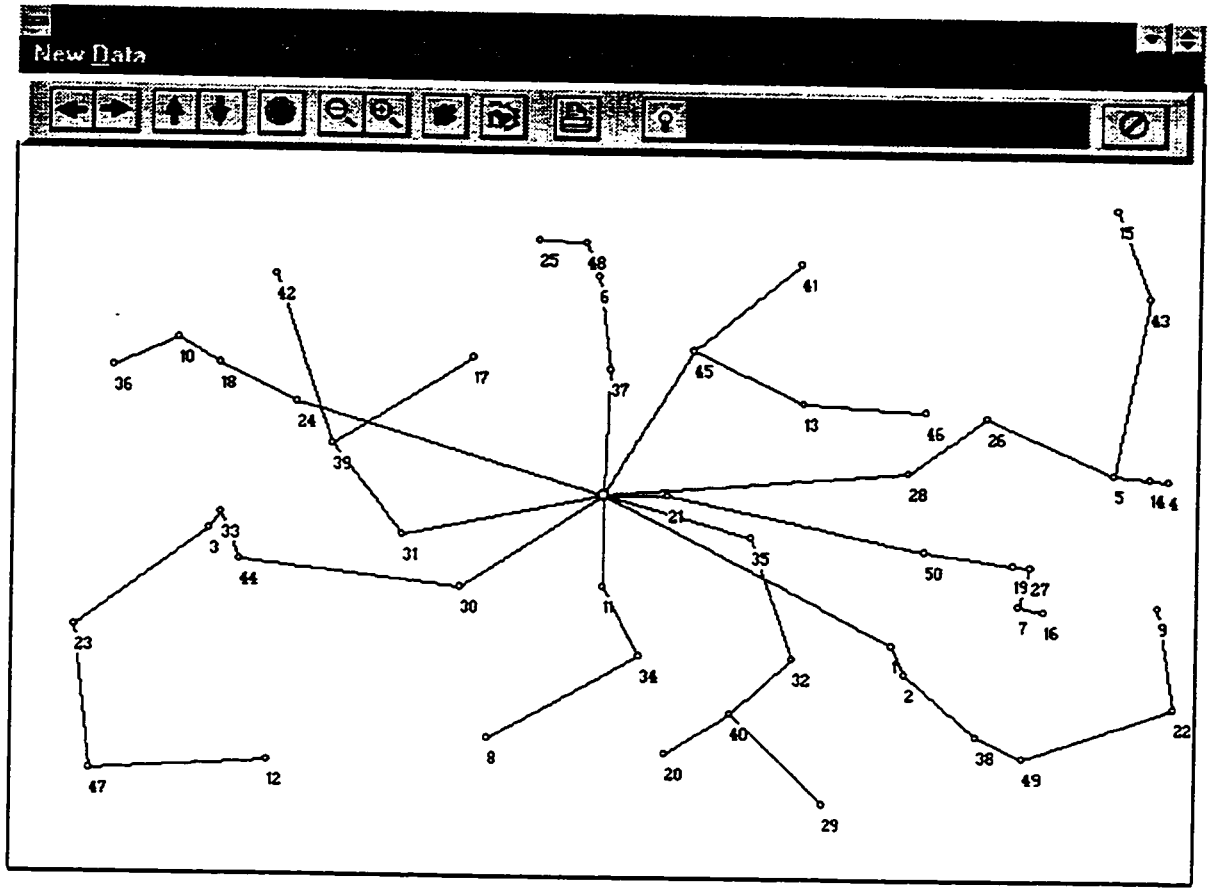
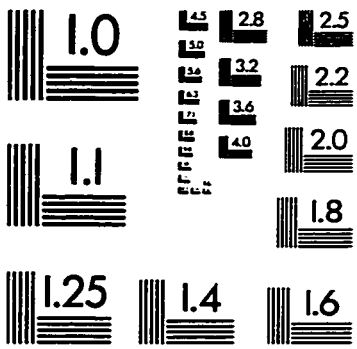
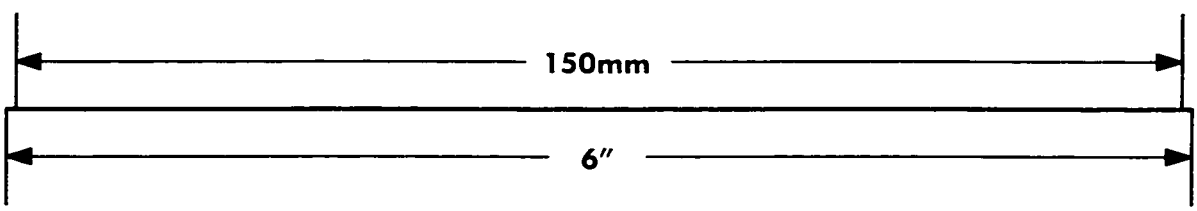


figure A18. Example of graphical output for the tree topology.(SA solution)





APPLIED IMAGE, Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989